GUIA DE REFERÊNCIA

Índice

23. *Scroll in to view e hidden*
24. *Geolocation API*
25. *Undo*

Sobre esse conteúdo

Nós fomos escolhidos pelo W3C do Brasil para ministrar um treinamento sobre HTML5 para os seus membros e alguns convidados. Para tanto, construímos uma apostila com todo o conteúdo abordado neste nosso curso. Essa apostila está sendo agora publicada sob Creative Commons aqui no Tableless. Esperamos que ajude a comunidade de desenvolvimento web brasileira.

Essa apostila ficará em processo de constante atualização já que muitos pontos do HTML5 não foram ainda definidos e também porque diversas outras características estão sendo planejadas e rascunhadas ainda. Se você tiver qualquer sugestão, por favor, nos contacte.

VISÃO GERAL DO HTML5

De acordo com o W3C a Web é baseada em 3 pilares:

- *Um esquema de nomes para localização de fontes de informação na Web, esse esquema chama-se URI.*
- *Um Protocolo de acesso para acessar estas fontes, hoje o HTTP.*
- *Uma linguagem de Hypertexto, para a fácil navegação entre as fontes de informação: o HTML.*

Vamos nos focar no terceiro pilar, o HTML.

Hypertexto

HTML é uma abreviação de Hypertext Markup Language - Linguagem de Marcação de Hypertexto. Resumindo em uma frase: o HTML é uma linguagem para publicação de conteúdo (texto, imagem, vídeo, áudio e etc) na Web.

O HTML é baseado no conceito de Hipertexto. Hipertexto são conjuntos de elementos – ou **nós** – ligados por conexões. Estes elementos podem ser palavras, imagens, vídeos, áudio, documentos etc. Estes elementos conectados formam uma grande rede de informação. Eles não estão conectados linearmente como se fossem textos de um livro, onde um assunto é ligado ao outro seguidamente. A conexão feita em um hipertexto é algo imprevisto que permite a comunicação de dados, organizando conhecimentos e guardando informações relacionadas.

Para distribuir informação de uma maneira global, é necessário haver uma linguagem que seja entendida universalmente por diversos meios de acesso. O HTML se propõe a ser esta linguagem.
Desenvolvido originalmente por Tim Berners-Lee o HTML ganhou popularidade quando o Mosaic - browser desenvolvido por Marc Andreessen na década de 1990 - ganhou força. A partir daí, desenvolvedores e fabricantes de browsers utilizaram o HTML como base, compartilhando as mesmas convenções.

O começo e a interoperabilidade

Entre 1993 e 1995, o HTML ganhou as versões HTML+, HTML2.0 e HTML3.0, onde foram propostas diversas mudanças para enriquecer as possibilidades da linguagem. Contudo, até aqui o HTML ainda não era tratado como um padrão. Apenas em 1997, o grupo de trabalho do W3C responsável por manter o padrão do código, trabalhou na versão 3.2 da linguagem, fazendo com que ela fosse tratada como prática comum. Você pode ver: http://www.w3.org/TR/html401/appendix/changes.html.

Desde o começo o HTML foi criado para ser uma linguagem independente de plataformas, browsers e outros meios de acesso. Interoperabilidade significa menos custo. Você cria apenas um código HTML e este código pode ser lido por diversos meios, ao invés de versões diferentes para diversos dispositivos. Dessa forma, evitou-se que a Web fosse desenvolvida em uma base proprietária, com formatos incompatíveis e limitada.

Por isso o HTML foi desenvolvido para que essa barreira fosse ultrapassada, fazendo com que a informação publicada por meio deste código fosse acessível por dispositivos e outros meios com características diferentes, não importando o tamanho da tela, resolução, variação de cor. Dispositivos próprios para deficientes visuais e auditivos ou dispositivos móveis e portáteis. O HTML deve ser entendido universalmente, dando a possibilidade para a reutilização dessa

informação de acordo com as limitações de cada meio de acesso.

WHAT Working Group

Enquanto o W3C focava suas atenções para a criação da segunda versão do XHTML, um grupo chamado Web Hypertext Application Technology Working Group ou WHATWG trabalhava em uma versão do HTML que trazia mais flexibilidade para a produção de websites e sistemas baseados na web.

O WHATWG (http://www.whatwg.org/) foi fundado por desenvolvedores de empresas como Mozilla, Apple e Opera em 2004. Eles não estavam felizes com o caminho que a Web tomava e nem com o rumo dado ao XHTML. Por isso, estas organizações se juntaram para escrever o que seria chamado hoje de HTML5.
Entre outros assuntos que o WHATWG se focava era Web Forms 2.0 que foi incluído no HTML5 e o Web Controls 1.0 que foi abandonado por enquanto.

A participação no grupo é livre e você pode se inscrever na lista de email para contribuir.

Por volta de 2006, o trabalho do WHATWG passou ser conhecido pelo mundo e principalmente pelo W3C - que até então trabalhavam separadamente - que reconheceu todo o trabalho do grupo. Em Outubro de 2006, Tim Berners-Lee anunciou que trabalharia juntamente com o WHATWG na produção do HTML5 em detrimento do XHTML 2. Contudo o XHTML continuaria sendo mantido paralelamente de acordo comas mudanças causadas no HTML. O grupo que estava cuidando especificamente do XHTML 2 foi descontinuado em 2009.

O HTML5 e suas mudanças

Quando o HTML4 foi lançado, o W3C alertou os desenvolvedores sobre algumas boas práticas que deveriam ser seguidas ao produzir códigos client-side. Desde este tempo, assuntos como a separação da estrutura do código com a formatação e princípios de acessibilidade foram trazidos para discussões e à atenção dos fabricantes e desenvolvedores.

Contudo, o HTML4 ainda não trazia diferencial real para a semântica do código. o HTML4 também não facilitava a manipulação dos elementos via Javascript ou CSS. Se você quisesse criar um sistema com a possibilidade de Drag'n Drop de elementos, era necessário criar um grande script, com bugs e que muitas vezes não funcionavam de acordo em todos os browsers.

O que é o HTML5?

O HTML5 é a nova versão do HTML4. Enquanto o WHATWG define as regras de marcação que usaremos no HTML5 e no XHTML, eles também definem APIs que formarão a base da arquitetura web. Essas APIs são conhecidas como **DOM Level 0**.
Um dos principais objetivos do HTML5 é facilitar a manipulação do elemento possibilitando o desenvolvedor a modificar as características dos objetos de forma não intrusiva e de maneira que seja transparente para o usuário final.

Ao contrário das versões anteriores, o HTML5 fornece ferramentas para a CSS e o Javascript fazerem seu trabalho da melhor maneira possível. O HTML5 permite por meio de suas APIs a manipulação das características destes elementos, de forma que o website ou a aplicação continue leve e funcional.

O HTML5 também cria novas tags e modifica a função de outras. As versões antigas do HTML não continham um padrão universal para a criação de seções comuns e específicas como rodapé, cabeçalho, sidebar, menus e etc. Não havia um padrão de nomenclatura de IDs, Classes ou tags. Não havia um método de capturar de maneira automática as informações localizadas nos rodapés dos websites.

Há outros elementos e atributos que sua função e significado foram modificados e que agora podem ser reutilizados de forma mais eficaz. Por exemplo, elementos como B ou I que foram descontinuados em versões anteriores do HTML agora assumem funções diferentes e entregam mais significado para os usuários.

O HTML5 modifica a forma de como escrevemos código e organizamos a informação na página. Seria mais semântica com menos código. Seria mais interatividade sem a necessidade de instalação de plugins e perda de performance. É a criação de código interoperável, pronto para futuros dispositivos e que facilita a reutilização da informação de diversas formas.

O WHATWG tem mantido o foco para manter a retrocompatibilidade. Nenhum site deverá ter de ser refeito totalmente para se adequar aos novos conceitos e regras. O HTML5 está sendo criado para que seja compatível com os browsers recentes, possibilitando a utilização das novas características imediatamente.

ANÁLISE DO SUPORTE ATUAL PELOS NAVEGADORES E ESTRATÉGIAS DE USO

O desenvolvimento modular

Antigamente, para que uma nova versão do HTML ou do CSS fosse lançada, todas as ideias listadas na especificação deveriam ser testadas e desenvolvidas para então serem publicadas para o uso dos browsers e os desenvolvedores.
Esse método foi mudado com o lançamento do HTML5 e o CSS3. A partir de agora, as duas tecnologias foram divididas em módulos. Isso quer dizer que a comunidade de desenvolvedores e os fabricantes de browsers não precisam esperar que todo o padrão seja escrito e publicado para utilizarem as novidades das linguagens.
As propriedades do CSS3, por exemplo, foram divididas em pequenos grupos. Há um grupo cuidando da propriedade Background, outro da propriedade Border, outro das propriedades de Texto e etc. Cada um destes grupos são independentes e podem lançar suas novidades a qualquer momento. Logo, o desenvolvimento para web ficou mais dinâmico, com novidades mais constantes.

O ponto negativo nesse formato, é que problemas de compatibilidade podem ocorrer com mais frequencia. Por exemplo, um browser pode adotar bordas arredondadas e outro não. Ou um browser pode escolher suportar um API diferente do API que o concorrente implementou. Contudo, os browsers tem mostrado grande interesse em se manterem atualizados em relação aos seus concorrentes.

Motores de Renderização

Há uma grande diversidade de dispositivos que acessam a internet. Entre eles, há uma série de tablets, smartphones, pc's e etc. Cada um destes meios de acesso utilizam um determinado browser para navegar na web. Não há como os desenvolvedores manterem um bom nível de compatibilidade com todos estes browsers levando em consideração a particularidade de cada um. Uma maneira mais segura de manter o código compatível, é nivelar o desenvolvimento pelos motores de renderização. Cada browser utiliza um motor de renderização que é responsável pelo processamento do código da página.

Abaixo, segue uma lista dos principais browsers e seus motores:

Motor	*Browser*
Webkit	Safari, Google Chrome
Gecko	Firefox, Mozilla, Camino
Trident	Internet Explorer 4 ao 9
Presto	Opera 7 ao 10

É interessante que você faça código compatível com estes motores. Focando a compatibilidade nos motores de renderização você atingirá uma amplitude maior de browsers.
Por exemplo, se seu código funcionar no Webkit, você alcançará o Safari e o Chrome, dois dos principais browsers do mercado para desktops. Além disso, você também alcança aparelhos como Blackberry, iPhone, iPod Touch, iPad e dispositivos que rodam Android.

Compatibilidade com HTML5

Atualmente o Webkit é o motor mais compatível com os Padrões do HTML5. Como a Apple tem interesse que seus dispositivos sejam ultracompatíveis com os Padrões, ela tem feito um belo trabalho de atualização e avanço da compatibilidade deste motor.
Contudo o Firefox e o Opera já estão compatíveis com grande parte da especificação do HTML5 e CSS3 e a cada upgrade eles trazem mais novidades e atualização dos padrões.

O que pode te preocupar de verdade é a retrocompatibilidade com versões antigas de browsers como o Internet Explorer. A Microsoft está fazendo um bom trabalho com o IE9, mas as versões 8 e 7 não tem quase nenhum suporte ao HTML5, o que é um problema sério para aplicações web baseadas em tecnologias mais recentes, mas que a base de usuários utiliza as versões antigas do Internet Explorer.

Abaixo segue uma tabela simples de compatibilidade entre os browsers e alguns módulos do HTML5:

	Safari	Chrome	Opera	Firefox	IE 8	IE 9
Local Storage						
Histórico de Sessão						
Aplicações Offline						
Novos tipos de campos						
Form: Autofocus						
Form: Autocomplete						
Form: Required						
Video, Audio e Canvas Text						

Técnicas de detecção

Pode ser que o usuário não utilize um browser que suporta HTML5. Neste caso, você pode redirecioná-lo para uma versão do site mais simples, ou talvez apenas mostrar uma mensagem alertando o usuário sobre a importância da atualização do browser. Para isso temos algumas técnicas de detecção para conferir se o browser suporta ou não HTML5.

