HTML5 e CSS3

ÍNDICE

1 UMA BREVE HISTÓRIA

Nós não podemos começar a falar sobre HTML e CSS sem contar uma breve história sobre o início de tudo.

Não sei quanto tempo você trabalha com web e não sei qual seu conhecimento sobre história da internet, mas saiba que nunca foi tão fácil desenvolver para web como é hoje. Nós produzimos websites acessíveis hoje por que ocorreram uma série de movimentos independentes entre os fabricantes de browsers e alguns profissionais da área que quebraram barreiras importantes.

Antes nossa preocupação rondava apenas o Internet Explorer 3+ e o Netscape. Estes dois navegadores davam mais problemas de compatibilidade do que todos os problemas de compatibilidades dos browsers atuais, combinados.

Fazer websites com tabela foi um marco para a história do desenvolvimento web mas também o início dos nossos problemas.

Mas não dá para negar que criar websites utilizando tabelas nos trouxe possibilidades interessantes, como por exemplo, criar websites com colunas. Uma maravilha! Imagine só: se antes podíamos apenas inserir texto corrido, centralizado, com alguns gifs animados, agora podíamos dividir o layout em colunas e fazer diagramações matadoras e sofisticados para a época. Um verdadeiro avanço. Estamos falando de algo em torno de 1998.

Mesmo com essas novas possibilidades, descobrimos cedo que desenvolver

websites utilizando tabelas é muito complicado. Ainda mais quando utilizamos o CSS da maneira errada. Naquele tempo era normal misturar o CSS com o HTML. Foi daí que o termo Tag Soup surgiu.

Era muito código! Desenvolvedores aninhavam tabelas como se não houvesse o dia de amanhã. Uma loucura.

O código ficava enorme, fazendo que com o download da página demorasse, a manutenção era terrível e o know-how de como o desenvolvimento foi feito era restrito a alguns profissionais da equipe. Era tudo muito inflexível. Isso encarecia a mão de obra e por isso os projetos ficavam cada vez mais caros.

Lembre-se que além de tudo isso havia a guerra dos browsers. Os problemas de compatibilidade de código entre os dois browsers eram tão terríveis que forçavam o desenvolvedor a criar duas versões de sites para abranger os usuários dos dois navegadores.

A Netscape e a Microsoft tinham uma mente predatória. Cada um tentava ganhar os usuários de forma nada honrosa. Para conseguir tal façanha era fácil: bastasse fazer com que os desenvolvedores fizessem websites compatíveis apenas para um browser, fazendo com que o concorrente ficasse às moscas. Por isso que fazíamos duas versões para cada site, para cada script, para cada HTML.

Essa novela durou longos anos até que um grupo de profissionais gringos decidiram criar um movimento chamado Web Standards Project - WaSP. O desenvolvimento web estava sendo massacrado por causa dessa guerra entre os browsers. O WaSP tinha um objetivo muito claro: fazer com que os browsers e os desenvolvedores seguissem os padrões web recomendados pelo W3C.

O Tim Berners-Lee, o inventor da Web, fundou a World Wide Web Consortium - W3C - para recomendar, criar e manter padrões tecnológicos baseados na web que fossem interoperáveis, abertos e acessíveis. Essas tecnologias devem ser manipuladas para a criação de designs inovadores e também para a

produção de sistemas avançados baseados na internet.

A perseverança deste grupo fez com que os fabricantes de browsers ouvissem suas ideias e suportassem as técnicas e ideias do W3C. Hoje os browsers suportam os padrões web de forma que antes nunca imaginávamos. Isso é bom para todos nós.

Atualmente o objetivo é fazer com que os desenvolvedores estudem e adotem a implementação dos padrões web de forma inteligente.

Ainda há muito o que fazer, principalmente aqui no Brasil. Mas estamos vivendo um cenário muito propício ao crescimento avançado das técnicas de desenvolvimento web. Mais desenvolvedores estão engajados a ensinar, sugerir e fazer com que o mercado de web cresça de forma inteligente. Se você está lendo este livro, você é a prova de que o interesse em uma web mais criativa tem aumentado.

“If not now, when? If not you, who?” WaSP Team.

2 AS 3 CAMADAS DE DESENVOLVIMENTO

O desenvolvimento client-side é baseado em 3 camadas principais: informação, formatação e comportamento.

As camadas possibilitam o desenvolvimento independente de cada área da produção. Se quisermos modificar o design, podemos fazê-lo manipulando apenas o CSS, sem se preocupar com HTML, Javascript ou programação server-side.

Embora sejam independentes, a evolução de cada camada influencia o caminho da outra. O CSS não consegue evoluir se o HTML manter-se congelado no tempo. Um dos principais problemas quando desenvolvíamos com tabelas era a mistura da formatação com a informação. O código HTML estava tão entrelaçado com o código CSS que a manipulação do layout se tornava trabalhosa e muito cara. Não era possível modificar colunas de lugar, características de textos ou até mesmo tamanho dos elementos sem ter que modificar alguma coisa do código HTML. Nada era independente. Esse era um dos motivos que encareciam os projetos para web. Fazer um site entre os anos de 96 e 2001 não era coisa fácil.

Primeira camada: Informação

A camada de informação é a mais importante. Ela vem antes de todas as outras e fica sob o controle do HTML. O HTML **marca a informação dando-lhe significado**. Esse significado é reconhecido por robôs, sistemas, aplicações ou outros meios que podem acessar e reutilizar a informação publicada. A informação precisa ser acessível a qualquer hora, de qualquer lugar e principalmente, por qualquer dispositivo e meio de acesso. Como já dizia o antigo ditado do desenvolvimento web: O conteúdo é o Rei. Mas o que é conteúdo? O que é informação?

Informação é tudo o que o usuário consome.

A Web foi criada para compartilhar informação. Desde o início, quando a internet foi planejada e criada, seu objetivo era claro: compartilhar informação com pessoas do mundo inteiro, de forma rápida e dinâmica. A informação que trafega na web deve ser reutilizada quantas vezes for necessário. Também é importante que a informação seja portável, de forma que ela não seja acessível por apenas um meio.

Entendendo o significado das coisas

Nós, seres humanos, entendemos o significado de cada elemento facilmente. Sabemos que um título é diferente do parágrafo por causa das suas características visuais: tamanho de fonte, quantidade de caractéres, escrita e etc...

As máquinas (e quando digo máquinas, quero abranger sistemas de busca, leitores de tela, smartphones, browsers, aplicações, sistemas, etc. Em suma, qualquer meio de acesso) não tem esse discernimento. Os robôs de sistemas de busca, por exemplo, não enxergam. Eles não tem como entender visualmente o que é um título, um parágrafo, uma imagem etc. Essa é a importância da marcação HTML: ela define o que é cada informação.
Tenha em mente que o HTML deve ser sempre a primeira camada. Se todas as outras camadas não funcionarem por algum motivo, o HTML

deve funcionar. A informação deve ser entregue, não importa se o visual tenha sido prejudicado por falta do CSS ou se o Javascript está desligado no browser do usuário.

Segunda camada: Formatação

A segunda camada é responsável por controlar o visual da informação exibida pelo HTML. É esta camada que deixa tudo bonito. Que faz vender. Que enche os olhos do cliente. Atualmente essa camada é controlada pelo CSS e parece que será por muito tempo ainda.

O CSS é a linguagem responsável por controlar o visual da informação exibida pelo HTML. O CSS formatará o conteúdo de forma que seja visualmente agradável em qualquer meio de acesso. A informação é acessada por diferentes meios de acesso, desde sistemas de busca até aparelhos como tablets, smartphones etc e o CSS é o **responsável por formatar a informação para que ela seja consumida em qualquer meio de acesso** de forma simples. Se você estiver usando um leitor de tela, o CSS poderá controlar a maneira com que a voz do leitor sairá pelas caixas de som. Controlamos também a forma com que a informação é mostrada em TVs ou outros aparelhos. Controlamos como o texto será exibido em uma impressão.

Quero que você abra sua mente quando ler em uma mesma frase as palavras **CSS** e **formatação**. Formatação com CSS quer dizer muito mais do que pintar divs, formatar letras e cores. Se a informação deve ser acessada em qualquer lugar, o CSS precisa cuidar para que essa informação apareça de maneira adequada em cada um destes meios de acesso. Essa é sua principal responsabilidade.

Vamos ver mais sobre a responsabilidade do CSS e alguma técnicas nos capítulos posteriores.

Terceira camada: Comportamento

Nessa terceira camada você decidirá quais serão os comportamentos dos elementos. O Javascript até hoje é o principal responsável por essa camada. Com o Javascript você definirá se os elementos serão arrastados, dimensionados, rotacionados, reformatados etc. O Javascript controla tudo isso manipulando as características dos elementos pelo CSS. Resumidamente, o Javascript controla os valores definidos pelo CSS e manipula estas propriedades.

Você verá no decorrer deste livro que o HTML5 trouxe muitas ferramentas e possibilidades para que o Javascript controle os elementos criados no código utilizando as novas APIs do HTML5

Nessa camada não importa se você utilizará Javascript ou se intermediará seu código com um framework como o JQuery.

O CSS também está com um pé nessa camada. Com o CSS3 podemos controlar comportamentos simples dos elementos, começando com animações e transições. Mesmo assim o CSS não será (talvez) uma ferramenta para fazer animações complexas como as animações que fazemos com SVG ou Canvas.

3 HTML - A ALMA DO CLIENT-SIDE

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.

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

Para entender melhor como o HTML funciona, vamos conversar sobre duas palavras que fazem parte do seu nome: Markup (marcação) e Hypertext (hipertexto).

Hipertexto

O HTML é baseado no conceito de Hipertexto, que é uma forma de organizar conteúdo de forma não linear. 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.

Quando a web foi criada, era necessário distribuir a informação de uma maneira simples mas organizada, era necessário então haver uma linguagem que fosse 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 e seus conceitos.

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. 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://bit.ly/ol3an1*.

Desde o começo o HTML foi criado para ser uma linguagem independente de plataformas, browsers e outros meios de acesso. Interoperabilidade significava menos custo. Você cria apenas um código 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.

Marcação

Ao ler um livro ou uma revista, você consegue distinguir títulos de parágrafos porque você é um ser humano. Você sabe que os títulos tem letras maiores, poucas palavras etc... Já o parágrafo tem um número maior de palavras e a sua letra é menor. Essa distinção é clara e muito óbvia visualmente.

A web não é apenas acessada por seres humanos. A informação publicada é totalmente reutilizada pelos meios de acesso. Os meios de acesso são qualquer coisa que acesse a web e consuma seu conteúdo. Pode ser os sistemas de busca, seu browser, um leitor de tela, seu smartphone, ou qualquer outro sistema ou dispositivo utilizado pelos usuários ou robôs. Estes meios de acesso não conseguem distinguir visualmente os elementos exibidos na tela. É por isso que o HTML é baseado em marcação.

Nós marcamos a informação, dando significado a estes objetos, tornando-os legíveis para os meios de acesso. Assim, quando marcamos um título com h1, h2 ou h3, indicamos para os meios de acesso que determinado bloco de texto é um título, com uma determinada importância e assim por diante com os outros elementos.
Dessa forma cada meio de acesso pode decidir o que fazer com esta informação. O browser, por exemplo, carrega a informação na tela do usuário. Já o robô do

Google guarda a informação e a utiliza em suas buscas. Cada meio de acesso tem sua função, logo, utiliza a informação de maneiras diferentes.

Daí vem a importância da semântica no código HTML. Se você marca um título com um elemento que não é da família de títulos, os meios de acesso tratarão essa informação de maneira errada. Não importa se você formate esse elemento de maneira que ele se pareça com um título visualmente. O que importa mesmo é o que está escrito no código.

O INÍCIO DO HTML5

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 WHATWG1 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 o WHATWG se focava em um padrão chamado Web Forms 2.0, que se propunha a modificar os formulários do HTML, ele foi incluído no HTML5 e o Web Controls 1.0 que foi abandonado por enquanto.

Por volta de 2006, o trabalho do WHATWG passou a 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. Mesmo assim o XHTML seria mantido paralelamente de acordo com as mudanças causadas no HTML. O grupo que estava cuidando especificamente do XHTML 2 foi descontinuado em 2009.

A participação no grupo que decide estes padrões é livre e você pode se inscrever na lista de email2 para contribuir. Sugiro fortemente que você faça isso. Você estará bebendo direto da fonte e poderá contribuir diretamente para a criação de uma Web mais livre e flexível.

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.

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, por exemplo, 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.

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, sem a necessidade de criações de grandes blocos de scripts.

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, tags como B e 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 precisará 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. A regra primordial entre os desenvolvedores web Don’t Break The Web é seguida à risca.

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 suporte dos browsers e o uso dos

desenvolvedores. Era feito um lançamento único para a linguagem inteira.

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 etc. Cada um destes grupos são independentes e podem lançar suas novidades a qualquer momento. Logo, o desenvolvimento ficou mais dinâmico, com novidades mais constantes.

A desvantagem desta constância 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. Sendo assim, os browsers tem mostrado grande interesse em se manterem atualizados em relação aos seus concorrentes. A Guerra dos Browsers está ainda presente no desenvolvimento web, e isso é bom. Não é uma guerra predatória, onde um browser tenta minar o mercado do outro, mas é uma guerra competitiva, saudável para o mercado.

Estrutura básica, DOCTYPE e charsets

A estrutura básica do HTML5 continua sendo praticamente a mesma das versões anteriores, mas com menos código. Segue abaixo como a estrutura básica pode ser seguida:

```
<!DOCTYPE html>
<html lang=”pt-br”>
<head>
```

```
    <meta charset="UTF-8">
    <title></title>
</head>
<body>
    Corpo do site.
</body>
</html>
```

O Doctype

O Doctype deve ser sempre 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. Isso determina a forma com que o meio de acesso irá renderizar o código lido. 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 (DOM, lembra?). 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 um novo atributo. Ele já existe desde

muito tempo e não é restrito ao elemento HTML, sendo possível ser utilizado em qualquer outro elemento para indicar o idioma do texto representado. O que é muito bom para leitores de tela.

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

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

O novo código para o HTML5 resumiu bastante as coisas, retirando todo o código inútil para nós, deixando a linha acima assim:

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

A Web é acessada por pessoas do mundo inteiro. Ter um sistema ou um site que limite o acesso de pessoas de outros países é algo que vai contra a tradição e os ideais da internet. Por isso, foi criado uma tabela que suprisse as necessidades impostas pelas linguagens do mundo todo, essa tabela chama-se 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 existir uma tabela padrão com o maior número de caracteres possível,

dando a possibilidade de guardarmos todos as letras e símbolos utilizados nos idiomas do mundo. O que o Unicode faz é fornecer um único número para cada caractére, não importa a plataforma, nem o programa, nem a língua.

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.

Tag LINK

Há dois tipos de links no HTML: a tag A, que é a abreviação de ANCHOR, que são elementos que levam o usuário para outros documentos o famoso Hiperlink. E há a tag LINK, que é uma tag para importar objetos externos que serão usados no documento, como arquivos de CSS, RSS, etc. Por isso não se confunda.

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. Essas regras são chamadas de **Modelos de Conteúdo**.

Dentre todas as categorias de modelos de conteúdo, existem dois tipos de elementos: elementos de linha e os elementos de bloco. Os elementos de linha marcam, na maioria das vezes, textos, elementos de formulários, imagens. Alguns exemplos: a, strong, em, img, input, abbr, span. Já os elementos de blocos são como caixas, que dividem o conteúdo em 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. Mas o inverso é possível.
- 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 inverso é possível.

Estes dois grandes grupos podem ser divididos em categorias. Estas categorias dizem qual modelo de conteúdo o elemento se encaixa e como pode ser seu comportamento.
Os navegadores utilizam muito estas regras para definir o fluxo de informação e quais as suas utilizações. Alguns browsers podem ajudar o desenvolvedor tentando consertar algum erro de sintaxe. Outros simplesmente quebram o layout ou a aplicação no local que o erro de sintaxe acontece. Tome cuidado

com essas diferenças de comportamento. É interessante entender exatamente como cada browser se comporta nestas situações.

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

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	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 até h6	header
hgroup	hr	i	iframe	img
input	ins	kbd	keygen	label
link	map	mark	math	menu
meta	meter	nav	noscript	object
ol	output	p	pre	progress
q	ruby	samp	script	section
select	small	span	strong	style
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 seções principais no código. Como por exemplo rodapés, cabeçalhos, área do texto principal, listas de navegações etc.

header	footer	article	aside	nav	section

Aqui é onde o HTML5 inseriu mais tags novas. Estas novas tags vieram para melhorar a semântica dos elementos estruturais do código. Antes era quase impossível identificar de maneira automática o que era um rodapé ou um

header de página. Era dificílimo localizar o bloco com o texto principal de uma página. Com estas tags esse trabalho ficou mais simples.

Heading content

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

h1 até h6	hgroup	header

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	audio	b
bdo	br	button	canvas	vite
code	command	datalist	del	dfn
em	embed	i	iframe	img
input	ins	kbd	keygen	label
link	map	mark	math	meta
meter	noscript	object	output	progress
q	ruby	samp	script	sup
select	small	span	strong	sub
svn	textarea	time	var	video
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	objevt	svg	video	

Interactive content

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

a	audio	button	details	embed
iframe	img	input	keygen	label
menu	object	select	textarea	video

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

O código HTML indica 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 letras, 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 e rederizadas 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. Conseguíamos diferenciar por exemplo,

um parágrafo de um título. Mas não conseguíamos diferenciar o rodapé do cabeçalho. Essa diferenciação era apenas percebida visualmente com o layout pronto. Também 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 criou uma série de elementos que nos ajudam a definir os setores principais no documento. 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 está 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.

Elementos de seção

Você deve conhecer o elemento DIV. O DIV tem a função de criar divisões. Quando criamos um website, dividimos as áreas do layout em seções. O problema do DIV, é que ele não tem nenhum significado semântico. Os sistemas de busca, por exemplo, não tem como saber o que é um rodapé, um cabeçalho, sidebar e etc, porque tudo é feito com DIVs e assim damos o mesmo nível hierárquico de informação para todas as seções.

Para resolver esse problema, foi criado um conjunto novo de elementos que além de dividir as áreas do layout, ele entrega significado. Esses elementos são chamados de Conteúdos de Seções ou Sectioning content.

Esse conjunto de tags tem a função de dividir as áreas do layout como fazíamos com os DIVs, mas cada uma delas carrega um significado específico. Entenda agora com mais detalhes o que cada um destes elementos significa.

O elemento 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, destaque, novidades, informação de contato e chamadas para conteúdo interno.

Basicamente o elemento section substitui o div em muitos momentos. A imagem seguinte foi retirada do site Globo.com.

Cada uma das colunas dos assuntos Notícias, Esportes e Entretenimento é definido e marcado por um elemento section em vez de utilizarmos divs como era de costume.