Quando o browser visita um website, ele constrói uma coleção de objetos que representam elementos HTML na página. Cada elemento no código é representado no DOM como um objeto diferente. Todo objeto DOM tem propriedades em comum, mas alguns objetos tem características específicas. Usaremos estes objetos para fazermos a detecção. Abaixo segue 4 meios que você poderá utilizar para detectar o suporte do browser:

1. *Verifique se uma determinada propriedade existe em objetos globais como WINDOW ou NAVIGATOR. Nesse caso, verificamos o suporte a geolocalização.*
2. *Crie um elemento e verifique se uma determinada propriedade existe neste elemento.*
3. *Crie um elemento e verifique se um determinado método existe neste elemento, então chame o método e verifique se o valor retorna. Por exemplo, teste quais formatos de vídeo são suportados.*
4. *Crie um elemento e defina um atributo com um determinado valor, então verifique se o atributo suporta este valor. Por exemplo, crie um input e verifique quais types são suportados.*

Utilizando o Modernizr

O Modernizr (http://www.modernizr.com/) é uma biblioteca de detecção que lhe permite verificar o suporte da maioria das características do HTML5 e CSS3.

O Modernizr roda automaticamente assim que você o adiciona no head do documento. Assim, se você quiser verificar se o browser suporta Geolocalização, por exemplo, basta inserir este

script na página. Se o browser suportar a feature testada, ele retornará true:

```
if (Modernizr.geolocation) {
  // Aceita a feature
} else {
  // Não aceita a feature testada.
}
```

ESTRUTURA BÁSICA, DOCTYPE E CHARSETS

A estrutura básica do HTML5 continua sendo a mesma das versões anteriores da linguagem, há apenas uma excessão na escrita do Doctype. Segue abaixo como a estrutura básica pode ser seguida:

Arquivo: exemplos/3/estruturabasica.html

```
<!DOCTYPE HTML>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">
    <link rel="stylesheet" type="text/css" href="estilo.css">
    <title></title>
</head>
<body>
    <p>Estrutura básica de um HTML</p>

<pre>
&lt;!DOCTYPE HTML&gt;<br>
&lt;html lang="pt-br"&gt;<br>
&lt;head&gt;<br>
    &lt;meta charset="UTF-8"&gt;<br>
    &lt;link rel="stylesheet" type="text/css" href="estilo.css"&gt;<br>
    &lt;title&gt;&lt;/title&gt;<br>
&lt;/head&gt;<br>
&lt;body&gt;<br>

&lt;/body&gt;<br>
&lt;/html&gt;<br>
</pre>
<a href="javascript: history.go(-1)">Voltar para o artigo</a>

</body>
</html>
```

O Doctype

O Doctype deve ser a primeira linha de código do documento antes da tag HTML.

```
<!DOCTYPE html>
```

O Doctype indica para o navegador e para outros meios qual a especificação de código utilizar. Em versões anteriores, era necessário referenciar o DTD diretamente no código do Doctype. Com o HTML5, a referência por qual DTD utilizar é responsabilidade do Browser.

O Doctype não é uma tag do HTML, mas uma instrução para que o browser tenha informações sobre qual versão de código a marcação foi escrita.

O elemento HTML

O código HTML é uma série de elementos em árvore onde alguns elementos são filhos de outros e assim por diante. O elemento principal dessa grande árvore é sempre a tag HTML.

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

O atributo LANG é necessário para que os user-agents saibam qual a linguagem principal do documento.
Lembre-se que o atributo LANG não é restrito ao elemento HTML, ele pode ser utilizado em qualquer outro elemento para indicar o idioma do texto representado.

Para encontrar a listagem de códigos das linguagens, acesse: http://www.w3.org/International

/questions/qa-choosing-language-tags.

HEAD

A Tag HEAD é onde fica toda a parte inteligente da página. No HEAD ficam os metadados. Metadados são informações sobre a página e o conteúdo ali publicado.

Metatag Charset

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

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

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

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

Essa forma antiga será também suportada no HTML5. Contudo, é melhor que você utilize a nova forma.

A Web é acessada por pessoas do mundo inteiro. Ter um sistema ou um site que limite o acesso e pessoas de outros países é algo que vai contra a tradição e os ideais da internet. Por isso, foi criado uma tabela que suprisse essas necessidades, essa tabela se chama Unicode. A tabela Unicode suporta algo em torno de um milhão de caracteres. Ao invés de cada região ter sua tabela de caracteres, é muito mais sensato haver uma tabela padrão com o maior número de caracteres possível. Atualmente a maioria dos sistemas e browsers utilizados por usuários suportam plenamente Unicode. Por isso, fazendo seu sistema Unicode você garante que ele será bem visualizado aqui, na China ou em qualquer outro lugar do mundo.

O que o Unicode faz é fornecer um único número para cada caractere, não importa a plataforma, nem o programa, nem a língua.

Tag LINK

Há dois tipos de links no HTML: a tag A, que são links que levam o usuário para outros documentos e a tag LINK, que são links para fontes externas que serão usadas no documento.

No nosso exemplo há uma tag LINK que importa o CSS para nossa página:

```
<link rel="stylesheet" type="text/css" href="estilo.css">
```

O atributo `rel="stylesheet"` indica que aquele link é relativo a importação de um arquivo referente a folhas de estilo.

Há outros valores para o atributo REL, como por exemplo o ALTERNATE:

```
<link rel="alternate" type="application/atom+xml" title="feed" href="/feed/">
```

Neste caso, indicamos aos user-agents que o conteúdo do site poder ser encontrado em um caminho alternativo via Atom FEED.

No HTML5 há outros links relativos que você pode inserir como o `rel="archives"` que indica uma referência a uma coleção de material histórico da página. Por exemplo, a página de histórico de um blog pode ser referenciada nesta tag.

MODELOS DE CONTEÚDO

Há pequenas regras básicas que nós já conhecemos e que estão no HTML desde o início. Estas regras definem onde os elementos podem ou não estar. Se eles podem ser filhos ou pais de outros elementos e quais os seus comportamentos.

Dentre todas as categorias de modelos de conteúdo, existem dois tipos de elementos: elementos de linha e de bloco.
Os elementos de linha marcam, na sua maioria das vezes, texto. Alguns exemplos: `a`, `strong`, `em`, `img`, `input`, `abbr`, `span`.

Os elementos de blocos são como caixas, que dividem o conteúdo nas seções do layout.

Abaixo segue algumas premissas que você precisa relembrar e conhecer:

- *Os elementos de linha podem conter outros elementos de linha, dependendo da categoria que ele se encontra. Por exemplo: o elemento `a` não pode conter o elemento `label`.*
- *Os elementos de linha nunca podem conter elementos de bloco.*
- *Elementos de bloco sempre podem conter elementos de linha.*
- *Elementos de bloco podem conter elementos de bloco, dependendo da categoria que ele se encontra. Por exemplo, um parágrafo não pode conter um DIV. Mas o contrário é possível.*

Estes dois grandes grupos podem ser divididos em categorias. Estas categorias dizem qual modelo de conteúdo o elemento trabalha e como pode ser seu comportamento.

Categorias

Cada elemento no HTML pode ou não fazer parte de um grupo de elementos com características similares. As categorias estão a seguir. Manteremos os nomes das categorias em inglês para que haja um melhor entendimento:

- *Metadata content*
- *Flow content*
- *Sectioning content*
- *Heading content*
- *Phrasing content*
- *Embedded content*
- *Interactive content*

Abaixo segue como as categorias estão relacionadas de acordo com o WHATWG:

Metadata content

Os elementos que compõe a categoria Metadata são:

- *base*
- *command*
- *link*
- *meta*
- *noscript*
- *script*
- *style*
- *title*

Este conteúdo vem antes da apresentação, formando uma relação com o documento e seu conteúdo com outros documentos que distribuem informação por outros meios.

Flow content

A maioria dos elementos utilizados no body e aplicações são categorizados como Flow Content. São eles:

- *a*
- *abbr*
- *address*
- *area (se for um decendente de um elemento de mapa)*
- *article*
- *aside*
- *audio*
- *b*
- *bdo*
- *blockquote*
- *br*
- *button*
- *canvas*
- *cite*
- *code*
- *command*
- *datalist*
- *del*
- *details*
- *dfn*
- *div*
- *dl*
- *em*
- *embed*
- *fieldset*
- *figure*
- *footer*
- *form*
- *h1*
- *h2*
- *h3*
- *h4*
- *h5*

- *h6*
- *header*
- *hgroup*
- *hr*
- *i*
- *iframe*
- *img*
- *input*
- *ins*
- *kbd*
- *keygen*
- *label*
- *link (Se o atributo `itemprop` for utilizado)*
- *map*
- *mark*
- *math*
- *menu*
- *meta (Se o atributo `itemprop` for utilizado)*
- *meter*
- *nav*
- *noscript*
- *object*
- *ol*
- *output*
- *p*
- *pre*
- *progress*
- *q*
- *ruby*
- *samp*
- *script*
- *section*
- *select*
- *small*
- *span*
- *strong*
- *style (Se o atributo `scoped` for utilizado)*
- *sub*
- *sup*
- *svg*
- *table*
- *textarea*
- *time*
- *ul*
- *var*
- *video*
- *wbr*
- *Text*

Por via de regra, elementos que seu modelo de conteúdo permitem inserir qualquer elemento que se encaixa no Flow Content, devem ter pelo menos um descendente de texto ou um elemento descendente que faça parte da categoria `embedded`.

Sectioning content

Estes elementos definem um grupo de cabeçalhos e rodapés.

- *article*
- *aside*
- *nav*
- *section*

Basicamente são elementos que juntam grupos de textos no documento.

Heading content

Os elementos da categoria Heading definem uma seção de cabeçalhos, que podem estar contidos em um elemento na categoria Sectioning.

- *h1*
- *h2*
- *h3*
- *h4*
- *h5*
- *h6*
- *hgroup*

Phrasing content

Fazem parte desta categoria elementos que marcam o texto do documento, bem como os elementos que marcam este texto dentro do elemento de parágrafo.

- *a*
- *abbr*
- *area (se ele for descendente de um elemento de mapa)*
- *audio*
- *b*
- *bdo*
- *br*
- *button*
- *canvas*
- *cite*
- *code*
- *command*
- *datalist*
- *del (se ele contiver um elemento da categoria de Phrasing)*
- *dfn*
- *em*
- *embed*
- *i*
- *iframe*
- *img*
- *input*
- *ins (se ele contiver um elemento da categoria de Phrasing)*
- *kbd*
- *keygen*
- *label*
- *link (se o atributo `itemprop` for utilizado)*

- *map (se apenas ele contiver um elemento da categoria de Phrasing)*
- *mark*
- *math*
- *meta (se o atributo `itemprop` for utilizado)*
- *meter*
- *noscript*
- *object*
- *output*
- *progress*
- *q*
- *ruby*
- *samp*
- *script*
- *select*
- *small*
- *span*
- *strong*
- *sub*
- *sup*
- *svg*
- *textarea*
- *time*
- *var*
- *video*
- *wbr*
- *Text*

Embedded content

Na categoria Embedded, há elementos que importam outra fonte de informação para o documento.

- *audio*
- *canvas*
- *embed*
- *iframe*
- *img*
- *math*
- *object*
- *svg*
- *video*

Interactive content

Interactive Content são elementos que fazem parte da interação de usuário.

- *a*
- *audio (se o atributo `control` for utilizado)*
- *button*
- *details*
- *embed*
- *iframe*
- *img (se o atributo `usemap` for utilizado)*

- *input (se o atributo `type` não tiver o valor hidden)*
- *keygen*
- *label*
- *menu (se o atributo `type` tiver o valor `toolbar`)*
- *object (se o atributo `usemap` for utilizado)*
- *select*
- *textarea*
- *video (se o atributo `control` for utilizado)*

Alguns elementos no HTML podem ser ativados por um comportamento. Isso significa que o usuário pode ativá-lo de alguma forma. O início da sequencia de eventos depende do mecanismo de ativação e normalmente culminam em um evento de click seguido pelo evento DOMActivate.

O user-agent permite que o usuário ative manualmente o elemento que tem este comportamento utilizando um teclado, mouse, comando de voz etc.

NOVOS ELEMENTOS E ATRIBUTOS

A função do HTML é indicar que tipo de informação a página está exibindo. Quando lemos um livro, conseguimos entender e diferenciar um título de um parágrafo. Basta percebermos a quantidade de letra, tamanho da fonte, cor etc. No código isso é diferente. Robôs de busca e outros user-agents não conseguem diferenciar tais detalhes. Por isso, cabe ao desenvolvedor marcar a informação para que elas possam ser diferenciadas por diversos dispositivos.

Com as versões anteriores do HTML nós conseguimos marcar diversos elementos do layout, estruturando a página de forma que as informações ficassem em suas áreas específicas. Conseguiámos diferenciar por exemplo, um parágrafo de um título. Mas não conseguíamos diferenciar o rodapé do cabeçalho. Essa diferenciação era apenas percebida visualmente pelo layout pronto ou pela posição dos elementos na estrutura do HTML. Entretanto, não havia maneira de detectar automaticamente estes elementos já que as tags utilizada para ambos poderiam ser iguais e não havia padrão para nomenclatura de IDs e Classes.

O HTML5 trouxe uma série de elementos que nos ajudam a definir setores principais no documento HTML. Com a ajuda destes elementos, podemos por exemplo diferenciar diretamente pelo código HTML5 áreas importantes do site como sidebar, rodapé e cabeçalho. Conseguimos seccionar a área de conteúdo indicando onde exatamente é o texto do artigo.

Estas mudanças simplificam o trabalho de sistemas como os dos buscadores. Com o HTML5 os buscadores conseguem vasculhar o código de maneira mais eficaz. Procurando e guardando informações mais exatas e levando menos tempo para estocar essa informação.

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

section	A tag `section` define uma nova seção genérica no documento. Por exemplo, a home de um website pode ser dividida em diversas seções: introdução ou destaque, novidades, informação de contato e chamadas para conteúdo interno.
nav	O elemento `nav` representa uma seção da página que contém links para outras partes do website. Nem todos os grupos de links devem ser elementos `nav`, apenas aqueles grupos que contém links importantes. Isso pode ser aplicado naqueles blocos de links que geralmente são colocados no Rodapé e também para compor o menu principal do site.
article	O elemento `article` representa uma parte da página que poderá ser distribuído e reutilizável em FEEDs por exemplo. Isto pode ser um post, artigo, um bloco de comentários de usuários ou apenas um bloco de texto comum.
aside	O elemento `aside` representa um bloco de conteúdo que referência o conteúdo que envolta do elemento `aside`. O `aside` pode ser representado por conteúdos em sidebars em textos impressos, publicidade ou até mesmo para criar um grupo de elementos `nav` e outras informações separados do conteúdo principal do website.
header	O elemento `header` representa um grupo de introdução ou elementos de navegação. O elemento `header` pode ser utilizado para agrupar índices de conteúdos, campos de busca ou até mesmo logos.
footer	O elemento `footer` representa literalmente o rodapé da página. Seria o último elemento do último elemento antes de fechar a tag HTML. O elemento `footer` não precisa aparecer necessariamente no final de uma seção.
time	Este elemento serve para marcar parte do texto que exibe um horário ou uma data precisa no calendário gregoriano.

Este atributos foram descontinuados porque modificam a formatação do elemento e suas funções são melhores controladas pelo CSS:

- `align` *como atributo da tag* `caption`, `iframe`, `img`, `input`, `object`, `legend`, `table`, `hr`, `div`, `h1`, `h2`, `h3`, `h4`, `h5`, `h6`, `p`, `col`, `colgroup`, `tbody`, `td`, `tfoot`, `th`, `thead` *e* `tr`.
- `alink`, `link`, `text` *e* `vlink` *como atributos da tag* `body`.
- `background` *como atributo da tag* `body`.
- `bgcolor` *como atributo da tag* `table`, `tr`, `td`, `th` *e* `body`.
- `border` *como atributo da tag* `table` *e* `object`.
- `cellpadding` *e* `cellspacing` *como atributos da tag* `table`.

- `char` e `charoff` *como atributos da tag* `col`, `colgroup`, `tbody`, `td`, `tfoot`, `th`, `thead` e `tr`.
- `clear` *como atributo da tag* `br`.
- `compact` *como atributo da tag* `dl`, `menu`, `ol` e `ul`.
- `frame` *como atributo da tag* `table`.
- `frameborder` *como atributo da tag* `iframe`.
- `height` *como atributo da tag* `td` e `th`.
- `hspace` e `vspace` *como atributos da tag* `img` e `object`.
- `marginheight` e `marginwidth` *como atributos da tag* `iframe`.
- `noshade` *como atributo da tag* `hr`.
- `nowrap` *como atributo da tag* `td` e `th`.
- `rules` *como atributo da tag* `table`.
- `scrolling` *como atributo da tag* `iframe`.
- `size` *como atributo da tag* `hr`.
- `type` *como atributo da tag* `li`, `ol` e `ul`.
- `valign` *como atributo da tag* `col`, `colgroup`, `tbody`, `td`, `tfoot`, `th`, `thead` e `tr`.
- `width` *como atributo da tag* `hr`, `table`, `td`, `th`, `col`, `colgroup` e `pre`.

Alguns atributos do HTML4 não são mais permitidos no HTML5. Se eles tiverem algum impacto negativo na compatibilidade de algum user-agent eles serão discutidos.

- `rev` e `charset` *como atributos da tag* `link` e `a`.
- `shape` e `coords` *como atributos da tag* `a`.
- `longdesc` *como atributo da tag* `img` *and* `iframe`.
- `target` *como atributo da tag* `link`.
- `nohref` *como atributo da tag* `area`.
- `profile` *como atributo da tag* `head`.
- `version` *como atributo da tag* `html`.
- `name` *como atributo da tag* `img` *(use* `id` *instead).*
- `scheme` *como atributo da tag* `meta`.
- `archive`, `classid`, `codebase`, `codetype`, `declare` e `standby` *como atributos da tag* `object`.
- `valuetype` e `type` *como atributos da tag* `param`.
- `axis` e `abbr` *como atributos da tag* `td` e `th`.
- `scope` *como atributo da tag* `td`.

Atributos

Alguns elementos ganharam novos atributos:

- *O atributo* `autofocus` *pode ser especificado nos elementos* `input` *(exceto quando há atributo* `hidden` *atribuído),* `textarea`, `select` *e* `button`.
- *A tag* `a` *passa a suportar o atributo* `media` *como a tag* `link`.
- *A tag* `form` *ganha um atributo chamado* `novalidate`. *Quando aplicado o formulário pode ser enviado sem validação de dados.*
- *O elemento* `ol` *ganhou um atributo chamado* `reversed`. *Quando ele é aplicado os indicadores da lista são colocados na ordem inversa, isto é, da forma descendente.*
- *O elemento* `fieldset` *agora permite o atributo* `disabled`. *Quando aplicado, todos os filhos de* `fieldset` *são desativados.*
- *O novo atributo* `placeholder` *pode ser colocado em* `inputs` *e* `textareas`.
- *O elemento* `area` *agora suporta os atributos* `hreflang` *e* `rel` *como os elementos* `a` *e* `link`
- *O elemento* `base` *agora suporta o atributo* `target` *assim como o elemento* `a`. *O atributo* `target` *também não está mais descontinuado nos elementos* `a` *e* `area` *porque são úteis para aplicações web.*

Os atributos abaixo foram descontinuados:

- *O atributo* `border` *utilizado na tag* `img`.

- *O atributo* `language` *na tag* `script`.
- *O atributo* `name` *na tag* `a`. *Porque os desenvolvedores utilizam ID em vez de name.*
- *O atributo* `summary` *na tag* `table`.

O W3C mantém um documento atualizado constantemente nesta página: http://www.w3.org/TR/2010/WD-html5-diff-20100624/.

ELEMENTOS MODIFICADOS E AUSENTES

Existiam no HTML alguns elementos que traziam apenas características visuais e não semânticas para o conteúdo da página. Esses elementos anteriormente foram descontinuados porque atrapalhavam o código e também porque sua função era facilmente suprida pelo CSS. Contudo, alguns destes elementos voltaram à tona com novos significados semânticos. Outros elementos que não descontinuados, mas seus significados foram modificados.

Elementos modificados

- *O elemento B passa a ter o mesmo nível semântico que um SPAN, mas ainda mantém o estilo de negrito no texto. Contudo, ele não dá nenhuma importância para o text marcado com ele.*
- *O elemento I também passa a ser um SPAN. O texto continua sendo itálico e para usuários de leitores de tela, a voz utilizada é modificada para indicar ênfase. Isso pode ser útil para marcar frases em outros idiomas, termos técnicos e etc.*

O interessante é que nestes dois casos houve apenas uma mudança semântica. Provavelmente você não precisará modificar códigos onde estes dois elementos são utilizados.

- *O elemento* `a` *sem o atributo* `href` *agora representa um placeholder no exato lugar que este link se encontra.*
- *O elemento* `address` *agora é tratado como uma seção no documento.*
- *O elemento* `hr` *agora tem o mesmo nível que um parágrafo, mas é utilizado para quebrar linhas e fazer separações.*
- *O elemento* `strong` *ganhou mais importância.*
- *O elemento* `head` *não aceita mais elementos* `child` *como seu filho.*

Elementos ou atributos descontinuados

Os elementos abaixo foram descontinuados por que seus efeitos são apenas visuais:

- *basefont*
- *big*
- *center*
- *font*
- *s*
- *strike*
- *tt*
- *u*

Os elementos abaixo foram descontinuados por que ferem os princípios de acessibilide e usabilidade:

- *frame*
- *frameset*
- *noframes*

Os elementos abaixo não foram incluídos na especificação porque não tiveram uso entre os desenvolvedores ou porque sua função foi substituída por outro elemento:

- `acronym` *não foi incluído porque criou um bocado de confusão entre os desenvolvedores que preferiram utilizar a tag* `abbr`.
- `applet` *ficou obsoleto em favor da tag* `object`.
- `isindex` *foi substituído pelo uso de form controls.*
- `dir` *ficou obsoleto em favor da tag* `ul`.

NOVOS TIPOS DE CAMPOS

Novos valores para o atributo type

O elemento input aceita os seguintes novos valores para o atributo type:

tel

Telefone. Não há máscara de formatação ou validação, propositalmente, visto não haver no mundo um padrão bem definido para números de telefones. É claro que você pode usar a nova API de validação de formulários (descrita no capítulo 8) para isso. Os agentes de usuário podem permitir a integração com sua agenda de contatos, o que é particularmente útil em telefones celulares.

search

Um campo de busca. A aparência e comportamento do campo pode mudar ligeiramente dependendo do agente de usuário, para parecer com os demais campos de busca do sistema.

email

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

url

Um endereço web, também com formatação e validação.

Datas e horas

O campo de formulário pode conter qualquer um desses valores no atributo type:

- *datetime*
- *date*
- *month*
- *week*
- *time*
- *datetime-local*

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

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

Opera 10

Enquanto escrevo, o Opera 10 é o único navegador Desktop que fez um bom trabalho implementando os novos recursos de formulário do HTML5. Se você instalá-lo, poderá testar quase tudo deste e dos próximos dois capítulos.

datetime-local

O tipo de campo datetime-local trata automaticamente as diferenças de fusos horários, submetendo ao servidor e recebendo dele valores GMT. Com isso você pode, com facilidade, construir um sistema que será usado em diferentes fusos horários e permitir que cada usuário lide com os valores em seu próprio fuso horário.

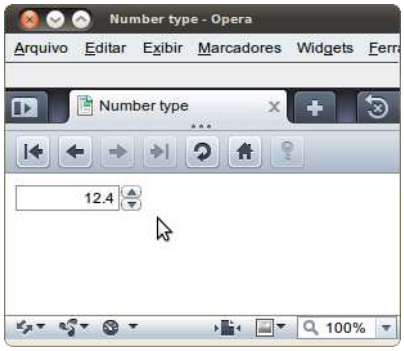
number

Veja um exemplo do tipo number com seus atributos opcionais:

Arquivo: exemplos/7/number.html

```
<!DOCTYPE html>
<html lang="en-US">
<head>
<meta charset="UTF-8" />
<title>Number type</title>
</head>

<body>

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

</body>

</html>

```

O Opera 10 nos dá uma excelente visualização do que um agente de usuário pode fazer nesse caso:



range

Vamos modificar, no exemplo acima, apenas o valor de type, mudando de "number" para "range":

Arquivo: exemplos/7/range.html

```
<!DOCTYPE html>
<html lang="en-US">
<head>
<meta charset="UTF-8" />
<title>Range type</title>
</head>

<body>

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

</body>

</html>

```

Novamente, Opera 10:

color

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

TIPOS DE DADOS E VALIDADORES

Formulários vitaminados

Conforme você deve ter percebido no último capítulo, o HTML5 avançou bastante nos recursos de formulários, facilitando muito a vida de quem precisa desenvolver aplicações web baseadas em formulários. Neste capítulo vamos avançar um pouco mais nesse assunto e, você vai ver, a coisa vai ficar ainda melhor.

autofocus

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

```
<input name="login" autofocus >
```

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

Placeholder text

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

Arquivo: exemplos/8/placeholderold.html

```
<!DOCTYPE html>
<html lang="en-US">
<head>
<meta charset="UTF-8" />
<title>Placeholder, the old style</title>
</head>

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

</html>
```

HTML5 nos permite fazer isso de maneira muito mais elegante:

Arquivo: exemplos/8/placeholder.html

```
<!DOCTYPE html>
<html lang="en-US">
<head>
<meta charset="UTF-8" />
<title>Placeholder, HTML5 way</title>
</head>

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

</html>
```

required

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

```
<input name="login" required>
```

maxlength

Você já conhecia o atributo `maxlength`, que limita a quantidade de caracteres em um campo de formulário. Uma grande lacuna dos formulário HTML foi corrigida. Em HTML5, o elemento textarea também pode ter maxlength!