Isso permite os sistemas de buscas ou outras aplicações saibam que cada um daqueles blocos trata-se de um assunto diferente. Meu desejo é que futuramente seja possível indicar qual tipo de assunto é abordado em cada

um destes elementos.

Entenda que o section apenas divide estes assuntos. Não é como o header, footer, aside, nav e article que dividem informações específicas, como veremos a seguir. O section é um elemento mais específico que o div, mas não mais específico que estes elementos. Entenda que todos os outros objetos que veremos também são chamados de sections, mas cada um deles é responsável por uma parte da página: o header pelos cabeçalhos, o nav pelas listas de navegações e assim por diante. O section muitas vezes envolverá todos estes elementos, separando-os como um assunto.

O elemento 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. Aqueles links que são considerados principais, por exemplo o menu principal do site. No caso dos portais, aquele menu lateral com uma série de links separados por assuntos poderiam ser uma nav. Isso também pode ser aplicado para aqueles blocos de links que geralmente são colocados no rodapé.

Imagine uma sidebar com uma série de links, como por exemplo a sidebar de um portal de conteúdo como o G1, R7, UOL e etc… Nestes portais é normal você encontrar diversos links para as categorias de assuntos. Anteriormente, se quiséssemos agrupar por exemplo um Título e uma lista de links faríamos assim:

```
<div class=”categ categ-esporte”>
  <h3>Esporte</h3>
  <ul>
    <li><a href=”#”>Copa 2014</a></li>
    <li><a href=”#”>Brasileirão</a></li>
    <li><a href=”#”>Fórmula 1</a></li>
    <li><a href=”#”>Baskete</a></li>
  </ul>
</div>
```

Este código resolvia nosso problema de formatação. Poderíamos utilizar o div que envolve o título e a lista para fazer algum detalhe visual e principalmente para agruparmos o conteúdo relacionado que existia. Mas a nível de informação não havia nenhuma indicação de que o título estivesse ligado ao conteúdo. Não há nenhuma referência de que o título ESPORTE apresenta a lista de links abaixo dele. Visualmente essa relação é muito forte porque o designer desenhou o layout para que isso acontecesse, mas a nível de código nós não tínhamos como indicar essa relação de informações. Poderíamos fazer apenas para que o código ficasse organizado, mas para os leitores de tela ou sistemas de busca essa relação simplesmente não existia. Os sistemas de busca não podem se basear apenas na posição dos elementos, é algo muito genérico para eles confirmarem que a lista e o título que a precede estão ligados em um mesmo assunto.

Com o HTML5, isso muda. Veja o código abaixo:

```
<nav>
  <h3>Esporte</h3>
  <ul>
    <li><a href=”#”>Copa 2014</a></li>
    <li><a href=”#”>Brasileirão</a></li>
    <li><a href=”#”>Fórmula 1</a></li>
    <li><a href=”#”>Baskete</a></li>
  </ul>
</nav>
```

Com a tag nav, há uma indicação de que aquele grupo é uma seção (nav é um tipo de section. Enquanto a tag section serve para indicar seções no site, a tag nav indica que um determinado grupo é uma seção de navegação) é um bloco de navegação.
Dentro da nav você pode agrupar uma série de listas de links:

```
<nav>
  <h3>Esportes</h3>
  <ul>
    <li><a href=”#”>Copa 2014</a></li>
    <li><a href=”#”>Brasileirão</a></li>
```

```
      <li><a href="#">Fórmula 1</a></li>
      <li><a href="#">Baskete</a></li>
    </ul>

  <h3>Educação</h3>
    <ul>
      <li><a href="#">Educação</a></li>
      <li><a href="#">Dicionários</a></li>
      <li><a href="#">Vestibular</a></li>
    </ul>

  <h3>Notícias</h3>
    <ul>
      <li><a href="#">Cotidiano</a></li>
      <li><a href="#">Política</a></li>
      <li><a href="#">Jornais</a></li>
    </ul>
  </nav>
```

A tag nav também pode estar em todos os elementos do HTML. Você pode colocá-la no header para definir menus, no footer para definir grupos de links, sidebars, articles e etc.

Com um bloco de navegação definido, a acessibilidade da página pode ser explorada. Os leitores de tela, sistemas de busca ou qualquer outro sistema interessado, podem localizar facilmente os links importantes e internos da página por meio da tag nav.

Entenda que o nav não carrega qualquer tipo de links. Tenha em mente em sempre que usar o nav, ele irá carregar grupos de links ligados ao site. Normalmente estes links são links internos, e por conta disso, o ranqueamento e a indexação feita pelos buscadores pode ser melhorada.

O elemento Article

A especificação diz:
"The article element represents a component of a page that consists of a self-contained composition in a document, page, application, or site and that is intended to be independently distributable or reusable, e.g. in syndication. This could be a forum post, a magazine or newspaper article, a blog entry, a user-submitted comment, an interactive widget or gadget, or any other independent item of content."

O elemento article é onde colocamos o texto ou a informação principal. Imagine que você está visitando um blog, há uma sidebar e há também o bloco de texto, que seria o bloco que carrega o conteúdo principal do site. Este bloco seria marcado como article. Dentro deste article haverá toda a informação necessária sobre o artigo: data de publicação, título, autor, o texto, informações de outros artigos relacionados etc.

Para entender melhor, a informação que vai dentro do article é exatamente a mesma informação que os leitores de feeds capturam do seu RSS/Atom. O leitor de FEED não disponibiliza o menu do site, a sidebar e etc, ele apenas disponibiliza o texto principal do post, e será esse texto que haverá no elemento article.

Um exemplo de código:

```
    <article>
       <header>
          <h1>Título do post</h1>
          <time datetime="12-03-1983" pubdate="pubdate">03 de
Dezembro de 1983</time>
          <p>Texto de introdução.</p>
       </header>

        <h2>Um outro título</h2>
        <p>Texto do post.</p>
        <p>Texto do post.</p>
        <p>Texto do post.</p>
    </article>
```

Alguns podem confundir os elementos de article, section e div. Entenda que

o article é um elemento mais específico que o section e o div. O article marca o texto principal do site. É o filé da página. O section é apenas um bloco de separação de assuntos diferentes. O div, o mais genérico de todos, é apenas aplicado para separar elementos em blocos, por isso ele não carrega nenhum significado semântico.

Sendo mais específico: Para marcar informações e conteúdos que fazem sentido se forem vistos de fora do site, como em leitores de RSS, utilize o article. Ou seja, o que você mostra no Feed? Menu, rodapé, header, sidebar ou somente o bloco de texto do artigo? Pois é isto que você marcará com a tag article.

Para separar e organizar conteúdos de diversos assuntos em blocos diferentes, utilize o section.

Para utilização não semântica, para formatação com CSS e detalhes genéricos, utilize o div.

O elemento ASIDE

O elemento aside representa um bloco de informação relativa ao conteúdo principal.

Algumas utilidades do aside: citações ou sidebars, agrupamento de publicidade, grupos e blocos de navegação ou para qualquer outro conteúdo que tenha importância secundária e relativa ao conteúdo principal da página.

Dentro do aside você pode agregar por exemplo grupos de elementos nav, headers, sections e etc, permitindo produzir um menu lateral com os grupos de informações:

```
<aside id=”menulateral”>
   <nav>
       <h3>Esportes</h3>
       <ul>
          <li><a href=”#”>Fórmula 1</a></li>
          <li><a href=”#”>Futebol</a></li>
```

```
            <li><a href="#">Baskete</a></li>
            <li><a href="#">Voley</a></li>
        </ul>
    </nav>

    <nav>
        <h3>Política</h3>
        <ul>
            <li><a href="#">Eleições 2010</a></li>
            <li><a href="#">Urna eletrônica</a></li>
            <li><a href="#">Candidatos</a></li>
        </ul>
    </nav>
</aside>
```

Note que não utilizamos nenhum div, pelo contrário, utilizamos apenas tags que trazem significado semântico. Neste exemplo, indicamos - para o navegador ou aplicação, sistema de busca ou qualquer outra coisa que acessará nosso código - que aquele é bloco lateral, e que cada grupo de nav é referente a um assunto.

O elemento aside também pode estar dentro de um elemento article, como uma caixa de notação ou algo do gênero. Nesse caso, quando o usuário imprimir, você pode dar ênfase como se fosse uma caixa de informação. Como é feito em livros ou revistas.

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 href, lang 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.

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.

O W3C mantém um documento atualizado neste link: *http://bit.ly/r5S9SX*.

ELEMENTOS MODIFICADOS OU 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. Acrônimos são abreviações, mas são um pouco diferentes. Acrônimos são abreviações que formam siglas, por exemplo N.A.S.A.

applet
Ficou obsoleto em favor da tag object.

isindex
Foi substituído pelo uso de form controls.

dir
Ficou obsoleto em favor da tag ul.

Compatibilidade do 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.

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. A boa notícia é que a partir de Janeiro de 2012 a Microsoft ativará o autoupdate dos sistemas WindowsXP e Windows Vista, fazendo com que o IE6 seja atualizado automaticamente. Desenvolver para IE8 e 9 é muito melhor do que desenvolver para IE6 e 7. Aleluia!

Abaixo segue uma tabela simples de compatibilidade entre os browsers e alguns módulos do HTML5 até a escrita deste livro:

Módulos	Safari	Chrome	Opera	Firefox	IE8	IE9
Local Storage	sim	sim	sim	sim	sim	sim
Histórico de Sessão	sim	sim	sim	sim	sim	sim
Aplicações Offline	sim	sim	não	sim	não	não
Novos tipos de campos	sim	sim	sim	não	não	não
Form: Autofocus	sim	sim	sim	não	não	não
Form: Autocomplete	não	não	sim	não	não	não

Form: Required	sim	sim	sim	não	não	não
Video, Audio e Canvas Text	sim	sim	sim	sim	não	sim

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:

- Verifique se uma determinada propriedade existe em objetos globais como WINDOW ou NAVIGATOR. Nesse caso, verificamos o suporte a geolocalização.
- Crie um elemento e verifique se uma determinada propriedade existe neste elemento.
- 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.
- 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 a Biblioteca Modernizr

Alguns browsers não aceitam as novas features de CSS3 e HTML5. Saiba como detectá-los e tratá-los com a biblioteca Modernizr.

Problemas de compatibilidade

Quando produzimos um site os problemas de compatibilidade fazem parte da regra do jogo. Para tentar contornar estes problemas utilizamos hacks, comentários condicionais, sniffing de browsers e outras coisas, que muitas vezes mais prejudicam do que ajudam.

Para ajudar mais ainda o CSS3 e o HTML5 apareceram derrubando tudo, e o problema de compatibilidade que já era chato, ficou mais chato que meia molhada. Embora os browsers estejam muito mais atuais e suportando propriedades avançadas de CSS3 e HTML5, não é garantia que todos eles suportem as mesmas propriedades. E é aqui que começamos a ter problemas novamente, como no passado.

Como você consegue reconhecer quem um determinado browser suporta CSS Animation? Como você sabe que o browser conhece LocalStorage do HTML5? Você não vai ficar olhando numa tabelinha toda vez que tiver essas dúvidas para fazer um visual ou uma solução alternativa para tais browsers.

É por essas e outras que você utilizará a Modernizr.

O que é a Modernizr

Modernizr é uma pequena biblioteca Javascript que detecta a disponibilidade das novas características do HTML5 e CSS3 nos browsers. Muitas destas características já estão implementadas nos browsers, mas é muito chato você decorar quais novidades os browsers já estão suportando. O que a Modernizr faz é simples: ela te diz quais features um determinado browser suporta e insere classes no HTML para que você possa utilizar para fazer uma versão

alternativa de visual ou solução.

Entenda que a Modernizr não é um sniffing de browser. Ela é diferente. A Modernizr faz o trabalho de detectar das seguintes formas:

- Ela testa 40 features de CSS3 e HTML5 em alguns milisegundos.
- Depois ela cria objetos javascript que contém os resultados destes testes.
- Aí são adicionadas classes no elemento HTML descrevendo exatamente quais propriedades e novidades são ou não nativamente suportadas.
- Depois disso você consegue ter os resultados descritos nos navegadores dinamicamente e então pode tomar decisões criando alternativas para aquelas propriedades não suportadas pelos browsers antigos.

Como funciona

É simples: primeiro você baixa a versão mais atual da biblioteca no endereço *http://www.modernizr.com/*. O interessante é que você tem a opção para personalizar a biblioteca, indicando quais features você quer que a Modernizr teste no seu projeto.

Depois você inclui esse pacote no seu HTML:

```
<!DOCTYPE html>

<html lang="pt-br">
<head>
    <meta charset="utf-8">
    <title>Teste de Modernizr</title>
    <script src="modernizr-2.0.6.js"></script>
</head>
<body>

</body>
</html>
```

Feito isso, insira uma classe no-js no elemento HTML:

```
<html class="no-js" lang="pt-br">
```

Quando a Modernizr rodar, ela irá substituir essa classe para uma js se o browser estiver com o Javascript ligado, já te dando um feedback para tomar alguma atitude se o usuário estiver com o Javascript desligado.
Junto com essa mudança são adicionadas outras classes, indicando o que o browser aceita nativamente ou o que ele não aceita. Ficará algo parecido com isso:

```
    <html class=" js flexbox canvas canvastext webgl
no-touch geolocation postmessage websqldatabase no-
indexeddb hashchange history draganddrop websockets rgba
hsla multiplebgs backgroundsize borderimage borderradius
boxshadow textshadow opacity cssanimations csscolumns
cssgradients cssreflections csstransforms csstransforms3d
csstransitions fontface generatedcontent video audio
localstorage sessionstorage webworkers applicationcache
svg inlinesvg smil svgclippaths" lang="pt-br">
```

O browser que eu utilizei é o Safari/Mac. Pelo visto ele aceita bastante coisa. ;-)
O que o browser não aceita, a Modernizr insere uma classe com o prefixo no- antes da classe, por exemplo: no-boxshadow, no-geolocation, no-touch etc.

A Modernizr também cria um objeto Javascript contendo um valor booleano para cada uma dessas features, possibilitando a criação de testes. Um exemplo:

```
if (Modernizr.geolocation) {
       alert("Aceita")
} else {
       alert("Não Aceita")
}
```

Exemplos de utilidade

Exemplo bem básico: imagine que você queira utilizar o box-shadow em seu projeto. Browsers como o IE6,7,8 não reconhecem essa feature, então seria interessante darmos uma alternativa, como por exemplo, colocando uma borda em vez de sombra. Assim o elemento não fica tão diferente do que deveria.

Como a Modernizr colocou uma classe no elemento HTML referente a

aceitação das features, podemos utilizá-la fazendo assim:

```
.loginBox {
    box-shadow:0 10px 10px rgba(0, 0, 0, 0.3);
}

.no-boxshadow .loginBox {
    border: 1px solid #CCC;
    border-bottom: 3px solid #CCC;
}
```

Assim, se o browser não aceitar a propriedade box-shadow o usuário verá uma borda no lugar. Você pode fazer isso com praticamente qualquer nova feature do CSS3 e do HTML5.

Ah, mais uma coisa: provavelmente você já utiliza um scriptzinho html5.js para fazer com o que os Internet Explorers reconheçam as tags do HTML5, correto? O Modernizr já faz isso automaticamente. Sugiro que pare de utilizar o html5.js e passe a utilizar a Modernizr somente.

A Modernizr facilita demais as coisas. A ideia é que você não prive seus projetos da utilização de features novas. A produção vai ficar mais eficaz e seu projeto sempre estará atualizado com as melhores práticas do mercado. Adote a Modernizr e seja feliz.

4 FORMULÁRIOS E MULTIMÍDIA

NOVOS TIPOS DE CAMPOS

Quando o grupo WHATWG resolveu reescrever o HTML, eles decidiram começar pelos formulários. Se você perceber, estamos escrevendo formulários da mesma maneira desde o início de tudo. Nunca houve uma atualização sensata e interessante que modificasse a forma com que usávamos e submetiamos informações via formulários.

Foi aí que o grupo decidiu criar alguns novos tipos de formulários, facilitando a usabilidade do usuário e facilitando a vida do desenvolvedor ao produzir campos de formulários.

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

tel

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

search

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

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

email

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

url

Um endereço web, também com formatação e validação.

DATAS E HORAS

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.

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.

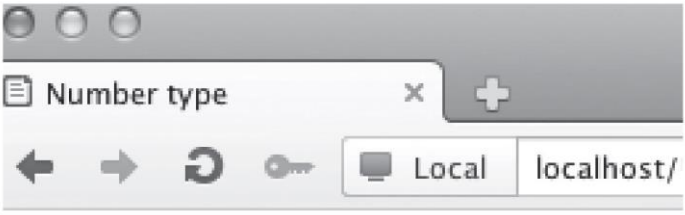
Oatributo adicional step define, para os validadores e auxílios ao preenchimento, a diferença mínima entre dois horários. O valor de step é em segundos, e o valor padrão é 60. Assim, se você usar step=”300” o usuário poderá fornecer como horários 7:00, 7:05 e 7:10, mas não 7:02 ou 7:08.

number

Veja um exemplo do tipo number com seus atributos opcionais:

```
<!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:

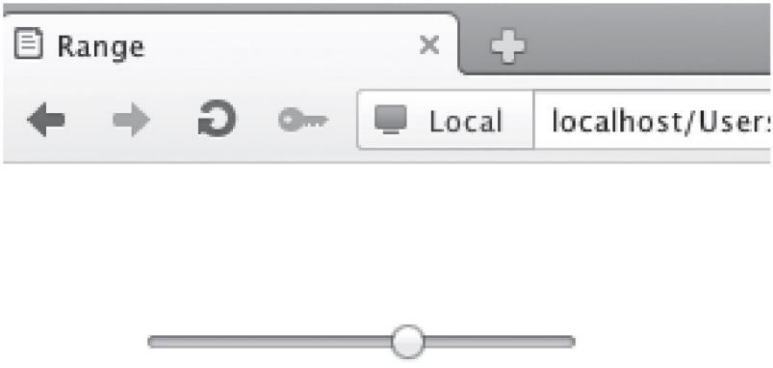


range

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

```
<!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.

O elemento <input> tem novos valores para o atributo type.

tel
O elemento representa um controle para a edição de um número de telefone. Quebras de linhas são automaticamente removidas do valor de entrada, mas nenhuma outra sintaxe é aplicada, poisnúmeros de telefone podem variar bastante internacionalmente. Você pode usar atributos como pattern e maxlength para restringir valores digitados no controle.

search
O elemento representa um campo de entrada de busca. Quebras de linhas são automaticamenteremovidas do valor de entrada, mas nenhuma outra sintaxe é aplicada.

url
O elemento representa um controle para a edição de URL. Quebras de linha e espaços, à direitae à esquerda, são automaticamente removidos do valor de entrada.

email
O elemento representa um endereço de e-mail. Quebras de linhas são automaticamente removidas do valor de entrada. Um endereço de e-mail inválido pode ser configurado, mas ocampo de entrada somente será satisfeito se contiver um endereço de e-mail que satisfaça a produção ABNF.