Validação de formulários

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

pattern

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

Arquivo: exemplos/8/pattern.html

```
<!DOCTYPE html>
<html lang="pt-BR">
<head>
<meta charset="UTF-8" />
<title>O atributo pattern</title>
</head>

<body>

<form>
  <label for="CEP">CEP:
    <input name="CEP" id="CEP" required pattern="\d{5}-?\d{3}" />
  </label>
  <input type="submit" value="Enviar" />
</form>

</body>

</html>

```

novalidate e formnovalidate

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

Outra situação comum é querer que o formulário não seja validade dependendo da ação de submit. Nesse caso, você pode usar no botão de submit o atributo `formnovalidate`. Veja um exemplo:

Arquivo: exemplos/8/formnovalidate.html

```
<!DOCTYPE html>
<html lang="pt-BR">
<head>
<meta charset="UTF-8" />
<title>Salvando rascunho</title>
<style>

  label{display:block;}
```

```
  8

</style>
  9 </head>
 10
 11 <body>
 12
 13 <form>
 14   <label>nome: <input name="nome" required></label>
 15   <label>email: <input name="email" type="email" required></label>
 16   <label>mensagem: <textarea name="mensagem" required></textarea></label>
 17   <input type="submit" name="action" value="Salvar rascunho" formnovalidate>
 18   <input type="submit" name="action" value="Enviar">
 19 </form>
 20
 21 </body>
 22
 23 </html>
 24
```

Custom validators

É claro que as validações padrão, embora atendam a maioria dos casos, não são suficientes para todas as situações. Muitas vezes você vai querer escrever sua própria função de validação Javascript. Há alguns detalhes na especificação do HTML5 que vão ajudá-lo com isso:

1. ***O novo evento oninput*** *é disparado quando algo é modificado no valor de um campo de formulário. Diferente de onchange, que é disparado ao final da edição, oninput é disparado ao editar. É diferente também de onkeyup e onkeypress, porque vai capturar qualquer modificação no valor do campo, feita com mouse, teclado ou outra interface qualquer.*
2. ***O método setCustomValidity*** *pode ser invocado por você. Ele recebe uma string. Se a string for vazia, o campo será marcado como válido. Caso contrário, será marcado como inválido.*

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

Arquivo: exemplos/8/customvalidator.html

```
  1 <!DOCTYPE html>
  2 <html lang="pt-BR">
  3 <head>
  4 <meta charset="UTF-8" />
  5 <title>Custom validator</title>
  6 <!-- O arquivo cpf.js contém a função validaCPF, que
  7      recebe uma string e retorna true ou false.      -->
  8 <script src="cpf.js"></script>
  9 <script>

 10 function vCPF(i){

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

 12 }

 13


</script>
 14 </head>
 15
 16 <body>
 17 <form>
 18   <label>CPF: <input name="cpf" oninput="vCPF(this)" /></label>
 19   <input type="submit" value="Enviar" />
 20 </form>
 21 </body>
 22
 23 </html>
 24
```

DETALHES E CONTEÚDO EDITÁVEL.

Ainda mais formulários

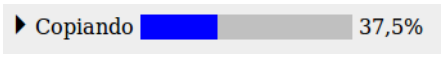
Vejamos mais duas coisas que você certamente já fez mais de uma vez e foram simplificadas pelo HTML5.

Detalhes e sumário

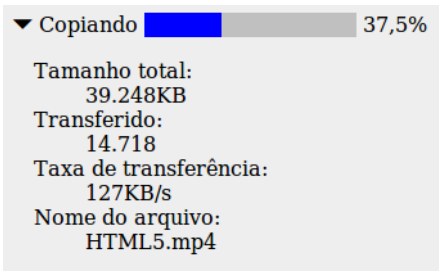
Veja um exemplo de uso dos novos elementos details e summary:

```
<details>
  <summary>Copiando <progress max="39248" value="14718"> 37,5%</summary>
  <dl>
    <dt>Tamanho total:</dt>
    <dd>39.248KB</dd>
    <dt>Transferido:</dt>
    <dd>14.718</dd>
    <dt>Taxa de transferência:</dt>
    <dd>127KB/s</dd>
    <dt>Nome do arquivo:</dt>
    <dd>HTML5.mp4</dd>
  </dl>
</details>
```

Veja como um agente de usuário poderia renderizar isso:



E ao clicar:



Conteúdo editável

Para tornar um elemento do HTML editável, basta incluir nele o atributo contenteditable, assim:

```
<div contenteditable="true">
  Edite-me...
</div>
```

Você pode ler e manipular os elementos editáveis normalmente usando os métodos do DOM.

Isso permite, com facilidade, construir uma área de edição de HTML.

DRAG-N-DROP E CORREÇÃO ORTOGRÁFICA

Drag and Drop

A API de Drag and Drop é relativamente simples. Basicamente, inserir o atributo `draggable="true"` num elemento o torna arrastável. E há uma série de eventos que você pode tratar. Os eventos do objeto sendo arrastado são:

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

drag
O objeto está sendo arrastado

dragend
A ação de arrastar terminou

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

dragenter
O objeto sendo arrastado entrou no objeto target

dragleave
O objeto sendo arrastado deixou o objeto target

dragover
O objeto sendo arrastado se move sobre o objeto target

drop
O objeto sendo arrastado foi solto sobre o objeto target

Detalhes importantes:

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

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

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

Arquivo: exemplos/10/drag.html

```
 1 <!DOCTYPE HTML>
 2 <html>
 3 <head>
 4 <meta content="text/html; charset=UTF-8" http-equiv="content-type"/>
 5 <title>HTML5 Drag and drop demonstration</title>
 6 <style type="text/css">

 7 #boxA, #boxB {

 8   float:left; width:100px; height:200px; padding:10px; margin:10px; font-size:70%;

 9 }

10 #boxA { background-color: blue; }

11 #boxB { background-color: green; }
```

```
12
13 #drag, #drag2 {
14   width:50px; padding:5px; margin:5px; border:3px black solid; line-height:50px;
15 }
16 #drag { background-color: red;}
17 #drag2 { background-color: orange;}
18
</style>
19 <script type="text/javascript">
20
21 // Quando o usuário inicia um drag, guardamos no dataset do evento
22 // o id do objeto sendo arrastado
23 function dragStart(ev) {
24     ev.dataTransfer.setData("ID", ev.target.getAttribute('id'));
25 }
26
27 // Quando o usuário arrasta sobre um dos painéis, retornamos
28 // false para que o evento não se propague para o navegador, o
29 // que faria com que o conteúdo fosse selecionado.
30 function dragOver(ev) { return false; }
31
32 // Quando soltamos o elemento sobre um painel, movemos o
33 // elemento, lendo seu id do dataset do evento
34 function dragDrop(ev) {
35     var idelt = ev.dataTransfer.getData("ID");
36     ev.target.appendChild(document.getElementById(idelt));
37 }
38
39
</script>
40 </head>
41 <body>
42     <!-- Painel 1 -->
43     <div id="boxA"
44         ondrop="return dragDrop(event)"
45         ondragover="return dragOver(event)">
46         <!-- Draggable 1 -->
47         <div id="drag" draggable="true"
48             ondragstart="return dragStart(event)">drag me</div>
49         <!-- Draggable 2 -->
50         <div id="drag2" draggable="true"
51             ondragstart="return dragStart(event)">drag me</div>
52     </div>
53
54     <!-- Painel 2 -->
55     <div id="boxB"
56         ondrop="return dragDrop(event)"
57         ondragover="return dragOver(event)">
58     </div>
59
60 </body>
61 </html>
```

Exemplo

Segue um exemplo de drag-and-drop, baseado no excelente exemplo de Laurent Jouanneau (http://ljouanneau.com/lab/html5/demodragdrop.html).

Revisão ortográfica e gramatical

Os agentes de usuário podem oferecer recursos de revisão ortográfica e gramatical, dependendo do que houver disponível em cada plataforma. Os desenvolvedores podem controlar o comportamento dessa ferramenta através do atributo `spellcheck`. Inserir `spellcheck="true"` num elemento faz com que a revisão esteja habilitada para ele. Você também pode desabilitar a revisão para determinado elemento, inserindo `spellcheck="false"`.

ELEMENTOS AUDIO E VIDEO, E CODECS

Áudio

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

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

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

Origens alternativas de áudio

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

```
<audio controls="true" autoplay="true">
  <source src="mus.oga" />
  <source src="mus.mp3" />
  <source src="mus.wma" />
</audio>
```

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

```
<audio controls="true" autoplay="true">
  <source src="mus.oga" />
  <source src="mus.mp3" />
  <source src="mus.wma" />
  <p>Faça o <a href="mus.mp3">download da música</a>.</p>
</audio>
```

Vídeo

O uso de vídeo é muito semelhante ao de áudio:

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

E com vários elementos source:

```
<video controls="true" autoplay="true" width="400" height="300">
  <source src="u.ogv" />
  <source src="u.mp4" />
  <source src="u.wmv" />
  <p>Faça o <a href="u.mp4">download do vídeo</a>.</p>
</video>
```

Codecs

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

Para indicar ao navegador o container e codecs de determinado arquivo, usa-se o atributo `type`, no formato:

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

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

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

Com MPEG-4 a coisa é um pouco mais complicada, por que é preciso indicar ao navegador também o profile do codec de vídeo utilizado. Veja um exemplo:

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

O que funciona na web

Mark Pilgrim está escrevendo um livro muito interessante (em inglês) chamado "Dive Into HTML 5". O capítulo sobre Vídeo é a referência de que você precisa para publicar vídeo na web com HTML5.

ELEMENTO DEVICE E STREAM API

O elemento device

Você pode inserir em seu HTML um elemento de acesso à webcam do usuário, assim:

```
<device type="media">
```

Isso vai exibir uma interface solicitando ao usuário acesso a sua webcam. Se ele tiver mais de uma, também será permitido que ele escolha que webcam usar. O atributo media também pode conter o valor "fs", que vai abrir uma caixa de seleção no sistema de arquivos, permitindo ao usuário escolher um arquivo para fazer stream.

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

Working Draft

O conteúdo desse capítulo está baseado numa especificação que ainda está em status de "Working Draft". Ou seja, as coisas ainda podem mudar bastante. Fique de olho no que vai acontecer com o elemento device e a Stream API, acessando (em inglês): http://dev.w3.org/html5/html-device/

Arquivo: exemplos/12/videochat.html

```
<!DOCTYPE html>
<html lang="en-US">
<head>
<meta charset="UTF-8" />
<title>Videochat, step 1</title>

<script>

function update(stream) {

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

}


</script>

</head>

<body>

<p>To start chatting, select a video camera: <device type=media onchange="update(this.data)"></p>
<video autoplay />

</body>

</html>

```

Streams

Você deve ter notado, no script acima, que a função de update recebe um parâmetro stream. Trata-se de um objeto da classe Stream, que possui uma propriedade url, que já usamos acima, e um método record. O método record inicia a gravação do stream e retorna um objeto StreamRecorder. Esse último possui um método stop, que retorna o arquivo que foi gravado.

Peer-to-peer

Cuidado! O W3C ainda está trabalhando nessa especificação, e tudo aqui pode mudar. Por isso,

não se preocupe em entender as minúcias. Saiba apenas que HTML5 prevê que os agentes de usuário tenham uma interface de comuicação P2P, que permite a troca de texto, imagem, vídeo e arquivos. Por enquanto, a especificação deste item esté sendo escrita junto da do elemento device, mas isso deve ganhar uma página própria em breve. Fique de olho.