“Nota: se o atributo multiple é configurado, podem ser digitados múltiplos endereços de e-mail neste campo <input>, como uma lista separada por espaços, mas atualmente isto não está funcionado no Firefox.”

datetime

Defini controle para uma data e hora com fuso horário

date

Define controle para uma data

month

Defini controle que indica o mês na qual o dia pertence.

week

Defini controle que indica a semana na qual o dia pertence.

time

Defini controle para informar a hora escolhida.

datetime-local

Definir controle para data e hora sem fuso horário

number

disponibiliza setas para aumentar e diminuir os números sendo possível limitar os númeromáximo e mínimo;

range

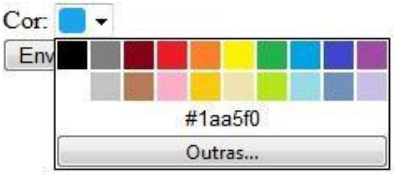
Se o elemento receber o atributo, ele representa um controle para definir um valor numéricoimpreciso.

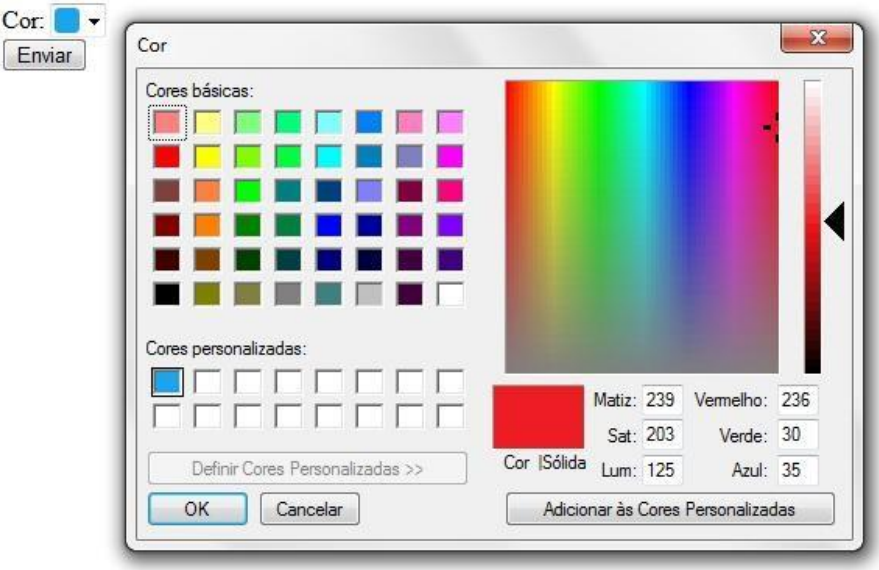
color

Se o elemento receber o atributo type=color, ele representa um controle que abrirá uma palhetade cores que retorna o valor hexadecimal que representa a cor escolhida. Este valor é uma seqüência sete caracteres, composto das seguintes partes e ordem:

Um caractere “#”.
Seis caracteres no intervalo 0-9, a-f, e A-F.

Observação: Palavras-chave com o nome da cor(por exemplo, seqüências como “red” ou“green”) não são permitidos.





Exemplo completo no documento.
exemplo_formtypes.html

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:

```
<!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:

```
<!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:

```
    <!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:

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

```
            label{display:block;}
        </style>
    </head>

    <body>

    <form>
        <label>nome: <input name="nome" required></
label>
        <label>email: <input name="email" type="email"
required></label>
        <label>mensagem: <textarea name="mensagem"
required></textarea></label>
        <input type="submit" name="action" value="Salvar
rascunho" formnovalidate>
        <input type="submit" name="action"
value="Enviar">Formulários e Multimídia


    </form>

    </body>

    </html>
```

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:
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.

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:

```
<!DOCTYPE html>
<html lang=”pt-BR”>
<head>
    <meta charset=”UTF-8”>
    <title>Custom validator</title>
    <!-- O arquivo cpf.js contém a função validaCPF,

    recebe uma string e retorna true ou false. -->
    <script src=”cpf.js”></script>
```



```
        <script>
            function vCPF(i){
              i.setCustomValidity(validaCPF(i.
value)?’’:’CPF inválido!’)
            }
        </script>
    </head>

    <body>
        <form>
            <label>CPF: <input name=”cpf”
oninput=”vCPF(this)”></label>
            <input type=”submit” value=”Enviar”>
        </form>
    </body>

    </html>
```

DETALHES E CONTEÚDO EDITÁVEL

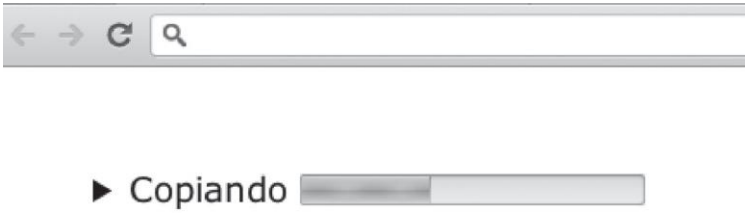
Vejamos mais duas coisas que você certamente já fez mais de uma vez e foram simplificadas pelo HTML5.

DETALHES E SUMÁRIO

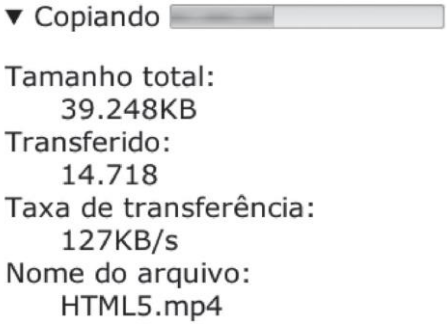
Veja um exemplo de uso dos novos elementos details e summary:

```
    <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.

```
    <!DOCTYPE HTML>
    <html>
    <head>
        <meta content="text/html; charset=UTF-8"
http-equiv="content-type"/>
        <title>HTML5 Drag and drop demonstration</title>
        <style type="text/css">
            #boxA, #boxB {
              float:left; width:100px; height:200px;
padding:10px; margin:10px; font-size:70%;
            }
            #boxA { background-color: blue; }
            #boxB { background-color: green; }

            #drag, #drag2 {
              width:50px; padding:5px; margin:5px;
border:3px black solid; line-height:50px;
            }
            #drag { background-color: red;}
            #drag2 { background-color: orange;}
        </style>

        <script type="text/javascript">

            // Quando o usuário inicia um drag,
guardamos no dataset do evento
            // o id do objeto sendo arrastado
            function dragStart(ev) {
                ev.dataTransfer.setData("ID", ev.target.
getAttribute('id'));
            }

            // Quando o usuário arrasta sobre um dos
painéis, retornamos
            // false para que o evento não se propague
para o navegador, o
            // que faria com que o conteúdo fosse
selecionado.
            function dragOver(ev) { return false; }
```

```
            // Quando soltamos o elemento sobre um
painel, movemos o
            // elemento, lendo seu id do dataset do

evento

function dragDrop(ev) {
    var idelt = ev.dataTransfer.

getData(“ID”);
                ev.target.appendChild(document.
```



```
getElementById(idelt));
            }

        </script>
    </head>
    <body>
        <!-- Painel 1 -->
        <div id=”boxA”
        ondrop=”return dragDrop(event)”
        ondragover=”return dragOver(event)”>

        <!-- Draggable 1 -->
        <div id=”drag” draggable=”true”
        ondragstart=”return dragStart(event)”>drag me</
div>

        <!-- Draggable 2 -->
        <div id=”drag2” draggable=”true”
        ondragstart=”return dragStart(event)”>drag me</
div>
        </div>

        <!-- Painel 2 -->
        <div id=”boxB”
        ondrop=”return dragDrop(event)”
        ondragover=”return dragOver(event)”>
        </div>

    </body>
    </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"'>
```

ELEMENTO DEVICE E STREAM API

O ELEMENTO DEVICE

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/*

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:

```
<!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.

5 A NOVA GERAÇÃO DE APLICAÇÕES WEB I

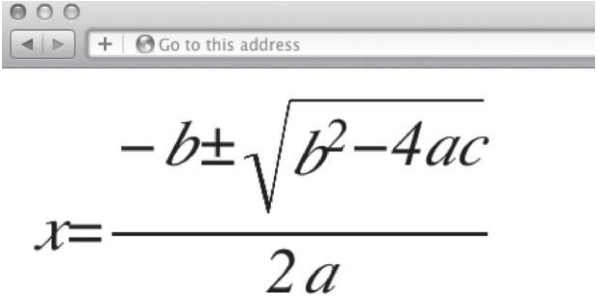
MATHML

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:

```
<!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:



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 ⁢, 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 ⁢? Você vai notar que se remover a entidade e a tag mo correspondente o resultado visual será o mesmo. Colocamos ⁢ 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.

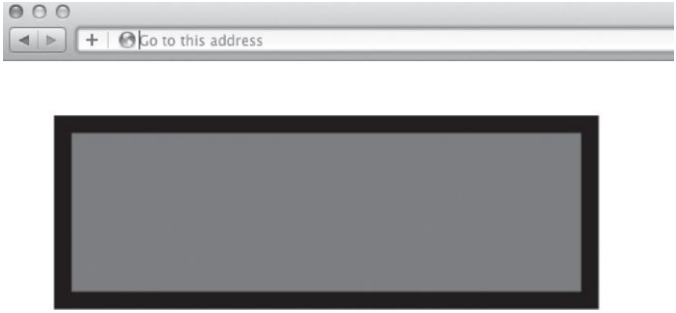
SVG

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:

```
<!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:

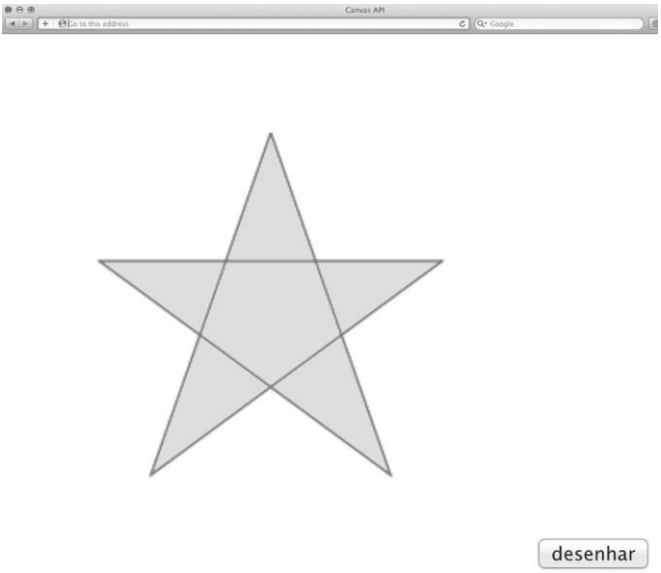
```
context.fillRect(10, 10, 50, 150)
```

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

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

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

<script>
function desenhar(){
```

```
      // Obtemos o contexto
      context=document.getElementById('x').
getContext('2d')

      //Iniciamos um novo desenho
      context.beginPath()

      //Movemos a caneta para o inicio do desenho
      context.moveTo(150,50)

      //Desenhamos as linhas
      context.lineTo(220,250)
      context.lineTo(50,125)
      context.lineTo(250,125)
      context.lineTo(80,250)
      context.lineTo(150,50)

      //O desenho não é de verdade enquanto você
      //não mandar o contexto pintá-lo.

      //Vamos pintar o interior de amarelo
      context.fillStyle='#ff0'
      context.fill()

      //Vamos pintar as linhas de vermelho.
      context.strokeStyle='#f00'
      context.stroke()

    }
    </script>

    </body>
    </html>
```

E veja o que acontece quando se clica no botão:



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

Canvas vs SVG

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

Assim, as vantagens do SVG são:

- O conteúdo é acessível a leitores de tela

- O gráfico é escalável, não perde resolução ou serrilha ao redimensionar
- O conteúdo é acessível via DOM

E as vantagens do Canvas:

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

SERVER-SENT EVENTS

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

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('destaq
ue')
```

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:

```
    <!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...’);setTi
meout(‘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:

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 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.

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

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

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.

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.

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:

```
<!DOCTYPE html>
<html lang=”pt-BR”>
```

```
    <head>
    <meta charset="UTF-8">
    <title>Zebra</title>
    <style>
      .zebraon{background:silver}
    </style>
    <script>
    window.onload=function(){
      var zebrar=document.querySelectorAll('.zebra tbody
tr')
      for(var i=0;i<zebrar.length;i+=2)
        zebrar[i].className='zebraon'
    }
    </script>
    </head>

    <body>

    <table class="zebra">
    <thead><tr>
    <th>Vendedor</th> <th>Total</th>
    </tr></thead>
    <tbody><tr>
    <td>Manoel</td> <td>12.300,00</td>
    </tr><tr>
    <td>Joaquim</td> <td>21.300,00</td>
    </tr><tr>
    <td>Maria</td> <td>13.200,00</td>
    </tr><tr>
    <td>Marta</td> <td>32.100,00</td>
    </tr><tr>
    <td>Antonio</td> <td>23.100,00</td>
    </tr><tr>
    <td>Pedro</td> <td>31.200,00</td>
    </tr></tbody>
    </table>

    </body>
    </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:

O objeto é buscado pelos atributos name ou id.

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

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:

- Um link, um elemento a com atributo href;
- Um botão, um elemento button;
- Um botão, um elemento input com o atributo type contendo button,

submit, reset ou image;
- Um radiobutton, um elemento input com o atributo type contendo radio;
- Um checkbox, um elemento input com o atributo type contendo checkbox;
- Um elemento select, contendo um ou mais options, define um grupo de comandos
- Um elemento qualquer com o atributo accesskey
- 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:

```
    <!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:

EXEMPLO DE MENU

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:

```
<!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">
            </menu>
            <menu label="Help">
                <command onclick="location='help.html'"
label="Help">
                <command onclick="location='about.html'"
label="About">
            </menu>
        </menu>

    </body>

    </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:

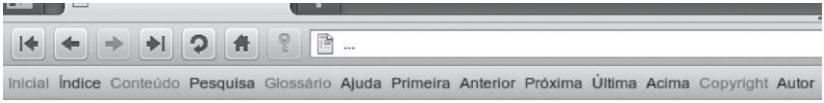
```
<link rel=”stylesheet” href=”estilo.css”>
```

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:



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

6 A NOVA GERAÇÃO DE APLICAÇÕES WEB II

MICRODATA

SEMÂNTICA ADICIONAL

Dê um olhada no seguinte código:

```
<!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:

```
    <!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:

```
<!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:

```
<!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:

```
    <!DOCTYPE html>
    <html>
    <head>
    <meta charset="UTF-8">
    <title>Visie Padrões Web</title>
    </head>
    <body>
    <address itemscope=" itemtype="http://data-
vocabulary.org/Organization"></address>
    <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>
    </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[3].

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

3 http://bit.ly/w9Mxn6

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:

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

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

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'
    }

    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:

Interface complexa: o código para armazenar Cookies envolve complexos cálculos com datas e controle do nome de domínio.

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:

- getItem(key): obtém um valor armazenado no Storage

- setItem(key,value) guarda um valor no Storage
- removeItem(key) exclui um valor do Storage
- clear() limpa o Storage

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 para que os agentes de usuário implementem corretamente esse recurso.

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:

```
localStorage.getItem(‘userChoice’)
```

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:

```
<!DOCTYPE HTML>
<html>
<head>
  <title>Clock</title>
  <script src="clock.js"></script>
  <link rel="stylesheet" href="clock.css">
</head>
<body>
 <p>The time is: <output id="clock"></output></p>
</body>
</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:

```
<!DOCTYPE HTML>
<html manifest="clock.manifest">
<head>
    <title>Clock</title>
    <script src="clock.js"></script>
    <link rel="stylesheet" href="clock.css">
</head>
<body>
    <p>The time is: <output id="clock"></output></p>
</body>
</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 documento 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		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

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

O usuário pode 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,erropo
```

```
s)
```

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:

Método	Descrição
length	O número de entradas no histórico
position	O número da entrada atual no histórico
add(data,title)	Adiciona uma entreada específica no histórico. Data pode ser um objeto literal com dados arbitrários. Title é como essa entreda 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')
```

7 CSS

O QUE É CSS?

O CSS formata a informação que é entregue pelo HTML. Essa informação pode ser qualquer coisa: imagem, texto, vídeo, áudio ou qualquer outro elemento criado. Grave isso: CSS formata a informação. Essa formatação na maioria das vezes é visual, mas não necessariamente. No CSS Aural, nós manipulamos o áudio entregue ao visitante pelo sistema lê a tela. Nós controlamos volume, profundidade do som, tipo da voz ou em qual caixa de som a voz sairá. De certa forma você está formatando a informação que está em formato de áudio e que o visitante está consumindo ao entrar no site. O CSS prepara essa informação para que ela seja consumida da melhor maneira possível.

Basicamente as atualizações do HTML5 foram a criação de novas tags, a mudança do significado de alguns elementos que foram modificados e outras tags que foram descontinuadas. As novidades interessantes ficaram para o pessoal que conhece Javascript. As APIs que o HTML5 disponibilizou são sem dúvida uma das features mais aguardadas por todos estes desenvolvedores. Diferentemente do CSS3 que trouxe mudanças drásticas para a manipulação e transição visual dos elementos do HTML.

Com o CSS que nós conhecemos podemos formatar algumas características básicas: cores, background, características de font, margins, paddings, posição e controlamos de uma maneira muito artesanal e simples a estrutura do site

com a propriedade float.

Você deve pensar: “mas com todas as características nós conseguimos fazer sites fantásticos, com design atraente e com a manutenção relativamente simples”. E eu concordo com você, mas outras características que nós não temos controles e dependemos de:

1) Algum programa visual como o Photoshop para criarmos detalhes de layout;
2) Javascript para tratarmos comportamentos ou manipularmos elementos específicos na estrutura do HTML;
3) Estrutura e controle dos elementos para melhorarmos acessibilidade e diversos aspectos do SEO;

Com as atualizações do CSS3 e com os browsers atualizando o suporte do CSS2.1, nós entramos em um patamar onde sem dúvida o CSS é a arma mais poderosa para o designer web. Segue uma pequena lista dos principais pontos que podemos controlar agora:

4) selecionar primeiro e último elemento;
5) selecionar elementos pares ou ímpares;
6) selecionarmos elementos específicos de um determinado grupo de elementos;
7) gradiente em textos e elementos;
8) bordas arredondadas;
9) sombras em texto e elementos;
10) manipulação de opacidade;
11) controle de rotação;
12) controle de perspectiva;
13) animação;
14) controle básico de 3D
15) estruturação independente da posição no código HTML;

Vamos passar basicamente por algumas novidades do CSS e entender como essas novidades podem nos ajudar a fazer websites mais dinâmicos e

interessantes.