MATHML E SVG

MathML

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

Arquivo: exemplos/13/mathml.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8" />
<title>MathML</title>
</head>
<body>

<math xmlns="http://www.w3.org/1998/Math/MathML">
  <mrow>
    <mi>x</mi>
    <mo>=</mo>
    <mfrac>
      <mrow>
        <mo form="prefix">&minus;</mo>
        <mi>b</mi>
        <mo>&PlusMinus;</mo>
        <msqrt>
          <msup>
            <mi>b</mi>
            <mn>2</mn>
          </msup>
          <mo>&minus;</mo>
          <mn>4</mn>
          <mo>&InvisibleTimes;</mo>
          <mi>a</mi>
          <mo>&InvisibleTimes;</mo>
          <mi>c</mi>
        </msqrt>
      </mrow>
      <mrow>
        <mn>2</mn>
        <mo>&InvisibleTimes;</mo>
        <mi>a</mi>
      </mrow>
    </mfrac>
  </mrow>
</math>

</body>
</html>
```

Veja como esse exemplo é renderizado no navegador:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

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

operador de multiplicação entre eles, porque por convenção se simplesmente escrevemos 4ac qualquer leitor saberá que isso é uma multiplicação.

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

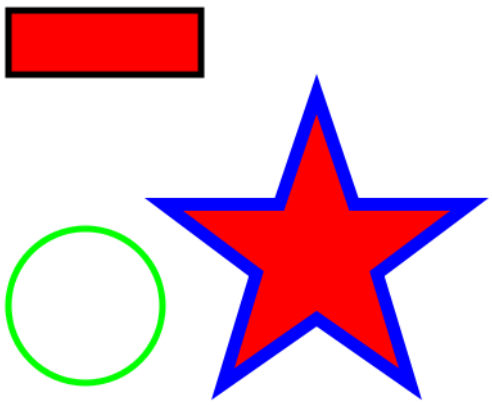
SVG

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

Arquivo: exemplos/13/svg.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8" />
<title>SVG</title>
</head>
<body>

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

  <!-- Um retângulo: -->
  <rect x="10" y="10" width="150" height="50" stroke="#000000" stroke-width="5" fill="#FF0000" />

  <!-- Um polígono: -->
  <polygon fill="red" stroke="blue" stroke-width="10"
    points="250,75  279,161 369,161 297,215
            323,301 250,250 177,301 203,215
            131,161 221,161" />

  <!-- Um círculo -->
  <circle cx="70" cy="240" r="60" stroke="#00FF00" stroke-width="5" fill="#FFFFFF" />

</svg>

</body>
</html>
```

E veja como isso é renderizado no navegador:



É possível fazer muito mais com SVG. A maioria dos editores de gráficos vetoriais hoje exporta e importa automaticamente SVG, permitindo a um designer construir um gráfico em seu editor vetorial predileto e exportá-lo diretamente. Em seguida, um programador pode construir javascript que manipula esse SVG, usando os métodos do DOM. Com isso você pode ter gráficos dinâmicos, com animação, escaláveis e com excelente qualidade visual, programáveis em Javascript, sem tecnologias proprietárias e plugins.

CANVAS API

O elemento canvas

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

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

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

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

Agora que temos um contexto, podemos desenhar nele. Vamos começar com um simples retângulo:

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

Simples, não? Que tal tentarmos algo um pouco mais complexo? Dê uma olhada no exemplo:

Existe um contexto 3D?

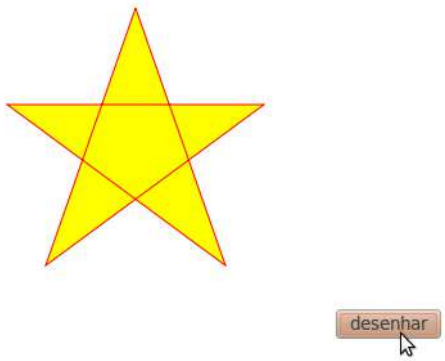
Ainda não. Existem várias implementações de contexto 3D, e cada fabricante de navegador tem criado a sua, mas ainda não há um padrão do W3C sobre esse assunto.

Arquivo: exemplos/14/canvas.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8" />
<title>Canvas API</title>
</head>
<body>

<canvas id="x" width="300" height="300"></canvas>
<button onclick="desenhar()">desenhar</button>

<script>

function desenhar(){

  // Obtemos o contexto

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


  //Iniciamos um novo desenho

  context.beginPath()


  //Movemos a caneta para o inicio do desenho

  context.moveTo(150,50)


  //Desenhamos as linhas

  context.lineTo(220,250)

  context.lineTo(50,125)
```

```
26   context.lineTo(250,125)
27   context.lineTo(80,250)
28   context.lineTo(150,50)
29
30   //O desenho não é de verdade enquanto você
31   //não mandar o contexto pintá-lo.
32
33   //Vamos pintar o interior de amarelo
34   context.fillStyle='#ff0'
35   context.fill()
36
37   //Vamos pintar as linhas de vermelho.
38   context.strokeStyle='#f00'
39   context.stroke()
40
41 }
42
</script>
43
44 </body>
45 </html>
```

E veja o que acontece quando se clica no botão:



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

Canvas e SVG

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

Assim, as vantagens do SVG são:

1. *O conteúdo é acessível a leitores de tela*
2. *O gráfico é escalável, não perde resolução ou serrilha ao redimensionar*
3. *O conteúdo é acessível via DOM*

E as vantagens do Canvas:

1. *A performance é muito superior ao SVG na maioria dos casos*
2. *É fácil desenhar via Javascript. Em SVG, é preciso fazer seu script escrever XML para você. Com Canvas você só manda desenhar, e pronto.*

SERVER-SENT EVENTS

EventSource

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

```
es=new EventSource('comm.php')
```

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

```
es.onmessage=function(e){
  alert("Chegaram dados: "+e.data)
}
```

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

O protocolo de comunicação

Em nosso exemplo acima, a página comm.php envia eventos para o agente de usuário. Você não precisa se preocupar em saber como isso funciona do lado do cliente, uma vez que o agente de usuário faz todo o trabalho. Mas é importante que saiba como isso deve funcionar do lado do servidor. A URL de comunicação deve devolver ao cliente um header `Content-type: text/event-stream`. Em seguida, envia as mensagens, que são blocos de texto separados um do outro por uma linha em branco:

```
data: mensagem 1

data: a mensagem 2 tem
data: mais de uma linha

data: mensagem 3
```

O prefixo data: indica que o que segue são os dados da mensagem. Você também pode usar o prefixo id:

```
data: mensagem 1
id: 1

data: a mensagem 2 tem
data: mais de uma linha
id: 2

data: mensagem 3
id: 3
```

Se você enviar prefixos id em suas mensagens e o agente de usuário perder a conexão, ao tentar reconectar ele vai enviar o valor do último id no header HTTP `Last-Event-ID`. Com isso você pode, por exemplo, enviar as mensagens do chat do ponto em que parou.

DOM E HTML5

DOM e HTML5

O Modelo de Objetos do Documento (DOM, na sigla em inglês) é a interface entre a linguagem Javascript e os objetos do HTML. DOM é o método padrão para construção de aplicações ricas com Javascript e é amplamente conhecido e utilizado. Neste capítulo, supondo que você já conhece DOM para HTML 4 ou XHTML, vamos nos focar na diferença entre as versões anteriores do DOM e a do HTML 5.

Por quê DOM?

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

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

Vamos às diferenças

getElementsByClassName

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

```
destaques = document.getElementsByClassName('destaque')
```

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

innerHTML

Outro sonho antigo que se torna realidade. A propriedade innerHTML é uma idéia tão boa que todos os navegadores atuais já a suportam há muito tempo e todo desenvolvedor web sabe usá-la. Apesar disso, ela nunca havia sido descrita como um padrão.

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

Veja um exemplo de innerHTML:

```
function adicionaItem(nome){
  document.getElementById('lista').innerHTML += '<li>'+nome+'</li>'
}
```

activeElement e hasFocus()

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

Veja um exemplo de script dependente de foco:

Arquivo: exemplos/16/focusNotify.html

```
<!DOCTYPE html>
<html lang="pt-BR">
<head>
<meta charset="UTF-8" />
  <title>Notifier</title>
  <script>

  function notify(text){
    document.getElementById('msg').innerHTML+='<p>'+text+'</p>'
    titleFlick()
  }

  function titleFlick(){
    if(document.hasFocus()){
      document.title='Notifier'
      return
    }
    document.title=document.title=='Notifier'?'* Notifier':'Notifier'
    setTimeout('titleFlick()',500)
  }


</script>
</head>

<body>
  <input type="button" id="notify" value="Notify in 5 seconds"
  onclick="notify('Will notify in 5 seconds...');setTimeout('notify(\'Event shoot!\')',5000)" />
  <div id="msg"></div>
</body>

</html>
```

getSelection()

Os objetos document e window possuem um método getSelection(), que retorna a seleção atual, um objeto da classe Selection. A seleção tem, entre outros, os seguintes métodos e propriedades:

anchorNode
O elemento que contém o início da seleção

focusNode
O elemento que contém o final da seleção

selectAllChildern(parentNode)
Seleciona todos os filhos de parentNode

deleteFromDocument()
Remove a seleção do documento

Usando getSelection() hoje

A maioria dos navegadores ainda não teve tempo de se atualizar em relação à especificação e, retorna uma string quando você chama `document.getSelection()` e um objeto Selection quando você

rangeCount
A quantidade de intervalos na seleção

getRangeAt(index)
Retorna o intervalo na posição index

addRange(range)
Adiciona um intervalo à seleção

removeRange(range)
Remove um intervalo da seleção

chama `window.getSelection()`. Como esperamos que num futuro próximo o comportamento de `document.getSelection()` mude, sugerimos que você prefira usar o método de window por enquanto.

Intervalos de seleção

Você deve ter notado acima que uma seleção é um conjunto de intervalos, da classe Range. Cada intervalo possui, entre outros, os seguintes métodos e propriedades:

deleteContent()
Remove o conteúdo do intervalo

setStart(parent,offset)
Seta o início do intervalo para o caractere na posição offset dentro do elemento DOM parent

setEnd(parent,offset)
Seta o final do intervalo para o caractere na posição offset dentro do elemento DOM parent

Tanto os objetos Selection quanto os objetos Range retornam o texto da seleção quando convertidos para strings.

document.head

O objeto document já possuía uma propriedade body, uma maneira conveniente de acessar o elemento body do HTML. Agora ele ganhou uma propriedade head, maneira também muito conveniente de acessar o elemento head.

Selector API

A Selector API não é novidade do HTML5, é anterior a ele. Mas como ainda é desconhecida de parte dos desenvolvedores, convém dizer que ela existe, e que continua funcionando no HTML5. Com a selector API você pode usar seletores CSS para encontrar elementos DOM.

A Selector API expõe duas funções em cada um dos elementos DOM: querySelector e querySelectorAll. Ambas recebem como argumento uma string com um seletor CSS. A consulta é sempre feita na subtree do elemento DOM a partir do qual a chamada foi disparada. A querySelector retorna o primeiro elemento que satisfaz o seletor, ou null caso não haja nenhum. A querySelectorAll retorna a lista de elementos que satisfazem o seletor.

Veja, neste exemplo, um script para tabelas zebradas com Selector API:

Arquivo: exemplos/16/zebra.html

```
 1 <!DOCTYPE html>
 2 <html lang="pt-BR">
 3 <head>
 4 <meta charset="UTF-8" />
 5 <title>Zebra</title>
 6 <style>

 7   .zebraon{background:silver}

 8

</style>
 9 <script>

10 window.onload=function(){

11   var zebrar=document.querySelectorAll('.zebra tbody tr')

12   for(var i=0;i<zebrar.length;i+=2)

13     zebrar[i].className='zebraon'

14 }
```

querySelector e jQuery

Se você é usuário de jQuery, já entendeu tudo. É exatamente a mesma idéia dos seletores jQuery.

Alguns preocupados usuários de jQuery têm nos perguntado se não é melhor, em termos de performance usar a Selector API. Mas é claro que é. Se você realmente souber programar, escrever todo o seu código sempre será melhor em performance que usar um framework. Mas o ganho, nesse caso, é desprezível. Talvez o conforte saber que, nos navegadores em que isto está disponível, a própria jQuery usa internamente a Selector API.

```
15

</script>
16 </head>
17
18 <body>
19 <table class="zebra">
20   <thead><tr>
21       <th>Vendedor</th> <th>Total</th>
22     </tr></thead>
23   <tbody><tr>
24       <td>Manoel</td>   <td>12.300,00</td>
25     </tr><tr>
26       <td>Joaquim</td>  <td>21.300,00</td>
27     </tr><tr>
28       <td>Maria</td>    <td>13.200,00</td>
29     </tr><tr>
30       <td>Marta</td>    <td>32.100,00</td>
31     </tr><tr>
32       <td>Antonio</td>  <td>23.100,00</td>
33     </tr><tr>
34       <td>Pedro</td>    <td>31.200,00</td>
35     </tr></tbody>
36 </table>
37 </body>
38 </html>
```

Características especiais de DomNodeList

As listas de elementos retornadas pelos métodos do DOM não são Arrays comuns, são objetos DomNodeList, o que significa que, entre outros métodos especiais, você pode usar `list[0]` ou `list(0)` para obter um elemento da lista. Também pode usar `list["name"]` ou `list("name")` para obter um objeto por seu nome. Duas adições interessantes do HTML5 ao usar este último método:

1. *O objeto é buscado pelos atributos name ou id.*
2. *Uma lista de campos de formulário com o mesmo valor no atributo name (uma lista de radio buttons, por exemplo) será retornada caso mais de um objeto seja encontrado. Essa lista contém um atributo especial, value, muito conveniente. Ele contém o valor do radio marcado e, ao ser setado, marca o radio correspondente.*

Datasets

Você pode atribuir dados arbitrários a um elemento HTML qualquer, prefixando seus atributos com "data-". Por exemplo:

```
<img src="computador1.jpg" alt="Novo GNexius Forethinker" id="c1"
  data-processor="35Ghz" data-memory="48GB"
  data-harddrive="16TB" data-screen='45"' >
```

Você pode acessar esses valores via Javascript, através do atributo dataset, assim:

```
var img=document.getElementById('c1')
proc=img.dataset.processor
```

As propriedades de dataset têm permissão de leitura e escrita.

NOVOS EVENTOS DOM

Uma palavra sobre eventos

O suporte ao tratamento de eventos disparados pelo usuário é parte essencial do DOM. HTML5 oferece a você um extenso conjunto de novos eventos. Vamos dar uma olhada nos mais interessantes:

Elementos multimídia:

oncanplay
O elemento audio ou video já tem dados suficientes no buffer para começar a tocar.

oncanplaythrough
O elemento audio ou video já tem dados suficientes no buffer para começar a tocar e, se a tranferência de dados continuar no ritmo em que está ocorrendo, estima-se que tocará até o final sem interrupções.

ondurationchange
O elemento audio ou video teve seu atributo `duration` *modificado. Isso acontece, por exemplo, ao alterar a origem da mídia.*

onemptied
O elemento audio ou video teve um erro de retorno vazio de dados da rede. O retorno vazio acontece quando, por exemplo, você tenta invocar o método play de um elemento que ainda não tem uma origem de mídia definida.

onended
O vídeo ou áudio chegou ao fim.

onloadeddata
Os dados começaram a ser carregados e a posição atual de playback já pode ser renderizada.

onloadedmetadata
Os metadados foram carregados. Já sabemos as dimensões, formato e duração do vídeo.

onloadstart
Os dados começaram a ser carregados.

onpause
O usuário clicou em pause.

onplay
O usuário clicou em play ou o playback começou por causa do atributo `autoplay`

onplaying
O vídeo ou áudio está tocando.

onprogress
O agente de usuário está buscando dados do vídeo ou áudio.

Eventos em campos de formulário:

oninput
O usuário entrou com dados no campo

oninvalid
O campo não passou pela validação

Eventos gerais:

oncontextmenu
O usuário disparou um menu de contexto sobre o objeto. Na maioria dos sistemas Desktop, isso significa clicar com o botão direito do mouse ou segurando uma tecla especial.

onmousewheel
A rodinha do mouse foi acionada.

onbeforeprint

Disparado antes da impressão da página. Você pode usá-lo para modificar, esconder ou exibir elementos, preparando a página para impressão.

onafterprint
Disparado após a impressão da página. Você pode usá-lo para reverter o status anterior à impressão.

onhashchange
A última porção da URL, após o hash (#), foi modificada.

onoffline
O agente de usuário ficou offline.

ononline
O agente de usuário está novamente conectado.

onredo
O usuário disparou a ação de "Refazer".

onundo
O usuário disparou a ação de "Desfazer".

Drag-and-drop:

Vimos a definição desses eventos no Capítulo 10:

- *ondrag*
- *ondragend*
- *ondragenter*
- *ondragleave*
- *ondragover*
- *ondragstart*
- *ondrop*

Atributos de evento

A especificação do HTML5 padronizou um formato de atribuição de eventos que já era amplamente utilizado. Você pode atribuir eventos através de atributos HTML com o nome do evento. Por exemplo:

```
<input onblur="return verifica(this)" />
```

É claro que você pode continuar usando o método do DOM `addEventListener`, com a vantagem de poder atribuir vários listeners ao mesmo evento.

MENUS E TOOLBARS

O elemento menu

O elemento `menu` é usado para definir menus e barras de ferramenta. Dentro do menu, você pode inserir submenus ou comandos. Para inserir submenus, basta inserir outros elementos `menu`. Para definir comandos, você pode inserir:

1. *Um link, um elemento `a` com atributo `href`;*
2. *Um botão, um elemento `button`;*
3. *Um botão, um elemento `input` com o atributo `type` contendo button, submit, reset ou image;*
4. *Um radiobutton, um elemento `input` com o atributo `type` contendo radio;*
5. *Um checkbox, um elemento `input` com o atributo `type` contendo checkbox;*
6. *Um elemento `select`, contendo um ou mais options, define um grupo de comandos*
7. *Um elemento qualquer com o atributo `accesskey`*
8. *Um elemento `command`*

Tipos de comando

Há três tipos de comando:

command
Uma ação comum;

checkbox
Uma ação que pode estar no status de ligada ou desligada, e alterna entre esses dois stauts quando clicada;

radio
Uma ação que pode estar no status de ligada ou desligada, e quando clicada vai para o status de ligada, deligando todas as ações com o mesmo valor no atributo `radiogroup`;

Da lista de elementos possíveis para definir comandos, os três primeiros, link, button e input button, definem comandos do tipo command. O quarto elemento, radiobutton, define um comando do tipo radio. O quinto, checkbox, define um comando do tipo checkbox.

O sexto elemento, o select, vai definir um grupo de comandos. Se o select tiver o atributo multiple, definirá uma lista de comandos do tipo checkbox. Caso contrário, os comandos serão do tipo radio, tendo o mesmo radiogroup.

No sétimo caso, um elemento qualquer com tecla de acesso, o tipo de comando vai depender do tipo de elemento que recebeu accesskey.

O elemento command

Por fim, temos o oitavo método, o elemento `command`. Neste caso o tipo de comando dependerá do valor do atributo `type`. Veja um exemplo de como usá-lo:

```
<command type="command" label="Salvar" onclick="salvar()" >
```

Prefira não usar command, por enquanto

Por que a especificação permite que se use o novo elemento command para definir comandos, e ao mesmo tempo permite que se use os velhos elementos como input, button e select para isso? Para possibilitar ao desenvolvedor oferecer alguma compatibilidade com navegadores antigos.

Veja este exemplo:

Arquivo: exemplos/18/command.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8" />
<title>Menus</title>
</head>

<body>

<menu type="toolbar">
  <li>
    <menu label="File">
      <button type="button" onclick="fnew()">New...</button>
      <button type="button" onclick="fopen()">Open...</button>
      <button type="button" onclick="fsave()">Save</button>
      <button type="button" onclick="fsaveas()">Save as...</button>
    </menu>
  </li>
  <li>
    <menu label="Edit">
      <button type="button" onclick="ecopy()">Copy</button>
      <button type="button" onclick="ecut()">Cut</button>
      <button type="button" onclick="epaste()">Paste</button>
    </menu>
  </li>
  <li>
    <menu label="Help">
      <li><a href="help.html">Help</a></li>
      <li><a href="about.html">About</a></li>
    </menu>
  </li>
</menu>

</body>

</html>
```

O agente de usuário deveria renderizar algo como:

Um agente de usuário que não conhece o novo elemento `menu` vai entender esse código como listas aninhadas com botões e links. E vai renderizar isso assim:

Não está bonito, mas é perfeitamente acessível. E o visual pode ser bem trabalhado com CSS. A mesma coisa poderia ser escrita com o elemento `command`:

Arquivo: exemplos/18/command2.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8" />
<title>Menus</title>
</head>

<body>

<menu type="toolbar">
  <menu label="File">
    <command onclick="fnew()" label="New..." />
    <command onclick="fopen()" label="Open..." />
    <command onclick="fsave()" label="Save" />
    <command onclick="fsaveas()" label="Save as..." />
  </menu>
  <menu label="Edit">
    <command onclick="ecopy()" label="Copy" />
    <command onclick="ecut()" label="Cut" />
    <command onclick="epaste()" label="Paste" />
```

```
21   </menu>
22   <menu label="Help">
23     <command onclick="location='help.html'" label="Help" />
24     <command onclick="location='about.html'" label="About" />
25   </menu>
26 </menu>
27 
28 </body>
29 
30 </html>
```

Mas um agente de usuário que não conhece os elementos `menu` e `command` não vai mostrar absolutamente nada.

TIPOS DE LINKS

Links

A possibilidade de linkar documentos é o que torna a Web o que ela é. Existem duas maneiras principais de linkar documentos, os elementos `a` e `link`. O elemento `a` cria um link no conteúdo da página. Você conhece sua sintaxe:

```
<a href="http://visie.com.br">Visie</a>
```

O elemento `link`, por sua vez, cria um metadado, um link que não é mostrado no conteúdo, mas o agente de usuário usa de outras maneiras. O uso mais comum é vincular um documento a uma folha de estilos:

```
<link rel="stylesheet" href="estilo.css" />
```

Note o atributo `rel="stylesheet"`. O atributo `rel` pode estar presente nos elementos `a` e `link`, e ter uma série de valores:

Metadados de navegação

archives
os arquivos do site

author
a página do autor do documento atual

bookmark
o permalink da seção a que este documento pertence

first
o primeiro documento da série a qual este pertence

help
ajuda para esta página

index
o índice ou sumário que inclui o link para esta página

last
o último documento da série a qual este pertence

license
a licensa que cobre este documento

next
o próximo documento da série a qual este pertence

prefetch
o agente de usuário deve fazer cache desse link em segundo plano tão logo o documento atual tenha sido carregado. O autor do documento indica que este link é o provável próximo destino do usuário.

prev
o documento anterior da série a qual este pertence

search
a busca deste site

up
O documento um nível acima deste

O Opera nos dá um interessante exemplo de como um agente de usuário pode exibir estes links:

Metadados da página

alternate
um formato alternativo para o conteúdo atual. Precisa estar acompanhado do atributo `type`*, contendo o tipo MIME do formato. Por exemplo, para indicar o RSS da página atual usamos:*

```
<link rel="alternate" type="application/rss+xml" href="rss.xml" />
```

icon
o ícone que representa esta página

pingback
a URL de pingback desta página. Através desse endereço um sistema de blogging ou gerenciador de conteúdo pode avisar automaticamente quando um link para esta página for inserido.

stylesheet
a folha de estilo linkada deve ser vinculada a este documento para exibição

Comportamento dos links na página

external
indica um link externo ao domínio do documento atual

nofollow
indica que o autor do documento atual não endossa o conteúdo desse link. Os robôs de indexação para motores de busca podem, por exemplo, não seguir este link ou levar em conta o nofollow em seu algoritmo de ranking.

noreferrer
o agente de usuário não deve enviar o header HTTP Referer se o usuário acessar esse link

sidebar
o link deve ser aberto numa sidebar do navegador, se este recurso estiver disponível

MICRODATA

Semântica adicional

Dê um olhada no seguinte código:

Arquivo: exemplos/20/microdata1.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8" />
<title>Microdata 1</title>
</head>
<body>

<h1>Resultados do trimestre</h1>
<ol>
  <li>
    <dl>
      <dt>nome</dt>  <dd>Joaquim</dd>
      <dt>total</dt> <dd>10.764</dd>
    </dl>
  </li>
  <li>
    <dl>
      <dt>nome</dt>  <dd>Manoel</dd>
      <dt>total</dt> <dd>12.449</dd>
    </dl>
  </li>
  <li>
    <dl>
      <dt>nome</dt>  <dd>Antonio</dd>
      <dt>total</dt> <dd>9.202</dd>
    </dl>
  </li>
  <li>
    <dl>
      <dt>nome</dt>  <dd>Pedro</dd>
      <dt>total</dt> <dd>17.337</dd>
    </dl>
  </li>
</ol>

</body>
</html>
```

A Microdata API nos permite tornar esta estrutura semântica um pouco mais específica, definindo o que é o conteúdo de cada elemento. Veja este outro exemplo:

Arquivo: exemplos/20/microdata2.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8" />
<title>Microdata 2</title>
</head>
<body>

<h1>Resultados do trimestre</h1>
<ol>
  <li>
    <dl itemscope>
      <dt>nome</dt>  <dd itemprop="nome">Joaquim</dd>
      <dt>total</dt> <dd itemprop="total">10.764</dd>
    </dl>
  </li>
  <li>
    <dl itemscope>
      <dt>nome</dt>  <dd itemprop="nome">Manoel</dd>
```

```
      <dt>total</dt> <dd itemprop="total">12.449</dd>
    </dl>
  </li>
  <li>
    <dl itemscope>
      <dt>nome</dt>  <dd itemprop="nome">Antonio</dd>
      <dt>total</dt> <dd itemprop="total">9.202</dd>
    </dl>
  </li>
  <li>
    <dl itemscope>
      <dt>nome</dt>  <dd itemprop="nome">Pedro</dd>
      <dt>total</dt> <dd itemprop="total">17.337</dd>
    </dl>
  </li>
</ol>

</body>
</html>
```

Adicionamos atributos especias, itemscope e itemprop. Cada elemento itemscope define um item de dados. Cada itemprop define o nome de uma propriedade. O valor da propriedade é o conteúdo da tag HTML. A Microdata API nos fornece acesso especial a esses dados. Veja como acessar esses dados:

```
resultados=document.getItems()
for(var i=0;i<resultados.length;i++){
  alert(resultados[i].properties.nome[0].content+": R$ "+
        resultados[i].properties.total[0].content)
}
```

Diferentes tipos de dados

No exemplo acima, temos uma listagem de pessoas. Agora imagine que você precise ter, no mesmo documento, uma listagem de pessoas e carros. Poderia escrever assim:

Arquivo: exemplos/20/microdata3.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8" />
<title>Microdata 3</title>
</head>
<body>

<h1>Resultados do trimestre</h1>
<ol>
  <li>
    <dl itemscope>
      <dt>nome</dt>  <dd itemprop="nome">Joaquim</dd>
      <dt>total</dt> <dd itemprop="total">10.764</dd>
    </dl>
  </li>
  <li>
    <dl itemscope>
      <dt>nome</dt>  <dd itemprop="nome">Manoel</dd>
      <dt>total</dt> <dd itemprop="total">12.449</dd>
    </dl>
  </li>
  <li>
    <dl itemscope>
      <dt>nome</dt>  <dd itemprop="nome">Antonio</dd>
      <dt>total</dt> <dd itemprop="total">9.202</dd>
    </dl>
  </li>
  <li>
    <dl itemscope>
      <dt>nome</dt>  <dd itemprop="nome">Pedro</dd>
      <dt>total</dt> <dd itemprop="total">17.337</dd>
    </dl>
  </li>
</ol>

<h2>Carros mais vendidos</h2>
<ol>
  <li>
    <dl itemscope>
      <dt>nome</dt>  <dd itemprop="nome">Fusca</dd>
      <dt>total</dt> <dd itemprop="total">382</dd>
    </dl>
  </li>
  <li>
    <dl itemscope>
      <dt>nome</dt>  <dd itemprop="nome">Brasília</dd>
      <dt>total</dt> <dd itemprop="total">298</dd>
```

```
    </dl>
  </li>
  <li>
    <dl itemscope>
      <dt>nome</dt>  <dd itemprop="nome">Corcel</dd>
      <dt>total</dt> <dd itemprop="total">102</dd>
    </dl>
  </li>
</ol>

</body>
</html>
```

Note que pessoas e carros tem propriedades em comum, nome e total. Quando você executar `document.getItems()` vai obter uma lista de todos os elementos com `itemscope`. Como obter uma lista apenas de pessoas ou de carros? Você pode adicionar a cada item um atributo `itemtype`, que diz de que tipo de entidade são aqueles dados:

Arquivo: exemplos/20/microdata4.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8" />
<title>Microdata 4</title>
</head>
<body>

<h1>Resultados do trimestre</h1>
<ol>
  <li>
    <dl itemscope itemtype="pessoa">
      <dt>nome</dt>  <dd itemprop="nome">Joaquim</dd>
      <dt>total</dt> <dd itemprop="total">10.764</dd>
    </dl>
  </li>
  <li>
    <dl itemscope itemtype="pessoa">
      <dt>nome</dt>  <dd itemprop="nome">Manoel</dd>
      <dt>total</dt> <dd itemprop="total">12.449</dd>
    </dl>
  </li>
  <li>
    <dl itemscope itemtype="pessoa">
      <dt>nome</dt>  <dd itemprop="nome">Antonio</dd>
      <dt>total</dt> <dd itemprop="total">9.202</dd>
    </dl>
  </li>
  <li>
    <dl itemscope itemtype="pessoa">
      <dt>nome</dt>  <dd itemprop="nome">Pedro</dd>
      <dt>total</dt> <dd itemprop="total">17.337</dd>
    </dl>
  </li>
</ol>

<h2>Carros mais vendidos</h2>
<ol>
  <li>
    <dl itemscope itemtype="carro">
      <dt>nome</dt>  <dd itemprop="nome">Fusca</dd>
      <dt>total</dt> <dd itemprop="total">382</dd>
    </dl>
  </li>
  <li>
    <dl itemscope itemtype="carro">
      <dt>nome</dt>  <dd itemprop="nome">Brasília</dd>
      <dt>total</dt> <dd itemprop="total">298</dd>
    </dl>
  </li>
  <li>
    <dl itemscope itemtype="carro">
      <dt>nome</dt>  <dd itemprop="nome">Corcel</dd>
      <dt>total</dt> <dd itemprop="total">102</dd>
    </dl>
  </li>
</ol>

</body>
</html>
```

Agora você pode executar: `document.getItems('carro')` para obter só os carros, por exemplo.

Falando um idioma comum

Você deve ter notado que pode definir seus próprios padrões de metadados com microdata. Recomendo que, antes de criar seu próprio formato, verifique se o mesmo problema não já foi resolvido por alguém. O site www.data-vocabulary.org contém alguns desses formatos padronizados. Por exemplo, para descrever os dados de sua empresa ou organização, não invente seu próprio formato, use o formato definido em http://www.data-vocabulary.org/Organization. O valor de `itemtype` deve ser a própria URL que documenta o formato. Veja como fica:

Arquivo: exemplos/20/visie.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8" />
<title>Visie Padrões Web</title>
</head>
<body>

<address itemscope itemtype="http://data-vocabulary.org/Organization">
  <h1 itemprop="name">Visie Padrões Web</h1>
  <div itemprop="address" itemscope itemtype="http://data-vocabulary.org/Address">
    <p itemprop="street-address">Alameda dos Ubiatans, 257 - Planalto Paulista</p>
    <p>
       <span itemprop="locality">São Paulo</span> -
       <span itemprop="region">SP</span> -
       <span itemprop="country-name">Brasil</span>
    </p>
    <p itemprop="postal-code">04070-030</p>
  </div>
  <div itemprop="tel">+55.11.3477-3347</div>
</address>

</body>
</html>
```

Claro que a vantagem de usar o formato padronizado ao invés de inventar o seu não é apenas não ter o trabalho de pensar os nomes das propriedades. Os sistemas de busca, e outros sistemas que acessem seu site, podem entender e tratar esses dados. O Google já faz isso, veja neste endereço:

http://www.google.com/support/webmasters/bin/answer.py?hl=en&answer=146861

HISTÓRICO DE SESSÃO E API STORAGE

Histórico de Sessão e API Storage

Um dos grandes desafios de usabilidade ao se construir aplicações web com a tecnologia atual é apresentar um modelo de navegação consistente para o usuário. Duas grandes lacunas nos impediam de fazê-lo:

1. *Não havia uma forma simples de fazer com que as ações locais do usuário numa página fossem refletidas na próxima. Por exemplo, se o usuário abre e fecha itens em um menu em árvore e em seguida navega para a próxima página, era muito difícil fazer com que o menu aparece no mesmo estado na segunda página.*
2. *Não havia uma forma simples de fazer com que as ações do usuário numa página Ajax respondessem corretamente aos botões de controle de histórico do navegador (voltar e avançar).*

HTML5 traz formas simples de solucionar os dois problemas.

Histórico de Sessão

Você provavelmente conhece o objeto history do navegador e seus métodos `go`, `back` e `forward`. Ele nos permite, via javascript, um controle básico do histórico de navegação. O mesmo controle que o usuário, voltar e avançar.

O objeto history foi vitaminado no HTML5 com dois novos métodos:

1. ***pushState(data,title[,url])**: acrescenta uma entrada na lista de histórico.*
2. ***replaceState(data,title[,url])**: modifica a entrada atual na lista de histórico.*

Com isso, você pode acrescentar itens à lista de histórico, associando dados ou mesmo uma URL a eles. Por exemplo, digamos que você tenha três elementos de conteúdo em sua página e um script que exiba um por vez de acordo com os cliques do usuário no menu:

```
function showContent(n){

  // Escondemos todos os elementos de conteúdo
  for(var i=1;i<4;i++)
    document.getElementById('cont'+i).style.display='none'

  // Exibimos o elemento escolhido
  document.getElementById('cont'+n).style.display='block'

}
```

Segurança

Claro, se seu script tentar associar uma URL fora do domínio do script à lista de histórico, isso vai resultar numa exceção de segurança.

Vamos fazer com que nosso script acrescente uma linha de histórico ao selecionar um elemento:

```
function showPage(n){

  // Escondemos todos os elementos de conteúdo
  for(var i=1;i<4;i++)
    document.getElementById('cont'+i).style.display='none'

  // Exibimos o elemento escolhido
  document.getElementById('cont'+n).style.display='block'

}

function showContent(n){
  // Mostramos o conteúdo escolhido
```

```
    showPage(n)
    // Salvamos a página atual no histórico
    history.pushState({page:n},'Conteúdo '+n)
  }
```

Fazendo isso, cada vez que o usuário escolher um item no menu, o elemento será exibido e uma linha será acrescentada no histórico. O usuário poderá acessar normalmente esses itens de histórico usando o botão de voltar do navegador. Cada vez que ele usar o histórico, será disparado um evento popstate. Assim, para que nosso script esteja completo, basta tratar esse evento:

```
function showPage(n){

  // Escondemos todos os elementos de conteúdo
  for(var i=1;i<4;i++)
    document.getElementById('cont'+i).style.display='none'

  // Exibimos o elemento escolhido
  document.getElementById('cont'+n).style.display='block'

}

function showContent(n){
  // Mostramos o conteúdo escolhido
  showPage(n)
  // Salvamos a página atual no histórico
  history.pushState({page:n},'Conteúdo '+n)
}

// Quando o usuário navegar no histórico, mostramos a página relacionada:
window.onpopstate=function(e){
  if(e.state)
    showPage(e.page)
}
```

localStorage e sessionStorage

Até o HTML4, quando precisávamos armazenar dados no agente de usuário que persistissem entre as páginas, usávamos Cookies. Cookies nos permitiam armazenar o status de um menu javascript que precisava ser mantido entre as páginas, lembrar o nome do usuário, o histórico de operações realizadas por ele ou a última vez que ele visitou nosso site.

Com o aumento da complexidade das aplicações baseadas em web, duas grandes limitações dos Cookies nos incomodam:

1. ***Interface complexa:*** *o código para armazenar Cookies envolve complexos cálculos com datas e controle do nome de domínio.*
2. ***Limite de armazenamento:*** *alguns agentes de usuário permitiam o armazenamento de no máximo 20 Cookies, com apenas 4KB cada.*

HTML5 traz uma nova maneira de armazenar dados no client, a API Storage. Um objeto Storage possui os métodos:

1. ***getItem(key)****: obtém um valor armazenado no Storage*
2. ***setItem(key,value)*** *guarda um valor no Storage*
3. ***removeItem(key)*** *exclui um valor do Storage*
4. ***clear()*** *limpa o Storage*

Estão disponíveis dois objetos no escopo global (window): localStorage e sessionStorage. O objeto localStorage armazena os dados no client sem expiração definida. Ou seja, se o usuário fechar o navegador e voltar ao site semanas depois, os dados estarão lá. O sessionStorage armazena os dados durante a sessão atual de navegação.