O QUE é UM SELETOR?

Um seletor representa uma estrutura. Essa estrutura é usada como uma condição para determinar quais elementos de um grupo de elementos serão formatados.

Seletores encadeados e seletores agrupados são a base do CSS. Você aprende naturalmente durante o uso do dia a dia. Para você lembrar o que são seletores encadeados e agrupados segue um exemplo abaixo:

Exemplo de seletor encadeado:

```
div p strong a {
 color: red;
}
```

Este seletor formata o link (a), que está dentro de um strong, que está dentro de P e que por sua vez está dentro de um DIV.

Exemplo de seletor agrupado:

```
strong, em, span {
   color: red;
}
```

Você agrupa elementos separados por vírgula para que herdem a mesma formatação.

Estes seletores são para cobrir suas necessidades básicas de formatação de elementos. Eles fazem o simples. O que vai fazer você trabalhar menos, escrever menos código CSS e também menos código Javascript são os seletores complexos.

SELETORES COMPLEXOS

A sintaxe do CSS é simples:

```
seletor {
     propriedade: valor;
}
```

A propriedade é a característica que você deseja modificar no elemento. O valor é o valor referente a esta característica. Se você quer modificar a cor do texto, o valor é um Hexadecimal, RGBA ou até mesmo o nome da cor por extenso. Até aqui, nada muito diferente. Muitas vezes você não precisa aprender do que se trata a propriedade, basta saber que existe e se quiser decorar, decore. Propriedades são criadas todos os dias e não é um ato de heroísmo você saber todas as propriedades do CSS e seus respectivos valores.

Os seletores são a alma do CSS e você precisa dominá-los. É com os seletores que você irá escolher um determinado elemento dentro todos os outros elementos do DOM para formatá-lo. Boa parte da inteligência do CSS está em saber a utilizar os seletores de uma maneira eficaz, escalável e inteligente.

Os seletores complexos selecionam elementos que talvez você precisaria fazer algum script em Javascript para poder marcá-lo com uma CLASS ou um ID para então você formatá-lo. Com os seletores complexos você consegue formatar elementos que antes eram inalcansáveis.

EXEMPLO DE FUNCIONAMENTO

Imagine que você tenha um título (h1) seguido de um parágrafo (p). Você precisa selecionar todos os parágrafos que vem depois de um título H1. Com os seletores complexos você fará assim:

```
h1 + p {
   color:red;
```

```
}
```

Esse seletor é um dos mais simples e mais úteis que já utilizei em projetos por aí. Não precisei adicionar uma classe com JQuery, não precisei manipular o CMS para marcar esse parágrafo, não precisei fazer nenhum milagre para encontrar os parágrafos que vem logo depois de um H1. Apenas apliquei o seletor CSS e pronto.

Abaixo, veja uma lista de seletores complexos e quais as suas funções. Não colocarei todas as possibilidades aqui porque podem haver modificações e novos formatos. Para ter uma lista sempre atualizada, siga o link no final da tabela:

PADRÃO	SIGNIFICADO	CSS
elemento[atr]	Elemento com um atributo específico.	2
elemento[atr=”x”]	Elemento que tenha um atributo com um valor específico igual a “x”.	2
elemento[atr~=”x”]	Elemento com um atributo cujo valor é uma lista separada por espaços, sendo que um dos valores é “x”.	2
elemento[atr^=”x”]	Elemento com um atributo cujo valor comece exatamente com string “x”.	3
elemento[atr$=”x”]	Elemento com um atributo cujo valor termina exatamente com string “x”.	3
elemento[atr*=”x”]	Elemento com um atributo cujo valor contenha a string “x”.	3
elemento[atr|=”en”]	Um elemento que tem o atributo específico com o valor que é separado por hífen começando com EN (da esquerda para direita).	2
elemento:root	Elemento root do documento. Normalmente o HTML.	3
elemento:nth-child(n)	Selecione um objeto N de um determinado elemento.	3

elemento:nth-last-child(n)	Seleciona um objeto N começando pelo último objeto do elemento.	3
elemento:empty	Seleciona um elemento vazio, sem filhos. Incluindo elementos de texto.	3
elemento:enabled		
elemento:disabled	Seleciona um elemento de interface que esteja habilitado ou desabilidade, como selects, checkbox, radio button etc	3
elemento:checked	Seleciona elementos que estão checados, como radio buttons e checkboxes.	3
E > F	Seleciona os elementos E que são filhos diretos de F.	2
E + F	Seleciona um elemento F que precede imediatamente o elemento E.	2

Lista atualizada pelo W3C http://www.w3.org/TR/css3-selectors/#selectors

PSEUDO-CLASSES E PSEUDO-ELEMENTOS

Há uma diferença muito sensível entre pseudo-classes e pseudo-elementos. Ambos ajudam o desenvolvedor a encontrar e separar elementos específicos na árvore de objetos do documento HTML. Normalmente temos algumas necessidades ao selecionar determinados elementos que não tem identificação de class ou id no código. A situação se agrava quando não podemos modificar o HTML ou quando o código é gerado dinâmicamente.

Para entender a diferença é simples: lembre-se que a pseudo-classe é como se você inserisse uma classe dinâmica no elemento. E os pseudo-elementos é como se você inserisse um elemento em uma determinada posição no documento.

Exemplo de pseudo-classe: imagine você queira formatar o LINK quando o usuário passar o mouse sobre ele. Você pode manipulá-lo pela pseudo-classe :hover. É como se o browser colocasse no elemento uma classe no momento que o usuário passasse o mouse no elemento e você pode formatar essa classe pelo CSS.

Exemplo de pseudo-elemento: imagine que você queira formatar a primeira letra do parágrafo. Você teria que envolver a primeira letra da primeira palavra do parágrafo com algum elemento - o SPAN serviria - inserir uma classe neste elemento e então formatá-lo como o desejado. Sem contar que você teria que fazer um script - provavelmente em JQuery - para encontrar a primeira letra para poder inserir este elemento.

Com a utilização do pseudo-elemento ::first-letter o browser já encontra este elemento para você, possibilitando a formatação.

Nota: Antigamente, os pseudo-elementos eram escritos como as pseudo-classes, iniciando-os com dois pontos (:). Mas isso causava muita confusão entre os desenvolvedores para distinguir pseudo-elementos de pseudo-classes, por este motivo a especificação foi modificada, fazendo com que os pseudo-elementos sejam escritos iniciando com um par de dois pontos (::), assim os desenvolvedores conseguem diferenciar um pseudo-elemento de uma pseudo-classe.

PSEUDO-CLASSES

As pseudo-classes manipulam um determinado estado do elemento. Todos os elementos tem estados adicionais além daquele que você consegue ver. Por exemplo: ao passar o mouse, ao clicar, quando ganha foco e etc. Estes estados podem ser formatados para trazer melhor experiência para o usuário. A formatação destes estados ganha muita importância em mobiles e tablets, já que a interação normalmente é feita com os dedos, fazendo com que o usuário

tenha um feedback sensitivo maior.

Lembrando que a especificação das pseudo-classes vem da versão 2.1 do CSS. Contudo com as atualização dos browsers juntamente com a moda do CSS3, essas pseudo-classes e também grande parte dos pseudo-elementos foram implementados e estão prontos para usar.

Uma lista das pseudo-classes completa, juntamente com usa explicação mais abrangente e detalista pode ser encontrada na própria especificação do W3C.

As pseudo-classes são divididas em 2 tipos: estruturais e dinâmicas.

Pseudo-classes dinâmicas

As pseudo-classes dinâmicas controlam os estados dos elementos. Todos os elementos tem estados definidos. É fácil de entender quando lembramos do elemento A (link). Ele tem vários estados: quando você passa o mouse, quando você tira o mouse, quando você clica e quando é visitado. Estes estados o LINK podem ser formatados. Por isso que você modifica a cor ou coloca sublinhado quando o usuário passa o mouse por cima do link. Estes estados também estão presentes em outros elementos. Claro, um DIV por exemplo, não terá a pseudo-classe de VISITED, já que você não visita um DIV.

Pseudo-Classe	Descrição de uso
:hover	Quando o mouse passa por cima do objeto
:visited	Quando o elemento foi visitado. Normalmente aplicado para link.
:active	Quando o elemento está sendo ativo. Para ver esse estado, clique em um link, mas não solte o botão.

[4] Especificação W3C sobre Pseudo-Classes: http://www.w3.org/TR/css3-selectors/#pseudo-classes

:focus	Quando o elemento ganha foco. Por exemplo quando os inputs ganham foco para serem preenchidos.

Pseudo-classes estruturais

As pseudo-classes estruturais encontram elementos em uma árvore de elementos. Eles não formatam os estados, mas nos ajudam a encontrar elementos específicos para que sejam formatados sem a ajuda de programação ou inserção de alguma identificação direta no código.

Lembra-se do tempo que para criarmos uma tabela zebrada (aquela tabela que tem as cores de linhas alternadas) precisávamos fazer um loop em Javascript ou alguma linguagem server-side para contar as linhas e descobrir qual era ímpar e par, para então inserirmos uma classe nas linhas determinadas e só depois podermos formatá-las com CSS para que a tabela ficasse da forma desejada. Muito trabalho para ter um resultado ínfimo. Hoje podemos encontrar estes elementos facilmente utilizando a pseudo-classe :nth-child(). Simples assim:

```
table tr:nth-child(odd) {background: gray;}
```

O seletor acima encontra todos as TR (linhas) ímpares dentro da tabela e aplica um background cinza. Simples, não é? Então lembre-se: pseudo-classes selecionam elementos específicos na árvore de elementos do HTML.

Pseudo-classes do CSS3

O CSS 3 trouxe uma série de pseudo-classes, que expandem o poder do desenvolvedor ao escolher e formatar um determinado elemento que não tenha uma identificação definida direto no código. Abaixo segue uma lista de pseudo-classes que o CSS3 adicionou à cartilha:

Pseudo-classe	Descrição
:target	Encontra um elemento alvo. Muito utilizado para fazer abas.
:root	Encontra o elemento que é o elemento raiz do documento.
:empty	Encontra elementos que não tem filhos.
:enabled	Encontra elementos que estão habilitados. Por exemplo, inputs de checkbox, texto e etc.
:disabled	Encontra elementos desabilitados para edição ou manipulação.
:checked	Encontra elementos como checkbox e botões radio que estão checados.
:first-of-type	Encontra um elemento específico que seja o primeiro filho de um definido.
:last-of-type	Encontra um elemento específico que seja o último filho de um elemento definido.
:nth-child()	Encontra um elemento tomando como base sua posição na árvore de elementos do mesmo gênero.
:nth-last-child()	Encontra o último elemento tomando como base sua posição na árvore de elementos do mesmo gênero.
:nth-of-type()	Encontra os elementos do mesmo tipo elemento tomando como base sua posição na árvore de elementos do pai.
:last-child	Encontra o último elemento.
:first-child	Encontra o primeiro elemento.
:only-of-type	Seleciona os elementos de um mesmo tipo que são os filhos.
:only-child	Encontra os elementos filhos do mesmo gênero de um determinado pai.

:not()	Seleciona os elementos definidos, MENOS os elementos específicados pela pseudo-classe.

PSEUDO-ELEMENTOS

Pseudo-elementos permitem que os autores selecionem informações inacessíveis e possibilitem os desenvolvedores um caminho para referenciar conteúdos que não existem. Por exemplo: podemos inserir um elemento antes (::before) ou depois (::after) de um objeto já existente no HTML.

Existem alguns pseudo-elementos básicos que você já deve ter usado algumas vezes em seus projetos:

Pseudo-elemento	Descrição
::first-letter	Encontra a primeira letra de da primeira palavra de um determinado bloco de texto.
::first-line	Seleciona a primeira linha de um bloco de texto.
::before	Insere um elemento no início do conteúdo de um elemento.
::after	Insere um elemento no final do conteúdo de um elemento.

Para entendermos melhor como os pseudo-elementos trabalham, vamos tomar como exemplo o pseudo-elemento ::first-letter. Veja o código abaixo:

```
p::first-letter {font-size:20px; font-weight: bold; }
```

O código acima define que a primeira letra do parágrafo ficará com o tamanho de 20px e em negrito.

Para entender melhor como o browser trabalha: imagina que o browser, ao encontrar a primeira letra, ele a insere em um elemento SPAN com a classe FIRST-LETTER. Como no código abaixo:

```
    <p>
         <span class=”first-letter”>V</span>eja, mas não
toque.
    </p>
```

Obviamente este é um exemplo. O browser não insere nenhum elemento no seu código. Mas é como se fizesse isso. Imagine este mesmo exemplo quando ler sobre os outros pseudo-elementos.

Infelizmente os pseudo-elementos não sofreram tantas modificações e inserções de novas features com a vinda do CSS3.

GRADIENTE

Todos os browsers mais novos como Safari, Opera, Firefox e Chrome já aceitam essa propriedade e você pode utilizá-la hoje. Infelizmente, mas você já sabia, os IEs atuais (8 e 9) não reconhecem ainda. Mesmo assim, até o dia que este capítulo estava sendo escrito, o suporte dos browsers ainda estava muito nebuloso. Como você verá abaixo, os códigos que cada browser suporta ainda eram diferentes e não seguiam um padrão. Bem como não havia padrão específico para criar gradientes no estilo radial.

Você pode perguntar: “Mas já que terei o trabalho de produzir a imagem do gradiente para browsers antigos, porque não utilizar imagens para todos os browsers?“ Lembre-se que se utilizar uma imagem, o browser fará uma requisição no servidor buscando essa imagem. Sem imagem, teremos uma requisição a menos, logo o site fica um pouquinho mais rápido. Multiplique isso para todas as imagens de gradiente que você fizer e tudo realmente fará mais sentido. Deixe para que os browsers não adeptos baixem imagens e façam mais requisições.

Veja abaixo um exemplo de código, juntamente com o fallback de imagem caso o browser não suporte ainda essa especificação:

```
    div {
        width:200px;
        height:200px;
        background-color: #FFF;

        /* imagem caso o browser não aceite a feature */
        background-image: url(images/gradiente.png);

        /* Firefox 3.6+ */
        background-image: -moz-linear-gradient(green,
red);

        /* Safari 5.1+, Chrome 10+ */
        background-image: -webkit-linear-gradient(green,
red);
        /* Opera 11.10+ */
        background-image: -o-linear-gradient(green,
red);
    }
```

Atenção: Até que os browsers implementem de vez essa feature, iremos utilizar seus prefixos. Como ficou:

"Stops" ou definindo o tamanho do seu gradiente
O padrão é que o gradiente ocupe 100% do elemento como vimos no exemplo anterior, mas muitas vezes queremos fazer apenas um detalhe.

Nesse caso nós temos que definir um STOP, para que o browser saiba onde uma cor deve terminar para começar a outra. Perceba o 20% ao lado da cor

vermelha. O browser calcula quanto é 20% da altura (ou largura dependendo do caso) do elemento, e começa o gradiente da cor exatamente ali. O código de exemplo segue abaixo:

```
       /* Firefox 3.6+ */
        background-image: -moz-linear-gradient(green,
red 20%);

        /* Safari 5.1+, Chrome 10+ */
        background-image: -webkit-linear-gradient(green,
red 20%);

        /* Opera 11.10+ */
        background-image: -o-linear-gradient(green, red
20%);
```

Veja o resultado:

Se colocarmos um valor para o verde, nós iremos conseguir efeitos que antes só conseguiríamos no Illustrator ou no Photoshop. Segue o código e o resultado logo após:

```
       /* Firefox 3.6+ */
        background-image: -moz-linear-gradient(green
10%, red 20%);

        /* Safari 5.1+, Chrome 10+ */
        background-image: -webkit-linear-gradient(green
10%, red 20%);

        /* Opera 11.10+ */
        background-image: -o-linear-gradient(green 10%,
red 20%);
```

Perceba que o tamanho da transição vai ficando menor a medida que vamos aumentando as porcentagens. Muito, mas muito útil.

PROPRIEDADE BORDER-IMAGE

Definir imagem para as bordas é uma daquelas propriedades da CSS que você se pergunta como vivíamos antes de conhece-la. É muito mais fácil entender testando na prática, por isso sugiro que se você estiver perto de um computador, faça testes enquanto lê este texto. A explicação pode não ser suficiente em algumas partes, mas a prática irá ajudá-lo a entender.

Esta propriedade ainda está em fase de testes pelos browsers, por isso utilizaremos os prefixos para ver os resultados.Utilizarei apenas o prefixo do Safari, mas o Firefox já entende essa propriedade muito bem.

A sintaxe do border-image se divide em três partes: 1) URL da imagem que será utilizada. 2) Tamanho do slice das bordas. 3) Como o browser irá aplicar a imagem na borda.

Segue um exemplo da sintaxe abaixo:

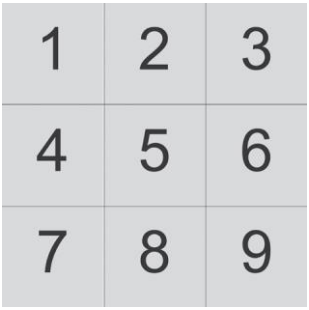
```
a {
   display:block;
   width:100px;
```

```
      -webkit-border-image: url(border.gif) 10 10 10 10
stretch;
    }
```

Acima definimos uma imagem com o nome de border.gif, logo depois definimos o width de cada uma das bordas do elemento. A sintaxe é igual a outras propriedades com 4 valores: top, right, bottom, left. E logo depois colocamos qual o tipo de tratamento que a imagem vai receber.

Dividindo a imagem

Para posicionar a imagem devidamente em seu objeto o browser divide a imagem em 9 seções:



Quando a imagem é colocada no elemento, o browser posiciona os cantos da imagem juntamente com os cantos correspondentes do elemento. Ou seja, o bloco 1 da imagem é colocado no canto superior esquerdo, o 3 no canto superior direito e assim por diante. Se você fizer o teste, a imagem aparecerá no elemento como se estivesse desproporcional. Isso é normal porque a imagem deve seguir as proporções do elemento e não as suas próprias.

Comportamento da imagem

O comportamento da imagem é a parte mais importante porque define como o centro da imagem (no caso do nosso exemplo a seção de número 5), irá ser tratada. Há vários valores, mas o que é mais simples de se entender é a stretch, que estica e escala a imagem para o tamanho do elemento aplicado. Há outros valores como round e repeat. Mas hoje alguns browsers não tem tanto suporte e acabam ignorando esses valores ou como no caso do Safari e o Chrome, interpretam o round como o repeat. Vamos nos concentrar com o stretch e você entenderá como funciona a propriedade.

Aplicação

Vamos utilizar a imagem abaixo para aplicar a propriedade. Iremos fazer um botão ao estilo iPhone. A coisa é simples e sugiro que você faça testes individualmente brincando com os valores das bordas e com diversas outras imagens para ver como funciona o recurso.

Irei aplicar o estilo em um link. O código segue abaixo:

```
a {
   display:block;
   width:100px;
   text-align:center;
```

```
        font:bold 13px verdana, arial, tahoma,
sans-serif;
        text-decoration:none;
        color:black;
    }
```

Para inserir a imagem, colocamos as duas linhas abaixo:

```
        border-width:10px;
        -webkit-border-image: url(button.png) 10 stretch;
```

Defini na primeira linha que a borda teria 10px de largura com a propriedade border-width. Na segunda linha define que a imagem utilizada seria a button. png, que as áreas da imagem teriam que se estender por 10px, e o valor stretch define que a imagem cobrirá o elemento inteiro.
Temos o border-width definindo a largura da borda, mas não temos nenhum valor dizendo quanto dessa largura a imagem deve tomar.

Os efeitos são bem estranhos quando esses valores estão mal formatados. Por isso, teste na prática essa propriedade para que não haja problemas a implementar em seus projetos. O pulo da gato, para mim, é a propriedade border-width.

SOMBRAS

Não tem segredo: A propriedade box-shadow produz drop shadow em um elemento e a propriedade text-shadow produz drop shadow diretamente nas letras do texto. A sintaxe das duas propriedades são idênticas. Veja abaixo:

```
    div {
       text-shadow:5px 10px 7px #000;
       box-shadow:5px 10px 7px #000;
    }
```

A sintaxe é divida em 3 partes. Os dois primeiros números são as coordenadas: significa que a sombra será deslocada 5px pra esquerda e 10px do topo.

O terceiro número é aplicado pra produzir o blur da sombra. Isso quer dizer que a sombra será esmaecida e ficará "diluída" em 7px.

O último valor é referente à cor. Você pode utilizar RGBA em vez de Hexadecimal. Aliás, o valor em RGBA pode ser usado em qualquer propriedade que dependa de cores.

Essa especificação ainda está iniciando, mas já funciona em alguns browsers sem a ajuda de prefixos, como no Safari. Mesmo assim, ainda sinto falta de algumas possibilidades, como por exemplo produzir apenas sombras em um dos lados do elemento. Atualmente a sombra é produzida em todos os lados, tornando difícil o controle do lado em que a sombra será aplicada. Entretanto a aplicação destas propriedades já salvam muito tempo na criação do layout.

Essas propriedades serão ignoradas em browsers que não as suportam. Logo, você tem uma decisão muito difícil: ou faz uma versão utilizando imagens PNGs para simular o efeito de sombra, ou, que é a opção que eu prefiro, não mostre sombras para os browsers que não suporta as propriedades. Você tem menos trabalho, mas haverá alguma diferença de visual nos browsers antigos.

As vezes, colocando uma borda no lugar da sombra já resolve o problema do visual e não temos o trabalho de criar um PNG somente para isso. Com a ajuda da biblioteca Modernizr fica fácil saber qual browser não aceita essas propriedades.

RGBA

Normalmente em web trabalhamos com cores na forma de hexadecimal. É a forma mais comum e mais utilizada desde os primórdios do desenvolvimento web. Mesmo assim, há outros formatos menos comuns que funcionam sem

problemas, um destes formatos é o RGB. O RGB são 3 conjuntos de números que começam no 0 e vão até 255 (0% até 100%), onde o primeiro bloco define a quantidade de vermelho (Red), o segundo bloco a quantidade de verde (Green) e o último bloco a quantidade de azul (Blue). A combinação destes números formam todas as cores que você pode imaginar.
z
No HTML o RGB pode ser usado em qualquer propriedade que tenha a necessidade de cor, como: color, background, border etc. Exemplo:

```
p {
        background:rgb(255,255,0);
        padding:10px;
        font:13px verdana;
}
```

Este código RGB define que o background o elemento P será amarelo.

RGBA E A DIFERENÇA DA PROPRIEDADE OPACITY

O CSS3 nos trouxe a possibilidade de modificar a opacidade dos elementos via propriedade opacity. Lembrando que quando modificamos a opacidade do elemento, tudo o que está contido nele também fica opaco e não apenas o background ou a cor dele. Veja o exemplo abaixo e compare as imagens:

A primeira é a imagem normal, sem a aplicação de opacidade:

Agora com o bloco branco, marcado com um P, com opacidade definida. Perceba que o background e também a cor do texto ficaram transparentes.

Isso é útil mas dificulta muito quando queremos que apenas a cor de fundo de um determinado elemento tenha a opacidade modificada. É aí que entra o RGBA. O RGBA funciona da mesma forma que o RGB, só que no caso do RGBA, além dos 3 canais RGB (Red, Green e Blue) há um quarto canal, A (Alpha) que controla a opacidade da cor. Nesse caso, podemos controlar a opacidade da cor do background sem afetar a opacidade dos outros elementos:

O funcionamento é assim:

Veja um exemplo aplicado abaixo:


```
p {
```

```
        background:rgba(255,255,0, 0.5);
        padding:10px;
        font:13px verdana;
}
```

O último valor é referente ao canal Alpha, onde 1 é totalmente visível e 0 é totalmente invisível. No exemplo acima está com uma opacidade de 50%.

Diferenças entre RGB e HSL

A manipulação de cores no HTML nunca foi muito flexível. No começo escolhíamos as cores escrevendo seus nomes por extenso como: black, blue, yellow, green, olive, maroon, fuchsia, red, white, silver, navy, teal, purple, lime, gray, aqua. Na década de 80 houve um acréscimo de 131 novas cores com nomes estranhos chamada X115. Estes nomes foram adotados pelos primeiros browsers tem sido suportados até hoje. Há nomes como mintcream, moccasin, navajowhite, powderblue, springgreen entre outros… O W3C ainda mantém uma lista completa dos nomes dessas cores.

Além de ser muito difícil de manter uma coleção de cores com seus nomes esquisitos, é quase impossível de criar um website tentando se adequar a quantidade limitada de cores disponíveis. Já tínhamos problemas suficientes com a quantidade limitada de fonts, imagine o trabalho que teríamos se os layouts fossem feitos apenas com um punhado e cores. Com esse problemas vários padrões matemáticos foram estabelecidos para a criação de qualquer cor. Existem dois padrões conhecidos chamados de RGB e HSL.

O formato RGB está ficando mais popular com o CSS3 por que agora podemos controlar o canal alpha utilizando o formato RGBA. Há também outro formato que ganhou alguma atenção do workgroup no W3C que é o formato de cor chamado HSL. Como o RGB, o HSL também ganhou um canal de opacidade, ficando HSLA. Muitos desenvolvedores ainda tem dúvidas sobre as diferenças entre RGBA e HSLA e qual utilizar.

[5] http://en.wikipedia.org/wiki/X11_color_names

[6] http://www.w3.org/TR/SVG/types.html#ColorKeywords

Tudo é muito simples de entender. RGB e HSL são dois formatos de composição de cores digitais. Você pode escolher qual dos dois utilizar, vai do seu gosto. Contudo como até hoje utilizamos o HEXADECIMAL como padrão de cores para web, minha sugestão é esperar para ver qual das duas especificações cairá no gosto do mercado para escolher um dos formatos para utilizá-la mais frenquentemente nos projetos. Ao que parece o RGB está ficando mais popular.

Entretanto os dois formatos tem flexibilidades diferentes e por isso pode ser muito difícil apenas uma delas se tornar mais popular que outra.

O RGB

O processo é simples: como na vida real onde você mistura cores para obter uma outra cor como resultado, você faz a mesma coisa com o RGB: você mistura as cores para obter uma outra cor como resultado. Para isso utilizaremos a soma de 3 valores: Red, Green e Blue: rgba(red, green, blue);

Veja a sintaxe abaixo para entender melhor a aplicação:

```
p {
   color: rgb(100%, 100%, 0%);
}
```

Os três valores são ligados às três principais respectivamente vermelha, verde e azul. No caso acima inseri 100% de cor para Vermelho e Verde. Como bom aluno de educação artística, você deve saber que misturando vermelho e verde a cor resultante será amarelo.

Você pode ser mais específico controlando até mesmo a fração da porcentagem, por exemplo:

```
p {
```

```
    color: rgb(55.2%, 100%, 0%);
}
```

Assim você consegue exatamente a cor que precisa. Esse formato de porcentagem e controle de fração também é aplicado ao HSL.
O RGB pode ser configurado utilizando valores hexadecimais. A conta não é simples e você precisa ser um pouco nerd para entender. A explicação é meio longa. Sugiro que você leia no Wikipedia algo mais detalista.

O HSL

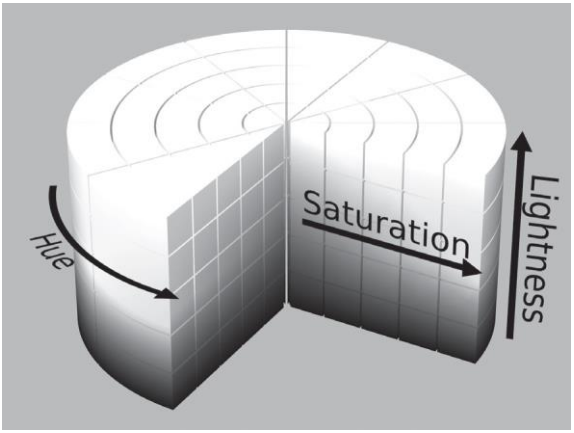
O HSL funciona um pouco diferente. A sintaxe é como abaixo:

```
p {
    color: hsl(0, 100%, 30%);
}
```

A escolha de cores no HSL não é baseado na mistura mas sim em um esquema baseado em um cilindro. O primeiro número de valor na sintaxe é onde escolhemos a cor. Começamos no topo com vermelho, onde o valor é 0, e damos uma volta de 360 graus, retornando novamente no topo, na cor vermelha. Conforme aumentamos o valor vamos selecionando as cores. Por exemplo, se selecionarmos um valor por volta de 120 obtemos um verde.

Veja a próxima imagem para entender melhor:

[7] http://en.wikipedia.org/wiki/RGB_color_model



Embora possamos escolher qualquer cor misturando as cores com o RGB, é muito mais instintivo escolhermos uma cor específica e modificarmos sua luminosidade. Escolhemos o azul e modificamos sua luminosidade para obtermos o tom que você deseja.

Para escolhermos a luminosidade e a saturação da cor modificamos os dois outros valores, o segundo valor é a luminosidade e o terceiro valor é a saturação ou a quantidade de cinza que você colocará na cor. Modificar a saturação é como se você mudasse a quantidade de cor. Quanto menos cor, mais cinza. Se você quiser uma cor mais suja, mais apagada, você diminui este valor. Caso contrário você a mantém como 100% e utiliza a quantidade integral da cor. Normalmente esse será o padrão.

E o hexadecimal?

O hexadecimal, queridinho dos nossos corações, sempre estará ao nosso lado. A sintaxe é muito mais curta que as outras duas especificações mas não temos todas as vantagens que as outras especificações nos dão, começando pelo canal alpha. Com a grande maioria dos programas visuais dando suporte ao formato hexadecimal ele ainda perdurará durante muito tempo.

CURRENTCOLOR

O valor currentColor é muito simples de se entender e pode ser muito útil para diminuirmos o retrabalho em alguns momentos da produção. Imagine que o currentColor é uma variável cujo seu valor é definido pelo valor da propriedade color do seletor. Veja o código abaixo:

```
p {
	background:red;
	padding:10px;
	font:13px verdana;
	color: green;
	border:1px solid green;
}
```

Note que o valor da propriedade color é igual ao valor da cor da propriedade border.

Há uma redundância de código, que nesse caso é irrelevante, mas quando falamos de dezenas de arquivos de CSS modulados, com centenas de linhas cada, essa redundância pode incomodar a produtividade. A função do currentColor é simples: ele copia para outras propriedades do mesmo seletor o valor da propriedade color. Veja o código abaixo para entender melhor:

```
p {
	background:red;
	padding:10px;
	font:13px verdana;
	color: green;
	border:1px solid currentColor;
}
```

Veja que apliquei o valor currentColor onde deveria estar o valor de cor da propriedade border. Agora, toda vez que o valor da propriedade color for modificado, o currentColor aplicará o mesmo valor para a propriedade onde ela está sendo aplicada.

Isso funciona em qualquer propriedade que faça utilização de cor como background, border, etc. Obviamente não funciona para a propriedade color. O currentColor também não funciona em seletores separados, ou seja, você não consegue atrelar o valor da propriedade color ao currentColor de outro seletor.

@FONT-FACE

A regra @font-face é utilizada para que você utilize fonts fora do padrão do sistema em seus sites. Para que isso funcione nós disponibilizamos as fonts necessárias em seu servidor e linkamos estas fonts no arquivo CSS. A sintaxe é bem simples e tem suporte a todos os navegadores, com algumas ressalvas.

```
    @font-face {
         font-family: helveticaneue;
         src: url(helveticaNeueLTStd-UltLt.otf);
    }
```

Na primeira linha você customiza o nome da font que você usará durante todo o projeto. Na segunda linha definimos a URL onde o browser procurará o arquivo da font para baixar e aplicar no site. Você aplica a fonte como abaixo:

```
    p {font:36px helveticaneue, Arial, Tahoma,
Sans-serif;}
```

Suponha que o usuário tenha a font instalada, logo ele não precisa baixar a font, assim podemos indicar para que o browser possa utilizar o arquivo da própria máquina do usuário. Menos requisições, mais velocidade. Veja o código abaixo:

```
    @font-face {
      font-family: helveticaneue;
      src: local(HelveticaNeueLTStd-UltLt.otf),
url(HelveticaNeueLTStd-UltLt.otf);
    }
```

Abaixo segue uma série de formatos que podem ser usados e que os browsers podem adotar:

String	Font Format	Common Extensions
truetype	TrueType	.ttf
opentype	OpenType	.ttf, .otf
truetype-aat	TrueType with Apple Advanced Typography extensions	.ttf
embedded-opentype	Embedded OpenType	.eot
woff	WOFF (Web Open Font Format)	.woff
svg	SVG Font	.svg, .svgz

Compatibilidade

As versões 7, 8 e 9 do Internet Explorer aceitam o @font-face apenas se a font for EOT. Você pode encontrar qualquer conversor online e esse problema é resolvido. Você pode converter suas fonts para EOT diretamente no Font Squirrel. O Safari, Firefox, Chrome e Opera aceitam fonts em TTF e OTF.

Para suprir a necessidade de atenção do Internet Explorer, você precisa especificar a URL da font .eot para que o Internet Explorer possa aplicar a font corretamente. A sintaxe fica desta forma:

```
@font-face {
     font-family: helveticaneue;
     src: url(helveticaNeueLTStd-UltLt.eot);
     src: url(helveticaNeueLTStd-UltLt.otf);
}
```

Criando um pacote Font-Face

O Font Squirrel fez um pequeno favor para toda a comunidade. É um sisteminha que converte suas fonts para os formatos necessários e te devolve para você utilizar em seus sites: http://migre.me/4qST9

MÚLTIPLOS BACKGROUNDS

Quem nunca teve que criar um background onde havia um gradiente em cima, embaixo e dos lados? Você sabe que para criar algo parecido você não pode utilizar uma imagem só. A solução até hoje seria criar 4 elementos divs aninhados (ou seja, um dentro do outro) e aplicar um pedaço deste background em cada destes elementos para dar a sensação de um background único. O resultado é interessante mas o meio que isso é feito não é nada bonito. Você era obrigado a declarar 4 elementos "inúteis" no seu HTML apenas para compensar um efeito visual. A possibilidade de atribuirmos múltiplos backgrounds em apenas um elemento é a feature que vai ajudá-lo a não sujar seu código.

A sintaxe para múltiplos backgrounds é praticamente idêntica a sintaxe para definir um background. Segue abaixo um exemplo:

```
div {
   width:600px;
   height:500px;
   background:
   url(img1.png) top left repeat-x,
   url(img2.png) bottom left repeat-y;
}
```

Definimos apenas uma propriedade background, o valor dessa propriedade em vez de conter apenas um valor como normalmente fazemos, poderá haver

vários, com suas respectivas definições de posição, repeat e attachment (fixed).

COLUMNS

Com a propriedade columns definimos colunas de texto de forma automática. Até hoje não havia maneira de fazer isso de maneira inteligente com CSS e o grupo de propriedades columns pode fazer isso de maneira livre de gambiarras.

COLUMN-COUNT

A propriedade column-count define a quantidade de colunas terá o bloco de textos.

```
     /* Define a quantidade de colunas, a largura é
definida uniformimente. */
    -moz-column-count: 2;
    -webkit-column-count: 2;
```

COLUMN-WIDTH

Com a propriedade column-width definimos a largura destas colunas.

```
    -moz-column-width: 400px;
    -webkit-column-width: 400px;
```

Fiz alguns testes aqui e há uma diferença entre o Firefox 3.5 e o Safari 4 (Public Beta).
O column-width define a largura mínima das colunas. Na documentação do W3C é a seguinte: imagine que você tenha uma área disponível para as colunas de 100px. Ou seja, você tem um div com 100px de largura (width).

E você define que as larguras destas colunas (column-width) sejam de 45px. Logo, haverá 10px de sobra, e as colunas irão automaticamente ter 50px de largura, preenchendo este espaço disponível. É esse o comportamento que o Firefox adota.

Já no Safari acontece o seguinte: se você define um column-width, as colunas obedecem esse valor e não preenchem o espaço disponível, como acontece na explicação do W3C e como acontece também no Firefox. Dessa forma faz mais sentido para mim.

Como a propriedade não está 100% aprovada ainda, há tempo para que isso seja modificado novamente. Talvez a mudança da nomenclatura da classe para column-min-width ou algo parecida venha a calhar, assim, ficamos com os dois resultados citados, que são bem úteis para nós de qualquer maneira.

COLUMN-GAP

A propriedade column-gap cria um espaço entre as colunas, um gap.

```
 /* Define o espaço entre as colunas. */
-moz-column-gap: 50px;
-webkit-column-gap: 50px;
```

Utilizamos aqui os prefixos -moz- e -webkit-, estas propriedades não funcionam oficialmente em nenhum browser. Mas já podem ser usados em browsers como Firefox e Safari.