O código para armazenar um valor na Storage se parece com isso:

```
localStorage.setItem('userChoice',33)
```

E quando você precisar desse valor, em outra página:

Serializar

Uma outra complicação dos Cookies resolvida pela API Storage é o fato de Cookies só armazenarem strings, nos obrigando a serializar arrays e objetos javascript. A especificação da API Storage rege que qualquer valor javascript pode ser armazenado e recuperado. Infelizmente, em alguns dos navegadores em que testamos, os valores são convertidos para strings assim como nos Cookies. Torçamos

```
localStorage.getItem('userChoice')
```

para que os agentes de usuário implementem corretamente esse recurso.

Essa interface já é muito mais simples que a de Cookies. Mas pode ficar melhor. Você pode usar o Storage como um array. Por exemplo:

```
if(!sessionStorage['theme']){
  sessionStorage['theme']='oldfurniture';
}
```

Não há como isso ser mais simples! Além disso, o espaço de armazenamento sugerido pela documentação é de 5MB para cada domínio, resolvendo, acredito que por mais uma década, o problema de espaço de armazenamento local.

APLICAÇÕES OFFLINE

Caching

HTML5 provê uma maneira de se indicar ao navegador que elementos são necessários e devem ser postos em cache para que uma aplicação funcione offline. O exemplo da documentação oficial é bastante esclarecedor. Dê uma olhada na seguinte página:

Arquivo: exemplos/22/clock.html

```
 1 <!DOCTYPE HTML>
 2 <html>
 3   <head>
 4     <title>Clock</title>
 5     <script src="clock.js"></script>
 6     <link rel="stylesheet" href="clock.css">
 7   </head>
 8   <body>
 9     <p>The time is: <output id="clock"></output></p>
10   </body>
11 </html>
```

Trata-se de um widget de relógio. Para funcionar, este HTML depende dos arquivos "clock.js" e "clock.css". Para permitir que o usuário acesse esta página offile, precisamos escrever um arquivo de manifesto, indicando que URLs devem ser postas em cache. Vamos preparar uma nova versão do widget, contendo o manifesto, que é um arquivo com a extensão .manifest e que deve ser servido com o tipo MIME `text/cache-manifest`. Em nosso caso, o arquivo vai se chamar clock.manifest e terá o seguinte conteúdo:

```
CACHE MANIFEST
clock1.html
clock.css
clock.js
```

Agora veja o HTML com o arquivo de manifesto linkado:

Arquivo: exemplos/22/clock1.html

```
 1 <!DOCTYPE HTML>
 2 <html manifest="clock.manifest">
 3   <head>
 4     <title>Clock</title>
 5     <script src="clock.js"></script>
 6     <link rel="stylesheet" href="clock.css">
 7   </head>
 8   <body>
 9     <p>The time is: <output id="clock"></output></p>
10   </body>
11 </html>
```

Note que é recomendado que você insira o próprio HTML principal na lista de URLs do arquivo de manifesto, embora não seja necessário. Ao encontrar uma página com um arquivo de manifesto vinculado, o navegador fará cache das URLs listadas no manifesto e da própria página.

Note também que não é necessário que todas as URLs para cache estejam importadas no documnto atual. O arquivo de manifesto pode contar todas as páginas de sua aplicação que forem necessárias para permitir o funcionamento offline, inclusive a navegação entre páginas.

O objeto ApplicationCache

O objeto ApplicationCache controla o status e operações de caching da página. Ele pode ser acessado via javascript, assim:

```
window.applicationCache
```

Seu método mais interessante é o update(), que faz com que o agente de usuário recarregue o cache da aplicação. Além disso, ele possui a propriedade status, cujo valor numérico pode ser um dos seguintes:

0 - UNCACHED
Não há um arquivo de manifesto nesta página ou apontando para ela

1 - IDLE
O objeto ApplicationCache está ocioso. O cache está atualizado.

2 CHECKING
O arquivo de manifesto está sendo baixado e conferido.

3 - DOWNLOADING
As URLs vinculadas no manifesto estão sendo baixadas.

4 - UPDATEREADY
O cache é antigo, mas ainda não foi marcado como obsoleto.

5 - OBSOLETE
O cache foi marcado como obsoleto e precisa ser atualizado assim que possível.

O objeto ApplicationCache também possui os seguintes eventos, relacionados a sua mudança de status:

- *onchecking*
- *onerror*
- *onnoupdate*
- *ondownloading*
- *onprogress*
- *onupdateready*
- *oncached*
- *onobsolete*

Como você pode ver, além de onerror, temos um evento para cada um dos status da lista acima.

Controle de status da aplicação

No exemplo do relógio acima não há formulários ou submissões Ajax. O agente de usuários não troca dados com o servidor. Assim é muito fácil fazer sua aplicação rodar offline, mas essa não é a realidade da maioria das aplicações. Vimos no capítulo anterior como fazer armazenamento local de dados. Com isso, você pode armazenar os dados que o navegador deveria enviar para o servidor enquanto a aplicação estiver offline e, tão logo ela esteja online, enviar tudo.

Para saber se a aplicação está online, basta acessar a propriedade onLine do objeto window.navigator:

```
function salva(dados){
  if(window.navigator.onLine){
    enviaAjax(dados)
  }else{
    salvaLocal(dados)
  }
}
```

E para disparar o envio quando a aplicação estiver online e avisar o usuário quando ela estiver offline, usamos os eventos ononline e onoffline do objeto window:

```
window.ononline=function(){
```

```
  enviaAjax(obtemLocal())
  document.getElementById('warning').innerHTML=''
}

window.onoffline=function(){
  document.getElementById('warning').innerHTML='Aplicação offline.'
}
```

SCROLL IN TO VIEW E HIDDEN

Scrolling into view

Um truque simples, mas muito útil. Você pode fazer:

```
document.getElementById('aviso').scrollIntoView()
```

Isso vai rolar a página até que o elemento com o id "aviso" esteja visível no topo do viewport.

Você pode passar um parâmetro opcional top:

```
document.getElementById('aviso').scrollIntoView(false)
```

O valor default é true. Se você passar false, a rolagem vai deixar o objeto visível na base do viewport.

hidden

Ocultar e exibir elementos é uma das tarefas mais comuns em Javascript. Em HTML5 existe um atributo específico para isso, o atributo hidden. Ao inserí-lo em um elemento assim:

```
<div hidden>Xi, se esconde!</div>
```

Ou assim:

```
<div hidden="true">Xi, se esconde!</div>
```

O elemento estará oculto.

hidden e Javascript

Acessar o atributo hidden em Javascript é muito conveniente:

```
function switchElement(elm){
  if(elm.hidden)
    elm.hidden=false
  else
    elm.hidden=true
}
```

Claro, você pode fazer:

```
function switchElement(elm){
  elm.hidden=!elm.hidden
}
```

Sugiro que você sempre use o atributo hidden. Descobrir se o elemento está oculto lendo as propriedades display e visibility do CSS, além de dar mais trabalho, pode gerar confusão.

GEOLOCATION API

Métodos de Geolocalização

Há três populares maneiras de um agente de usuário descobrir sua posição no globo:

> ***Geolocalização IP***
> *É o método usado pela maioria dos navegadores web em computadores. Através de consultas whois e serviços de localização de IP, vai determinar a cidade ou região em que você está.*
>
> ***Triangulação GPRS***
> *Dispositivos conectados a uma rede de celulares e sem um GPS, ou com o GPS desligado, podem determinar sua posição pela triângulação das antenas GPRS próximas. É bem mais preciso que o método baseado em IP, vai mostrar em que parte do bairro você está.*
>
> ***GPS***
> *É o método mais preciso. Em condições ideais, a margem de erro é de apenas 5 metros.*

Embora essas sejam as três maneiras mais populares de se resolver o problema, podem não ser as únicas. Alguns agentes de usuário podem usar uma combinação desses métodos, ou mesmo um novo método que venha a ser inventado. Por isso, a Geolocation API é agnóstica em relação ao método usado. Há apenas uma maneira de ligar e desligar o "modo de alta precisão", o que vai ter significado diferente em cada agente de usuário.

Para obter a posição do usuário, basta executar o script:

```
navigator.geolocation.getCurrentPosition(showpos)
```

Onde `showpos` é uma função callback, que vai receber um objeto de posicionamento. Veja um exemplo:

```
function showpos(position){
  lat=position.coords.latitude
  lon=position.coords.longitude
  alert('Your position: '+lat+','+lon)
}
```

Claro, você pode fazer o que quiser, abrir um mapa, submeter a posição via Ajax, enviar os dados para um webservice, etc.

O método getCurrentPosition recebe dois outros parâmetros. O primeiro é uma função para tratamento de erro. O segundo, um objeto de configuração.

Tratando erros

Quando o script tenta acessar o posicionamento, o navegador exibe uma barra como esta:

O usuário pode então escolher se deseja ou não compartilhar sua posição com o site. Além de o usuário poder dizer não, muita coisa pode dar errado na hora de obter a geolocalização. Para tratar isso, você pode passar o segundo parâmetro a getCurrentPosition:

```
navigator.geolocation.getCurrentPosition(showpos,erropos)
```

Caso algo dê errado, a função erropos vai receber um objeto PositionError, que tem o atributo

code, que pode ter um dos seguintes valores:

1 - Permissão negada
O usuário clicou em "não compartilhar".

2 - Posição indisponível
O agente de usuário está desconectado, os satélites de GPS não puderam ser alcançados ou algum erro semelhante.

3 - Timeout
Tempo esgotado ao obter uma posição. Você pode definir o tempo máximo ao chamar getCurrentPosition.

0 - Erro desconhecido
Alguma outra coisa impediu o agente de usuário de obter uma posição.

Não trate a resposta do usuário como um erro

Em sua função de tratamento de erro, se obtiver o código de erro 1, por favor, não incomode o usuário com mensagens de erro. Ele escolheu não compartilhar sua posição com o site. Talvez a melhor atitude seja não fazer nada nesse momento.

O objeto de configuração

O terceiro parâmetro de getCurrentPosition é um objeto de configuração, que pode ter as seguintes propriedades:

enableHighAccuracy
Se true, liga o modo de alta precisão. Num celular isso pode instruir o navegador, por exemplo, a usar o GPS ao invés da triangulação GPRS

timeout
O tempo em milissegundos que o agente do usuário vai esperar pela posição antes de disparar um erro tipo 3.

maximumAge
O tempo, em milissegundos, que o navegador pode cachear a posição.

watchPosition

Se o que você deseja é rastrear a posição do usuário continuamente, pode usar, ao invés de getCurrentPosition, o método watchPosition. Ele tem a mesma assinatura de getCurrentPosition:

```
w=navigator.geolocation.watchPosition(showpos,erropos)
```

A diferença é que a função showpos será chamada toda vez que a posição do usuário mudar. O valor de retorno é um número, que pode ser usado posteriormente para cancelar o watcher:

```
navigator.geolocation.clearWatch(w)
```

UNDO

O objeto UndoManager

O agente de usuário deve armazenar um histórico de alterações para cada documento carregado.

Esse histórico é controlado pelo objeto UndoManager, acessível através de `window.undoManager`. O histórico guarda dois tipos de alterações:

Alterações DOM
O próprio histórico de alterações do navegador, as alterações DOM são inseridas automaticamente no histórico quando o usuário usa um campo de edição.

Objetos undo
Os objetos undo são inseridos no histórico e controlados pelos seus scripts. Por exemplo, uma aplicação de e-mail pode guardar um objeto undo representando o fato de que o usuário moveu um e-mail de uma pasta para outra.

O objeto UndoManager possui os seguintes métodos e propriedades:

length
o número de entradas no histórico

position
o número da entrada atual no histórico

add(data,title)
adiciona uma entrada específica no histório. `data` *pode ser um objeto literal com dados arbitrários.* `title` *é como essa entrada vai aparecer descrita na lista do histórico*

remove(index)
remove uma entrada específica do histórico

clearUndo()
remove todas as entradas antes da atual no histórico

clearRedo()
remove todas as entradas após a atual no histórico

Além disso, os itens no histórico podem ser acessados com `window.undoManager[index]`.

Respondendo às ações de undo e redo

Cada vez que o usuário disparar uma ação de undo ou redo, e o item do histórico for um objeto undo, será disparado o evento correspondente, window.onundo ou window.onredo. As funções associadas a estes eventos receberão como parâmetro um objeto event, contendo uma propriedade data, cujo valor é o objeto undo que você inseriu no histórico.

Veja o exemplo:

```
window.onundo=function(e){
  alert('Refazer a alteração: '+e.data)
}
```

Disparando as ações de undo e redo

Se você quiser oferecer em sua aplicação botões para undo e redo, basta que eles executem:

```
document.execCommand('undo')
```

Ou:

```
document.execCommand('redo')
```