TRANSFORM 2D

A propriedade transform manipula a forma com que o elemento aparecerá na tela. Você poderá manipular sua perspectiva, escala e ângulos. Uma transformação é especificada utilizando a propriedade transform. Os browsers

que suportam essa propriedade, a suportam com o prefixo especificado. Os valores possíveis até agora estão abaixo:

scale

O valor scale modificará a dimensão do elemento. Ele aumentará proporcionalmente o tamanho do elemento levando em consideração o tamanho original do elemento.

skew

Skew modificará os angulos dos elementos. Você pode modificar os ângulos individualmente dos eixos X e Y como no código abaixo:

```
        -webkit-transform: skewY(30deg);
        -webkit-transform: skewX(30deg);
```

translation

O translation moverá o elemento no eixo X e Y. O interessante é que você não precisa se preocupar com floats, positions, margins e etc. Se você aplica o translation, ele moverá o objeto e pronto.

rotate

O rotate rotaciona o elemento levando em consideração seu ângulo, especialmente quando o ângulo é personalisado com o transform-origin.

CSS Transform na prática

Veja o código abaixo e seu respectivo resultado:

```
    img {
        -webkit-transform: skew(30deg); /* para webkit
*/
        -moz-transform: skew(30deg); /* para gecko */
        -o-transform: skew(30deg); /* para opera */
        transform: skew(30deg); /* para browsers sem
```

```
prefixo */
     }
```

O código acima determina que o ângulo da imagem será de 30deg. Colocamos um exemplo para cada prefixo de browser. Ficando assim:

Várias transformações em um único elemento

Para utilizarmos vários valores ao mesmo tempo em um mesmo elemento, basta definir vários valores separando-os com espaços em uma mesma propriedade transform:

```
img {
      /* para webkit */
     -webkit-transform: scale(1.5) skew(30deg);

      /* para gecko */
     -moz-transform: scale(1.5) skew(30deg);
```

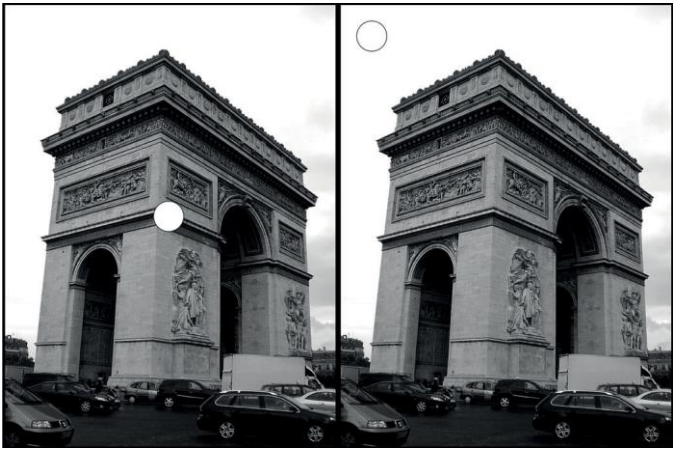
```
        /* para opera */
       -o-transform: scale(1.5) skew(30deg);

        /* para browsers sem prefixo */
        transform: scale(1.5) skew(30deg); }
```

transform-origin

A propriedade transform-origin define qual o ponto do elemento a transformação terá origem. A sintaxe é idêntica ao background-position. Observe o código abaixo:

```
    img {
        -webkit-transform-origin: 10px 10px; /* para
webkit */
        -moz-transform-origin: 10px 10px; /* para webkit
*/
        -o-transform-origin: 10px 10px; /* para webkit
*/
        transform-origin: 10px 10px; /* para webkit */
    }
```

Como padrão as transições sempre acontecem tendo como ponto de âncora o centro do objeto. Há algumas situações que pode ser que você queira modificar essa âncora, fazendo com que por exemplo, a rotação aconteça em algum dos cantos do elemento. O código de exemplo acima fará com que a transformação aconteça a partir dos 10px do topo no canto esquerdo. Veja a comparação entre o padrão e o resultado do código acima.



A propriedade transform fica mais interessante quando a utilizamos com a propriedade transition, onde podemos executar algumas animações básicas manipulando os valores de transformação.

INTRODUÇÃO AO CSS 3D

O CSS 3D é sem dúvida uma das features do CSS mais aguardadas por todas as crianças do Brasil. Fala a verdade! Fazer efeitozinhos com sombra, gradientes, transparências e etc já foi um dia na vida do desenvolvimento algo bacana. Hoje é muito fora de moda. Carne de vaca, sabe? Por isso o CSS 3D é tão esperado. Ele trará para a web efeitos visuais para layout que antes só viamos em sistemas que rodam em smartphones, tipo um iPhone ou nos sistemas operacionais mais populares como Linux e OSX.

Mas não se anime muito. Eu sei que você está ansioso para sair por aí colocando efeitos 3D de CSS em tudo quanto é aplicação. Mas calma… entenda que o CSS foi feito para estilizar documentos. Você o utiliza para melhorar a experiência dos usuários nos diversos dispositivos e não para enfeitar seu website como se

fosse uma penteadeira. Lembra-se dos websites cheios de gifs animados? Pois é, cuidado para não cair no mesmo erro. Você estará utilizando o CSS 3D da maneira certa se seus efeitos passarem desapercebidos pelo usuário ao utilizar seu sistema. Encher seu sistema com efeitos a cada clique ou a cada ação pode fazer com que o usuário perca tempo e a paciência.

Mas vamos ao que interessa.

O suporte

Infelizmente isso ainda está restrito para browsers. O Internet Explorer não tem suporte ainda e nem tem data para tal. Todos os exercícios que você ver neste post são feitos para browsers que tem WebKit como motor de renderização. Por isso teste em seu Chrome ou no seu Safari. Eu testei no Chrome porque o Safari mostrou algumas inconsistências. O Opera está esperando as especificações de CSS Transforms amadurecerem para adicionar este recurso. Testei no Firefox 8.0.1 e o exercício não funcionou.

A degradação do CSS 3D para os browsers que não o suportam é um pouco infeliz. Sugiro que se você for utilizar essas features, tente fazê-lo em projetos restritos para que não haja problemas com usuários de browsers antigos. Se ainda assim você quiser arriscar, crie soluções especifica para seu projeto, fazendo com que a experiência do seu cliente não seja muito prejudicada.
Sugiro que utilize a biblioteca Modernizr para identificar os browsers que não entendem o CSS 3D.

Tudo é uma questão de perspectiva

Para falar de 3D, precisamos falar sobre perspectiva. Para ativar uma área 3D o elemento precisará de perspectiva.

Você pode aplicar a perspectiva ao elemento de duas formas: utilizando diretamente a propriedade perspective ou adicionando um valor perspective()

na propriedade transform.

```
div {
  -webkit-perspective: 600;
}
```

Ou:

```
div {
  -webkit-transform: perspective(600);
}
```

Estes dois formatos são os gatilhos que ativam a área 3D onde o elemento irá trabalhar.

O valor da perspectiva determina a intensidade do efeito. Imagine como se fosse a distância de onde vemos o objeto. Quanto maior o valor, mais perto o elemento estará, logo, menos intenso será o visual 3D. Logo, se colocarmos um valor de 2000, o objeto não terá tantas mudanças visuais e o efeito 3D será suave. Se colocarmos 100, o efeito 3D será muito visível, como se fosse um inseto olhando um objeto gigante.

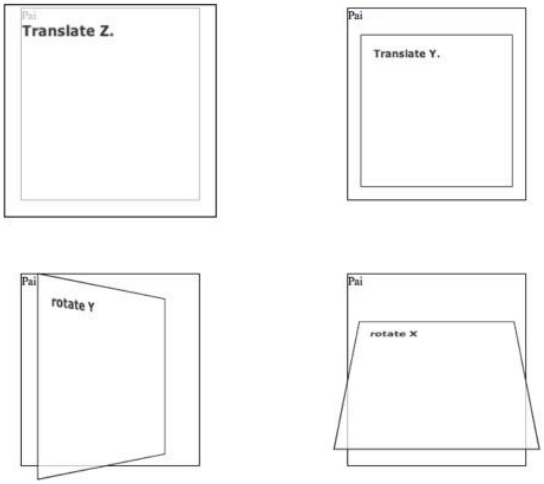
Você também precisa entender sobre o ponto de fuga. O ponto de fuga por padrão está posicionado no centro. Você pode modificar essa posição com a propriedade perspective-origin, que é muito parecido com a propriedade transform-origin, que define onde a ação de transformação do objeto acontecerá, nesse caso quando falamos de ações 2D. A propriedade perspective-origin afeta os eixos X e Y do elemento filho.

CSS 3D Transforms

Se você ainda não leu sobre CSS Transforms você pode ler algo aqui e ver em ação aqui. As propriedades são praticamente iguais, mas aplicadas para os princípios de 3D e não 2D.

Você deve estar acostumado a trabalhar com os eixos X e Y no CSS padrão. No CSS 3D podemos manipular também o eixo Z, que representa a profundidade. Veja um exemplo utilizando os valores rotateY, rotateX e translateZ. Perceba que no translateZ eu utilizei valores negativos e positivos. Quando utilizo o valor negativo, o objeto fica "mais longe", se coloco valores positivos, o objeto fica "mais perto".

Abaixo segue uma imagem do resultado do exemplo:



Nós podemos utilizar também alguns atalhos para estes valores onde podemos definir as três dimensões de uma vez:

translate3d(x,y,z)
scale3d(x,y,z)
rotate3d(x,y,z,angle)

Muito importante: ao utilizar as propriedades resumidas, os browsers ativam automaticamente a aceleração por hardware no Safari para que as animações tenham uma melhor performance.

FAzENDO O EFEITO DE CARD FLIP

O efeito de Card Flip é muito conhecido entre os usuários de iPhone.

A estrutura HTML é esta:

```
    <section class="geral">
      <div class="carta">
        <figure class="frente"><img src="card-front.
jpg"></figure>
        <figure class="atras"><img src="card-back.
jpg"></figure>
      </div>
    </section>
```

O elemento .geral é onde iniciaremos o ambiente 3D. O elemento .carta age como container dos objetos 3D. Cada face da carta está separada por um elemento figure, com uma imagem.

Para começar, precisamos aplicar a perspectiva para o elemento .geral iniciar o espaço 3D.

```
    .geral {
        width: 200px;
        height: 293px;
        position: relative;
        margin:10% auto 0;
        -webkit-perspective: 500;
    }
```

Defini uma largura e altura, coloquei um position: relative; para que os elementos dentro dele sejam posicionados se referenciando por ele. Coloquei uma margem só para separá-lo do topo do body a fim de conseguirmos ver melhor os efeitos.
Por fim, coloquei a propriedade -webkit-perspective: 500; para aplicarmos o efeito 3D. O valor de 500 faz uma boa perspectiva.

Agora definiremos as dimensões da carta e suas propriedades.

```
.carta {
  width: 100%;
  height: 100%;
  position: absolute;
  -webkit-transition: -webkit-transform 1s;
}
```

Largura e altura precisam ser de 100% para definir a área que o 3D irá aplicar. O position: absolute; é necessário para que as cartas fiquem relativas ao elemento .geral. A propriedade -webkit-transition: -webkit-transform 1s; define o tempo de transição do efeito, neste caso ele vai durar 1 segundo.

Formatando as cartas:

```
.carta figure {
  margin:0;
  display: block;
  position: absolute;
  width: 100%;
  height: 100%;
  -webkit-backface-visibility: hidden;
}
```

Vamos direto para a propriedade -webkit-backface-visibility: hidden; já que as outras dispensam comentários. Essa propriedade faz com que a face de trás da carta não apareça e nem se sobreponha no momento do efeito.

E finalmente, para fazer com que a parte de trás da carta apareça no verso correto, nós temos que rotacioná-la.

```
.atras{-webkit-transform: rotateY(180deg);}
```

E feito se completa com o trigger para fazer a animação acontecer. Nesse caso farei com um hover no elemento .carta, onde iremos rotacioná-lo em -180 graus.

```
.carta:hover {
  -webkit-transform: rotateY(-180deg);
}
```

E Voilá! Se quiser brincar um pouco, modifique a origem da transformação com a propriedade -webkit-transform-origin. Adicionando essa linha, a transformação acontece para a direita em vez de ser pelo centro, como é o padrão:

```
.carta:hover {
  -webkit-transform: rotateY(-180deg);
  -webkit-transform-origin: right center;
}
```

PROPRIEDADE TRANSITION

Durante muito tempo o CSS só serviu para pintar quadradinhos e mais nada. Desde quando o pessoal do WaSP organizou todo o movimento dos Padrões Web fazendo com que todos os desenvolvedores, fabricantes de browsers e até mesmo o W3C acreditassem no poder dos padrões não houve grandes atualizações no CSS. Praticamente formatávamos font, background, cor, tamanhos e medidas de distância e posição.

A propriedade transition, a regra keyframe e outras propriedades vieram vitaminar a função que o CSS tem perante o HTML acrescentando maneiras de produzirmos animações e transições. Não estou dizendo que você fará animações complicadas, com diversos detalhes técnicos e etc. Para esse tipo de resultado existem outras ferramentas, incluindo Canvas e SVG, que com certeza farão um trabalho melhor com menos esforço. Mas é fato que as animações via CSS nos ajudam a levar a experiência do usuário para outro patamar.

Lembrando que o nível de suporte de algumas dessas técnicas ainda é muito baixo. A propriedade transition funciona em boa parte dos browsers atuais. Mas a regra keyframe que nos permite controlar as fases de uma animação ainda é muito restrito. Para uma tabela mais atual e detalhada de suporte e compatibilidade, faça uma procura no Google ou procure no site Can I Use (http://caniuse.com/). Onde for possível demonstrarei o código com o prefixo dos browsers que suportam as propriedades.

Pequenas transIçÕES

A propriedade transition é praticamente auto explicativa. Sua sintaxe tão simples que talvez até dispense explicações mais elaboradas. Vamos começar com o código abaixo:

```
a {
    color: white;
    background: gray;
}
```

No código definimos que o link terá sua cor de texto igual a preta e seu background será cinza.

O resultado esperado é que ao passar o mouse no link a cor do texto seja modificada, mudando do branco para o preto e que a cor de background mude de cinza para vermelho. O código abaixo faz exatamente isso:

```
a {
    color: white;
    background: gray;
}

a:hover {
    color: black;
    background: red;
}
```

O problema é que a transição é muito brusca. O browser apenas modifica as características entre os dois blocos e pronto. Não há nenhuma transição suave entre os dois estados.

Podemos fazer a mudança de um estado para outro utilizando a propriedade transition. Suponha que ao passar o mouse, as mudanças acontecessem em um intervalo de meio segundo. Bastaria colocar a propriedade transition no a:hover e pronto. Ao passar o mouse, o browser modificaria as características do link com uma pequena transição de meio segundo. O código seria como se segue abaixo:

```
    a:hover {
        color: black;
        background: red;
        -webkit-transition: 0.5s; /* Com prefixo do
Safari */
        transition: 0.5s; /* Para browsers que suportam
a propriedade */
    }
```

Dessa forma a transição apenas acontece quando o hover é ativado. O problema é que ao tirar o mouse, o browser volta bruscamente para as características iniciais. Para modificar isso basta inserir também a propriedade transition no estado inicial.

```
    a {
        color: white;
        background: gray;
        -webkit-transition: 0.5s;
    }

    a:hover {
        color: black;
        background: red;
        -webkit-transition: 0.5s;
    }
```

O que a propriedade transition faz é comparar os valores das propriedades em comum entre os dois estados do link ou de qualquer outro elemento, assim ela modifica suavemente os valores quando há a ativação da função. Esta é uma técnica simples e que serve para manipularmos transições básicas como cor, tamanho, posição etc.

Agora suponha que em um bloco há uma determinada propriedade que no outro bloco não há, como no código abaixo:

```
a {
    border:1px solid orange;
    color: white;
    background: gray;
    -webkit-transition: 0.5s;
}

a:hover {
    color: black;
    background: red;
    -webkit-transition: 0.5s;
}
```

Nesse caso o browser detecta que há uma propriedade no primeiro estado, mas não no segundo, por isso ele não faz a transição desta propriedade, apenas das propriedades em comuns.

Abaixo veja o código. copie em um arquivo HTML e veja o efeito:

```
<!DOCTYPE html>

<html lang="pt-br">
<head>
  <meta charset="utf-8">
  <title>CSS Transition</title>
  <style type="text/css" media="screen">
  a {
        color:white;
```

```
        background:gray;
        -webkit-transition: 0.5s linear;
    }
    a:hover {
        color:black;
        background:red;
        -webkit-transition: 0.5s linear;
    }
    </style>
</head>
<body>

<a href="#">Link! Hello World!</a>

</body>
</html>
```

PROPRIEDADE ANIMATION E REGRA KEYFRAME

A propriedade trasition trabalha de forma muito simples. Você praticamente diz para o browser qual o valor inicial e o valor final para que ele aplique a transição automaticamente, controlamos praticamente apenas o tempo de execução. Para termos mais controle sobre a animação temos a propriedade animation que trabalha juntamente com a rule keyframe.

Basicamente você consegue controlar as características do objeto e suas diversas transformações definindo fases da animação. Observe o código abaixo e veja seu funcionamento:

```
@-webkit-keyframes rodar {
    from {
        -webkit-transform:rotate(0deg);
    }
    to {
        -webkit-transform:rotate(360deg);
```

```
    }
}
```

O código acima define um valor inicial e um valor final. Agora vamos aplicar esse código a um elemento. Minha ideia é fazer um DIV girar. ;-)

O código HTML até agora é este. Fiz apenas um div e defini este keyframe:

```
<!DOCTYPE html>

<html lang="pt-br">
<head>
   <title></title>
   <meta charset="utf-8">
   <style>
   @-webkit-keyframes rodaroda {
          from {
                 -webkit-transform:rotate(0deg);
          }
          to {
                 -webkit-transform:rotate(360deg);
          }
   }
   </style>
</head>
<body>

<div></div>

</body>
</html>
```

Primeiro você define a função de animação, no caso é o nosso código que define um valor inicial de 0 graus e um valor final de 360 graus. Agora vamos definir as características deste DIV.

```
div {
   width:50px;
   height:50px;
```

```
    margin:30% auto 0;
    background:black;
}
```

Nas primeiras linhas defini qual será o estilo do div. Ele terá uma largura e uma altura de 50px. A margin de 30% do topo garantirá um espaço entre o objeto e o topo do documento e background preto.

A propriedade animation tem uma série de propriedades que podem ser resumidas em um shortcode bem simples. Veja a tabela logo a seguir para entender o que cada propriedade signifca:

Propriedade	Definição
animation-name	Especificamos o nome da função de animação
animation-duration	Define a duração da animação. O valor é declarado em segundos.
animation-timing-function	Descreve qual será a progressão da animação a cada ciclo de duração. Existem uma série de valores possíveis e que pode ser que o seu navegador ainda não suporte, mas são eles: ease, linear, ease-in, ease-out, ease-in-out, cubic-bezier(<number>, <number>, <number>, <number>) [, ease, linear, ease-in, ease-out, ease-in-out, cubic-bezier(<number>, <number>, <number>, <number>)]*O valor padrão é ease.
animation-interation-count	Define o número de vezes que o ciclo deve acontecer. O padrão é um, ou seja, a animação acontece uma vez e pára. Pode ser também infinito definindo o valor infinite no valor.

animation-direction	Define se a animação irá acontecer ou não no sentido inverso em ciclos alternados. Ou seja, se a animação está acontecendo no sentido horário, ao acabar a animação, o browser faz a mesma animação no elemento, mas no sentido antihorário. Os valores são alternate ou normal.
animation-play-state	Define se a animação está acontecendo ou se está pausada. Você poderá por exemplo, via script, pausar a animação se ela estiver acontecendo. Os valores são running ou paused.
animation-delay	Define quando a animação irá começar. Ou seja, você define um tempo para que a animação inicie. O valor 0, significa que a animação começará imediatamente.

Voltando para o nosso código, vamos aplicar algumas dessas propriedades:

```
div {
   width:50px;
   height:50px;
   margin:30% auto 0;
   background:black;

   -webkit-animation-name: rodaroda;
   -webkit-animation-duration: 0.5s;
   -webkit-animation-timing-function: linear;
   -webkit-animation-iteration-count:  infinite;
   -webkit-animation-direction: alternate;

}
```

Veja que na propriedade animation-name chamamos o mesmo nome que demos na nossa função de keyframe logo no começo da explicação. Depois definimos uma duração de ciclo de meio segundo. Definimos que o comportamento da animação será linear, e com a propriedade

animation-iteration-count definimos que ele girará infinitamente. E por último definimos pelo animation-direction que a animação deverá ser alternada, ou seja, o DIV girará para um lado, e quando alcançar o final da animação, o browser deverá alternar essa animação.

Podemos melhorar esse código utilizando a versão shortcode, que é mais recomendado. Veja a ordem que devemos escrever as propriedades animation em forma de shortcode:

```
    animation: [<animation-name> || <animation-duration>
|| <animation-timing-function> || <animation-delay> ||
<animation-iteration-count> || <animation-direction>] [,
[<animation-name> || <animation-duration> || <animation-
timing-function> || <animation-delay> || <animation-
iteration-count> || <animation-direction>] ]*
```

Aplicando isso ao nosso código:

```
    div {
      width:50px;
      height:50px;
      margin:30% auto 0;
      background:black;

      -webkit-animation: rodaroda 0.5s linear infinite
alternate;
    }
```

Pronto. Agora temos um elemento que gira sem parar, ora para direita ora para esquerda.

Definindo ciclos

Nós definimos no keyframe do nosso último exemplo apenas um início e um fim. Mas e se quiséssemos que ao chegar na metade da animação o nosso elemento ficasse com o background vermelho? Ou que ele mudasse

de tamanho, posição e etc? É aí onde podemos flexibilizar melhor nosso keyframe definindo as fases da animação. Por exemplo, podemos dizer para o elemento ter uma cor de background diferente quando a animação chegar aos 10% do ciclo, e assim por diante.

Veja o exemplo:

```
@-webkit-keyframes rodaroda {
	0% {
		-webkit-transform:rotate(0deg);
	}
	50% {
		background:red;
		-webkit-transform:rotate(180deg);
	}
	100% {
		-webkit-transform:rotate(360deg);
	}
}
```

Definimos acima que o início da animação o elemento começará na posição normal, 0 graus.

Quando a animação chegar aos 50% do ciclo, o elemento deverá ter girado para 180 graus e o background dele deve ser vermelho. E quando a animação chegar a 100% o elemento deve ter girado ao todo 360 graus e o background, como não está sendo definido, volta ao valor padrão, no nosso caso black, que foi definido no CSS onde formatamos este DIV.

Logo nosso elemento girará pra sempre e ficará alternando a cor de fundo de preto para vermelho. Fiz um exemplo bem simples modificando apenas o background, mas você pode muito bem definir um position e modificar os valores de left e top para fazer o elemento se movimentar.
No exemplo também defini apenas 3 estágios (0%, 50% e 100%) você pode definir um maior número de estágios: 5%, 10%, 12%, 16% e etc... Adequando as fases da animação às suas necessidades.

Há exemplos muito interessantes na internet onde podemos ver todo o poder das animações feitas pela CSS. Veja o exemplo que fizemos aqui neste texto no endereço: http://migre.me/4ubym

MÓDULO TEMPLATE LAYOUT

Para mim a parte mais fácil de desenvolver um site com CSS é o planejamento e diagramação do layout. Coincidentemente é a parte que mais os desenvolvedores tem problemas de compatibilidade crossbrowser ou por falta de recursos mais avançados. Mas se você parar para pensar, apenas uma propriedade cuida dessa parte, que é a propriedade float. Até agora, para mim, o float era a propriedade mais importante que existia no CSS. O float cuidava de toda a diagramação do site, desde os elementos que definiam as áreas principais do site até os pequenos detalhes de imagens e ícones.

A propriedade float é muito simples de se entender. O problema não é o funcionamento, mas os efeitos que a propriedade float causa nos elementos próximos. Se você pede para duas colunas ficarem flutuando à esquerda e outra coluna à direita, o rodapé sobe. Ou se você coloca um elemento envolvendo outros elementos com float, esse elemento perde a altura. Estes são problemas corriqueiros que já tem soluções inteligentes e que não apresentam chateações mais graves.

Infelizmente o float não é o ideal para a diagramação e organização dos elementos do layout. Ele resolve muitos problemas, mas deixa a desejar em diversos sentidos. A principal desvantagem do float é que ele é completamente dependente da ordem dos elementos no HTML. Existem técnicas que você consegue fazer quase que qualquer oganização visual sem encostar no código HTML. Mas há outras necessidades que invariavelmente você precisará modificar a ordem dos elementos no meio do HTML para que a diagramação do site saia conforme o esperado. Essa organização do HTML pode alterar desde programação server-side e até resultados de SEO e acessibilidade. Por

isso é interessante que o HTML fique organizado de forma que ele supre as necessidades dessas bases. Sua organização visual deve ser independente desta organização. Quem nunca sentiu falta de alternar a posição dos elementos verticalmente?

Tendo em vista estes e outros problemas o W3C criou um novo módulo. Na verdade ele não é o único e nem pode ser para que tenhamos diversas formas de trabalhar. O módulo em questão é chamado de Template Layout. Esse módulo consiste em uma forma de criarmos e organizarmos os elementos e informações do layout de forma menos espartana e mais flexível.

Podemos dividir a construção do layout em duas grandes partes: 1) Definição dos elementos mestres e grid a ser seguido. 2) Formatação de font, cores, background, bordas etc.

As propriedades nesta especificação trabalham associando uma política de layout de um elemento. Essa política é chamada de template-based positioning (não tem nada a ver com a propriedade position, pelo contrário é uma alternativa) que dá ao elemento um grid invisível para alinhar seus elementos descendentes.

Porque o layout deve se adaptar em diferentes janelas e tamanhos de papéis, as colunas e linhas do grid deve ser fixas ou flexiveis dependendo do tamanho.

O W3C mostra alguns casos comuns:

- Páginas complexas com múltiplas barras de navegação em áreas com posições fixas como publicidade, submenus e etc.

- Formulários complexos, onde o alinhamento de labels e campos podem ser facilmente definidos com as propriedades deste módulo com a ajuda das propriedades de margin, padding e tables.

- GUIs, onde botões, toolbars, labels, ícones etc, tem alinhamentos complexos

que precisam estar sempre alinhados caso o tamanho ou a resolução da tela mudem.

- Mídias que são paginadas, como Mídias de impressão onde cada página são divididos em áreas fixas para conteúdos de diferentes gêneros.

Template-based positioning são uma alternativa para a propriedade position com o valor absolute. Antigamente lembro-me que quase todos os desenvolvedores tentavam organizar e diagramar layouts utilizando position. Não que seja errado, mas definitivamente não é a melhor forma. Costumo dizer em meus cursos e palestras que position é para detalhes. Nada muito grande, mas para pequenos detalhes. Usamos position para posicionarmos elementos que não tem relação direta com sua posição no código HTML. Ou seja, não importa onde o elemento esteja, o position:absolute; irá posicionar o elemento nas coordenadas que você quiser.

Sintaxe e funcionamento

O módulo Template Layout basicamente define slots de layout para que você encaixe e posicione seus elementos. O mapeamento dos slots é feito com duas propriedades que já conhecemos que este módulo adiciona mais alguns valores e funcionalidades, são as propriedades position e display.

A propriedade display define como será o Grid, ou seja, quantos espaços úteis terá o layout e a propriedade position irá posicionar seus elementos nestes slots.

Veja o código HTML abaixo:

```
<div class=”geral”>
  <nav class=”menu”>...</nav>
  <aside class=”menulateral”>...</aside>
  <aside class=”publicidade”>...</aside>
  <article class=”post”>...</article>
```

```
  <footer>...</footer>
</div>
```

Agora iremos definir a posição destes elementos. O código CSS seria assim:

```
.geral {
   display: “aaa”
            “bcd”
            “eee”;
}

nav.menu {position:a;}
aside.menulateral {position:b;}
aside.publicidade {position:d;}
article.post {position:c;}
footer {position:e;}
```

O FUNCIONAMENTO DA PROPRIEDADE DISPLAY

A propriedade display define a organização dos slots. Um slot é o local onde os elementos serão colocados.

Aqui o elemento display trabalha como um table, onde seu conteúdo será colocando em colunas e linhas. A única diferença é que o número de linhas e colunas não dependem do conteúdo. A outra principal diferença é que a ordem dos descendentes no código não importa. Ou seja, não importa a estrutura dos elementos no HTML, você pode colocá-los em qualquer lugar do layout.

Cada letra diferente é um slot de conteúdo diferente. O @ (arroba) define um sinal para um slot padrão. E o “.” (ponto) define um espaço em branco.

Quando repetimos as letras como no exemplo anterior, tanto na horizontal quanto na vertical, é formado um slot único que se expande para o tamanho da quantidade de slots. Lembra do colspan ou rowspan utilizados na tabela?

Pois é, funciona igualzinho.

Não é possível fazer um slot que não seja retangular ou vários slots com a mesma letra. Um template sem letra nenhuma também não é possível. Um template com mais de um @ também é proibido. Se houverem esses erros a declaração é ignorada pelo browser.

Para definir a altura da linha (row-height) podemos utilizar o valor padrão "auto", que define a altura da linha de acordo com a quantidade de conteúdo no slot. Você pode definir um valor fixo para a altura. Não é possível definir um valor negativo. Quando definimos um * (asterísco) para a altura, isso quer dizer que todas as alturas de linha serão iguais.

A largura da coluna (col-width) é definida com valores fixos, como o row-height. Podemos definir também o valor de * que funciona exatamente igual ao altura de linha, mas aplicados a largura da coluna. Há dois valores chamados max-content e min-content que fazem com que a largura seja determinada de acordo com o conteúdo. Outro valor é o minmax(p,q) que funciona assim: a largura das colunas são fixadas para ser maiores ou iguais a p e menores ou iguais a q. Pode haver um espaço branco (white space) em volta de p e q. Se q > p, então q é ignorado e o minmax(p,q) é tratado como minmax(p,p). O valor fit-content é o equivalente a minmax(min-content, max-content).

Definindo a largura e altura dos slots

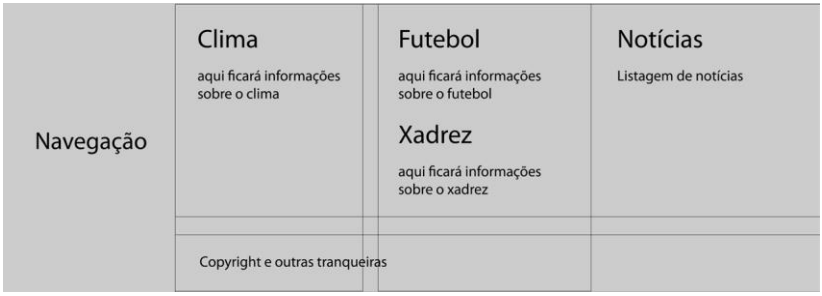
Para definir a altura dos slots, utilizamos uma sintaxe diretamente na propriedade display. Veja abaixo um exemplo de como podemos fazer isso:

```
display: "a a a" /150px
         "b c d"
         "e e e" / 150px
         100px 400px 100px;
```

Formatando de uma maneira que você enteda, fica assim:

```
display: “a      a      a” /150px
         “b      c      d”
         “e      e      e” /150px
         100px  400px  100px;
```

Ou seja, a primeira coluna do grid terá 100px de largura, a segunda 400px e a terceira 100px.
As medidas que coloquei ao lado, iniciando com uma / (barra) definem a altura das linhas. Logo a primeira linha terá 150px e a terceira linha também. A linha do meio, como não tem altura definida terá a altura de acordo com a quantidade de conteúdo.



O espaço entre as colunas são definidos com “.” (pontos). Veja o exemplo abaixo:

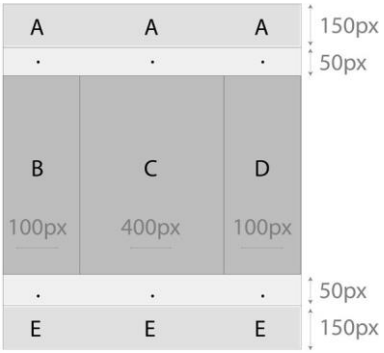
```
display: “a      a      a” /150px
         “.      .      .” /50px
         “b      c      d”
         “.      .      .” /50px
         “e      e      e” /150px
         100px  400px  100px;
```

No exemplo acima fiz duas colunas no código compostos por pontos em vez de fazer com letras. Isso quer dizer que entre as colunas do grid haverá um espaço em branco de 50px de altura.

O FUNCIONAMENTO DA PROPRIEDADE POSITION

O valor da propriedade position especifica qual linha e coluna o elemento será colocado no template. Você escreve apenas uma letra por elemento. Vários elementos podem ser colocados em um mesmo slot. Logo estes elementos terão o mesmo valor de position.

Abaixo veja uma imagem que pegamos diretamente do exemplo do W3C. O layout é muito simples:



Este layout é representado pelo código abaixo. Primeiro o HTML:

```
<ul id=”nav”>
     <li>navigation</li>
</ul>
<div id=”content”>
    <div class=”module news”>
         <h3>Weather</h3>
         <p>There will be weather</p>
    </div>
    <div class=”module sports”>
         <h3>Football</h3>
         <p>People like football.</p>
    </div>
    <div class=”module sports”>
         <h3>Chess</h3>
         <p>There was a brawl at the chess
```

```
tournament</p>
        </div>
        <div class=”module personal”>
             <h3>Your Horoscope</h3>
             <p>You’re going to die (eventually).</p>
        </div>
        <p id=”foot”>Copyright some folks</p>
    </div>
```

Agora veja o CSS com toda a mágica:

```
    body {
         display: “a     b”
                  10em   *;
    }
    #nav {
         position: a;
    }
    #content {
         position: b;
         display: “c   .   d   .   e “
                  “.   .   .   .   . “  /1em
                  “.   .   f   .   . “
                   *  1em  *  1em  *;
    }
    .module.news {position: c;}
    .module.sports {position: d;}
    .module.personal {position: e;}
    #foot {position: f;}
```

Lembre-se que não importa a posição dos elementos no código. E essa é a mágica. Podemos organizar o código HTML de acordo com nossas necessidades e levando em consideração SEO, Acessibilidade e processo de manutenção. O HTML fica totalmente intacto separado de qualquer formatação. Muito, mas muito interessante.

Pseudo-elemento ::slot()

Um slot é uma área do layout separada para colocarmos os elemetnos que escolhermos. Suponha que você queira que um determinado slot tenha um fundo diferente, alinhamento e etc... Essa formatação será aplicada diretamente no slot e não no elemento que você colocou lá. Fica mais simples de formatar porque você não atrela um estilo ao elemento e sim ao slot. Se você precisar posicionar aquele elemento em outro lugar, você consegue facilmente.

```
    body { display: “aaa”
                    “bcd” }
    body::slot(b) { background: #FF0 }
    body::slot(c), body::slot(d) { vertical-align:
bottom }
```

As propriedades aplicadas no pseudo elemento slot() seguem abaixo:
Todos as propriedades background.
vertical-align
overflow
box-shadow, block-flow e direction ainda estão sendo estudados pelo W3C se elas entrarão ou não.

Mas e o float?

É senhores... Eu acho que o float tem seus dias contados para a criação de estruturas de layouts. Quando utilizamos o float para estruturar o layout, nós dependemos profundamente da organização e posição dos elementos no código HTML. Não estou dizendo que o float ficará obsoleto, você ainda vai utilizá-lo e muito. Você vai pará-lo de utilizar para criar a base estrutural do site. Ou seja, a estrutura básica de divisão de conteúdo será feita pelo Template Layout, mas muitos dos detalhes internos e organização dos elementos contidos nos elementos mestres serão feitos com float.

Com o Template Layout a estrutura do layout não depende da posição dos

elementos do HTML no código, você poderá otimizar o código ao máximo para os leitores de tela e sistemas de busca, já que estes meios de acesso prezam pela estrutura do seu conteúdo.

Já funciona?

Sim, já funciona, mas não nativamente nos browser. Esta especificação ainda é um rascunho do W3C e por isso os browsers ainda não a suportam. Mesmo assim um desenvolvedor criou um script em Javascript que entende o CSS desta especificação e simula os resultados. Funciona muito bem.

PAGED MEDIA

Com certeza você já deve ter tentado ler um livro ou uma apostila em algum site na web e preferiu imprimir o texto para ler offiline, no papel por ser mais confortável ou por ser mais prático quando não se está conectado. Existem vários motivos para que um leitor queira imprimir o conteúdo de um site, principalmente sites com textos longos e pesados. Durante muito tempo o principal motivo era que ler na tela do computador era cansativo. Hoje isso ainda é um problema, mas com o avanço das telas e do aparecimento das tablets no mercado, você consegue passar mais tempo na frente de uma tela lendo grandes quantidades de texto. O problema é que geralmente a organização de páginas e o conteúdo não é exatamente confortável para passarmos horas lendo.

Outro problema comum é que nós desenvolvedores não temos uma maneira fácil de formatar páginas. Na verdade temos, mas é um pouco de gambiarra e claro, não é maneira correta. A especificação de Paged Media nos possibilita formatar as páginas, transparências (aqueles “plásticos“ que usamos com retroprojetores) ou até mesmo páginas que serão vistas pelo monitor. Controlaremos suas medidas, tamanhos, margens, quebras de páginas e etc...

Nota: Para você não se confundir, quando digo páginas, quero dizer páginas

físicas, de papel, não páginas web, ok? ;-)

@PAGE

Definiremos com CSS3 um modelo de página que especifica como o documento será formatado em uma área retangular, chamada de page box, com larguras e alturas limitadas. Nem sempre o page box tem referência correspondente para uma folha de papel física, como normalmente conhecemos em diversos formatos: folhas, transparências e etc. Esta especificação formata o page box, mas é o browser ou o meio de acesso que o usuário está utilizando que tem a responsabilidade de transferir o formato do page box para a folha de papel no momento da impressão.

O documento é transferido no modelo da página para um ou mais page boxes. O page box é uma caixa retangular que será sua área de impressão. Isso é como se fosse um viewport do browser. Como qualquer outro box, a page box tem margin, border, padding e outras áreas. O conteúdo e as áreas de margin do page box tem algumas funções especicias:

A área de conteúdo do page box é chamada de área da página ou page area. O conteúdo do documento flui na área de página. Os limites da área da página na primeira página estabelece o retângulo inicial que contém o bloco do documento.

A área da margem da page box é dividido em 16 caixas de margem ou margin boxes. Você pode definir para cada caixa de margem sua própria borda, margem, padding e áreas de conteúdo. Normalmente essas caixas de margem são usadas para definir headers e footers do documento. Confesso que o nome utilizado (caixas de margem) é meio estranho.

As propriedades do page box são determinadas pelas propriedades estabelecidas pelo page context, o qual é a regra de CSS @page. Para definir a

largura e altura de uma page box não se usa as propriedades width e height. O tamanho da page box é especificada utilizando a propriedade size no page context.

Terminologia e Page Model (modelo de página)

O page box tem algumas áreas simples de se entender que facilitará a explicação. Veja abaixo uma imagem e uma explicação de suas respectivas áreas:

Page box

O page box é onde tudo acontece. Tenha em mente que o page box é o viewport das medidas impressas. É lá que conterá as áreas de margem, padding, border e onde o texto será consumido.

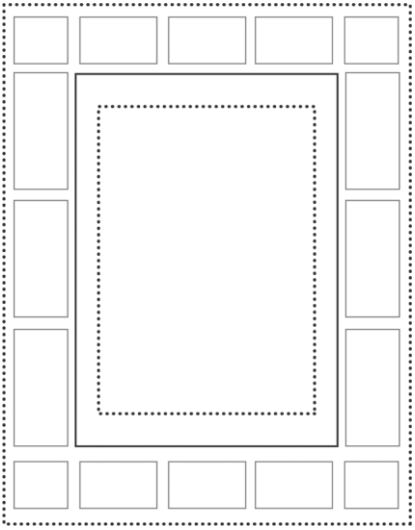
A largura e altura do page box é determinada pela propriedade size. Em um caso simples, o page box tem a largura e a altura de uma folha. Entretanto em casos complexos onde page box difere das folhas de papel em valores e orientações já que você pode personalizar de acordo com sua necessidade.

Page area

A page area é a area de conteúdo (content area) do page box.

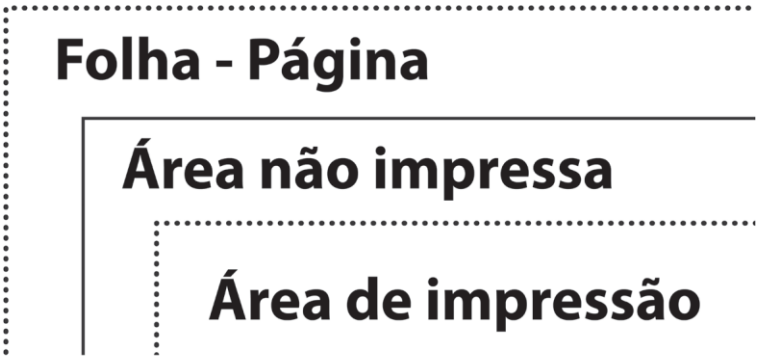
Margin box

Margin boxes contém boxes para header e footer. São conjunto de 16 caixas onde você pode inserir conteúdo útil como número da página, título do livro, etc, etc, etc. Essas áreas ficam fora do Page area. Cada um tem suas margins, paddings e bordas individuais. Veja o diagrama abaixo para visualizar melhor.



Page sheet

A folha, a página, a superfície que será impresso o conteúdo. A ilustração abaixo mostra a representação de uma folha.



Non-printable area - Área não impressa

A área de não impressão é a área onde o dispositivo de impressão não é capaz de imprimir. Esta área depende do dispositivo que você está utilizando. O page box fica dentro da área de impressão.

Área de impressão

A área impressa é onde o dispositivo de impressão é capaz de imprimir. A área de impressão é o tamanho da page sheet menos a área de não impressão. Como a área de não impressão, a área útil de impressão depende muito do dispositivo. O dispositivo pode ajustar o conteúdo para que seja impresso sem problemas nessa área. Cada dispositivo tem seu meio de ajuste.

PROPRIEDADE SIzE

A propriedade size especifica o tamanho e a orientação da área do de conteúdo, o page box. O tamanho do page box pode ser definida com valores absolutos (px) ou relativos (%). Você pode usar três valores para definir a largura e a orientação do page box:

auto

O page box irá ter o tamanho e orientação do page sheet escolhido pelo usuário.

landscape

Define que a página será impressa em modo paisagem. O page box neste caso tem o mesmo tamanho da página, mas o lado maior é o horizontal. Se o tamanho da página não for definido, o tamanho é especificado pelo usuário e seu dispositivo.

portrait

Define que a página será impressa em modo retrato. O page box neste caso tem o mesmo tamanho da página, mas o lado maior é o vertical. Se o tamanho da página não for definido, o tamanho é especificado pelo usuário e seu dispositivo.

Veja um exemplo abaixo:

```
@page {
  size: auto; /* auto é o valor padrão */
  margin: 10%; /* margem */
}
```

Como nessa caso a margem é variável, ela está sendo relativa às dimensões da página. Logo se a página uma A4, com as dimensões: 210mm x 297mm, as margens serão 21mm e 29.7mm.

Outro exemplo:

```
@page {
  size: 210mm 297mm; /* definindo page-sheet para um
tamanho de A4 */
}
```

PAGE-SIZE

O page-size pode ser especificado utilizando um dos media names abaixo. Isso é o equivalente a utilizar os valores escritos diretamente na propriedade size. Contudo é muito melhor utilizar o nome de um formato de formato de papel.

Formato	Descrição

A5	A página deve ser definida para o tamanho ISO A5: 148mm x 210mm.
A4	A página deve ser definida para o tamanho ISO A4: 210 mm x 297 mm.
A3	A página deve ser definida para o tamanho ISO A3: 297mm x 420mm.
B5	A página deve ser definida para o tamanho ISO B3 media: 176mm x 250mm.
B4	A página deve ser definida para o tamanho ISO B4: 250mm x 353mm.
letter	A página deve ser definida para o tamanho papel carta: 8.5 pol x 11 pol

Abaixo veja um exemplo de aplicação:

```
@page {
  size: A4 landscape;
}
```

O W3C tem uma especificação muito extensa que pode ser encontrada aqui: http://www.w3.org/TR/css3-page/

PRESENTATION-LEVELS

A informação na web é reutilizada de diversas maneiras. Toda informação publicada é reutilizada por diversos meios de acesso, seja o seu browser, leitor de tela ou robôs de busca. O HTML proporciona essa liberdade para a informação. Por ser uma linguagem muito simples, podemos reutilizar a informação marcada com HTML em diversos meios de acesso. Mas o HTML não cuida da forma com que o usuário vai visualizar a informação em seu dispositivo. O HTML apenas exibe a informação. A maneira que o usuário

consome essa informação é diferente em cada um dos meios de acesso e dispositivos. É aí que entra todo o poder do CSS. O CSS formata a informação para que ela possa ser acessível em diversos usar agents (meios de acesso). Se você acessa um determinado site por um monitor de 22'' ou pelo celular, a informação tem que aparecer bem organizada em ambos os cenários. É o CSS que organiza visualmente essas informações.

Além disso podemos apresentar a informação de diversas formas em um mesmo dispositivo. Por exemplo: você pode ver uma galeria de imagens da maneira convencional, clicando nas thumbs das fotos ou ver em forma de slideshow. Podemos levar essas experiências para websites de conteúdo textual também. A especificação de presentation-levels é uma das especificações que levam o usuário a terem conteúdo mostrados de uma outra forma da qual estamos acostumados. É muito útil para apresentações de slides, com efeitos, transições e etc ou qualquer documento que seria melhor apresentado no formato de apresentação, como uma proposta, documentos técnicos e etc.

Como funciona o modelo

O modelo por trás da especificação é simples. Cada elemento no documento é definido como um "elemento de apresentação" ou no formato original "element's presentation level" - EPL.

O EPL pode ser explicito em uma folha de estilo ou calculado automaticamente baseado na posição do elemento pela estrutura do documento. É assim que o browser calcula para mostrar os elementos progressivamente, como se faz normalmente em programas de apresentação.

O elemento fica em um dos três seguintes níveis que também são representadas por classes: below-level, at-level e above-level. Dependendo da pontuação de EPL que o browser dá, o elemento fica em um determinado nível. Essas pseudo-classes podem e devem ser modificadas via CSS.

A PROPRIEDADE PRESENTATION-LEVEL

A propriedade presentation-level define como os valores de apresentação (EPL) de um determinado objeto devem ser calculados. São três valores possíveis: números inteiros, increment e same.

Quando definimos um valor inteiro, o elemento tem aquele valor fixo.

Quando colocamos increment, o valor do objeto aumenta um ponto em relação ao objeto anterior. Suponha que há duas LI em uma UL. A primeira LI tem o valor de 1, a segunda tem valor de 2 e assim por diante.

Quando definimos o valor same, o browser computa o mesmo valor do objeto anterior.

Isso tudo vai ficar mais esclarecido com os exemplos a seguir.

Utilizando o mesmo exemplo da especificação do W3C, temos o código abaixo:

```
<!DOCTYPE html>
<html>
<body>
<h1>strategies</h1>
<h2>our strategy</h2>
<ul>
  <li>divide</li>
  <li>conquer
    <p>(in that order)</p>
  </li>
</ul>
<h2>their strategy</h2>
<ul>
  <li>obfuscate</li>
  <li>propagate</li>
</ul>
```

```
</body>
</html>
```

Vamos definir o CSS de presentation-levels para esse HTML adicionado o código CSS:

```
@media projection {
  h1 { page-break-before: always }
  li { presentation-level: increment }
  :below-level { color: black }
  :at-level { color: red }
  :above-level { color: silver }
}
```

Definimos que o H1 irá sempre iniciar em uma nova página.

Mas o mais importante é a propriedade presentation-level que definimos para a LI. Isso quer dizer que a cada LI o browser contará mais um ponto.

As três pseudo-classes que falamos no começo do texto: below-level, at-level, above-level, que formata os elementos que foram mostrados anteriores, o que elemento que está sendo mostrado e o próximo elemento.

Sendo assim, o browser calcula a pontuação de cada um dos elementos utilizados no exemplo como mostra abaixo:

HTML	Valor de EPL
strategies	0
our strategy	0
	0
divide	1
conquer	2
	0

their strategy	0
	0
obfuscate	1
propagate	2
	0

Temos um outro exemplo, segue abaixo o HTML e logo depois a tabela com os valores de EPL:

```
<!DOCTYPE html>
<html>
<style>
  @media projection {
  h1 { presentation-level: 0; }
  h2 { presentation-level: 1; }
  h3 { presentation-level: 2; }
  body * { presentation-level: 3; }
  :above-level { display: none; }
  }
</style>
<body>
<h1>strategies</h1>
<h2>our strategy</h2>
<ul>
  <li>divide</li>
  <li>conquer</li>
</ul>
<h2>their strategy</h2>
<ul>
  <li>obfuscate</li>
  <li>propagate</li>
</ul>

</body>
</html>
```

Perceba que agora definimos no CSS que tudo dentro de body tem o valor de

3. Logo o H1 que foi definido como 0 pela propriedade presentation-level tem o valor de 3.

Definimos também display:none; para os próximos elementos.
Agora veja a pontuação aplicada:

HTML	Valor de EPL
strategies	3
our strategy	2
	0
divide	0
conquer	0
	0
their strategy	2
	0
obfuscate	0
propagate	0
	0

O W3C em uma especificação completa e em constante atualização do presentation-levels aqui: http://www.w3.org/TR/css3-preslev/

8 BROWSERS

Motores de RenderIzAção

Quem nunca teve problemas para manter a compatibilidade do seu código entre os diversos browsers do mercado? Entender a particularidadade de cada browser é um trabalho muito complexo. Cada browser tem sua frequencia de atualização e sincronizar todas essas atualizações é praticamente impossível. Uma maneira mais segura de manter o código compatível, é nivelando 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, Mobile Safari, Android Browser, alguns celulares da Nokia e da RIM.
Gecko	Firefox, Mozilla, Camino, Flock, Firefox para Android
Trident	Internet Explorer 4 ao 9
Presto	Opera 7 ao 10

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.

Prefixos de browsers

Antes de começar os próximos assuntos, é ncessário que você conheça e saiba os porquês da utilização dos prefixos para browsers.

Muitas das características do CSS, principalmente da sua versão 3 ainda estão em fase de adequação e testes, portanto elas não foram implementadas definitivamente em alguns browsers e para evitar conflitos e também conseguir informações de feedback sobre o funcionamento dessas propriedades para fazer futuras adaptações ou correções. Outro ponto importante, é que algum destes fabricantes de browsers podem querer suportar uma determinada propriedade que ainda não faz parte do core do CSS mas que poderia ser muito útil para você utilizar em lugares específicos hoje, com muita cautela, claro.

A tabela abaixo nos mostra os prefixos dos principais browsers do mercado. Durante toda minha experiência com Web, utilizei muitas vezes os prefixos do Firefox, Safari e Opera, quase nunca utilizei o prefixo para Internet Explorer. Creio que irei utilizá-lo mais agora, já que ele está entrando pra valer com o IE9 e suas atualizações.

Safari	-webkit-
Firefox	-moz-
Opera	-o-
Chrome	-chrome-
Internet Explorer	-ms-
Konqueror	-khtml-

Como utilizar um prefixo?

Não se assuste, se você utilizará uma propriedade de CSS que ainda está sendo planejada mas ainda assim quer aplicar em seu projeto para que os usuários de novos browsers possam usufruir com uma melhor experiência ao acessar seu site, seu código pode ficar um pouco confuso. Por isso, organize-se melhor ao decidir quais propriedades você gostará de experimentar.

Para exemplo vamos utilizar a propriedade border-radius. Se quisessemos fazer uma borda arredondada com 10px, faríamos assim:

```
div {
  -webkit-border-radius: 10px;
  -moz-border-radius: 10px;
  border-radius: 10px;
}
```

Note que colocamos por último a propriedade verdadeira, sem nenhum prefixo, essa propriedade cobrirá os browsers que não precisam da utilização de prefixos para renderizar a propriedade, por exemplo o Opera e o Internet Explorer 9.

Prefixos são css-hacks?

Não. Nem se comparam.

Com os prefixos você está ajudando os fabricantes e o W3C a entenderem melhor novas propriedades. Concordo com você que o código não fica muito bonito de se ver e que pode causar muito transtorno quando mal organizado. Diferentemente dos css-hacks os prefixos fazem parte dos padrões web. Os css-hacks exploram uma falha/bug do browser, criando um código que apenas um determinado browser identifique ou ignore. Normalmente para isso usamos a sintaxe do CSS de forma errada como: **w\idth:200px; ou _width:200px**;.

Ao utilizar CSS Hacks colocamos o projeto em risco. Os browsers passarão a ignorar esse código e se seu CSS estiver baseado nele, provavelmente algo vai quebrar. Eu costumo sugerir esse tipo de hack apenas para versões muito antigas do IE como a versão 6, que é uma versão que no cenário atual oferece muitos problemas de compatibilidade.

9 COMO SERÁ?

Entenda que este livro aborda duas técnologias e alguns conceitos que estão em pauta agora no mercado de desenvolvimento web. Você já imaginou o que será do HTML ou do CSS daqui há 5 ou 10 anos? Será que ainda iremos continuar escrevendo CSS e HTML como fazemos hoje? Será que iremos utilizar outras linguagens ou vamos ainda manter os mesmos costumes e conceitos?

Eu não sei, mas provavelmente vai ser muito divertido desenvolver interfaces para celulares transparentes, televisões gigantes touch screens ou sistemas inteligentes que leem telas e respondem perguntas complexas.

Quero que você lembre que não importa o futuro, mas a informação precisa estar sempre acessível. Essa essência nunca mudará. O usuário precisa acessar a informação sem bloqueios, de forma rápida, fácil e consistente. Esteja preparado. O futuro está mais próximo do que você imagina.

Os desenvolvedores precisam acordar

O desenvolvimento web depende de três personagens principais: W3C, que regulamenta, cria e sanciona padrões para a web. Os browsers, que importam essas regras e padrões de forma que a web seja mais homogênea. E os desenvolvedores, que possibilitam a criação e publicação de conteúdo em vídeos, texto, imagem etc.

Cada um destes personagens tem papéis importantes para que a evolução da web possa se tornar forte, traçando novos caminhos, cobrindo as necessidades atuais dos usuários e prevendo necessidades e soluções futuras. Há um ciclo para que estes personagens possam cumprir com seus objetivos.

O W3C cuida dos padrões. Ele tem ideias, ele prevê problemas e tenta solucioná-los. O W3C não aplica, ele apenas planeja. É um trabalho difícil, por que é necessário uma visão muito apurada do cenário da web para saber quais caminhos ela deverá tomar. Uma decisão errada, pode acarretar problemas que levarão anos para serem solucionados por completo. Vide o desenvolvimento com tabelas. Por isso esse trabalho de planejamento deve ser meticuloso. Devo confessar que em muitos casos o W3C não supera as expectativas e faz com que iniciativas paralelas surjam e direcionem a Web para um caminho mais correto. Foi o que aconteceu com o HTML5.

Os browsers, por sua vez, precisam entender e adotar as idéias do W3C, absorvendo as soluções e criando suporte nos seus softwares. Esse trabalho também tem seus perigos. Os browsers precisam pesquisar quais das necessidades dos desenvolvedores é mais importante e assim implementá-la para que a utilização de projetos seja executada. Obviamente que alguns decidem suportar aquelas soluções que darão mais pontos estratégicos contra o concorrente.

Finalizando o ciclo, os desenvolvedores aplicam tudo o W3C define, mas apenas aquilo que os browsers “querem” ou podem suportar. E isso, claro, faz com que o desenvolvedor se depare com problemas na ponta produção. Vide o IE6/7/8 e nossos problemas de cada dia.

Durante muito tempo, esse ciclo não era afinado. Havia uma certa confusão e um jogo de interesses próprios envolvendo principalmente os browsers. O W3C estava apenas pensando em como resolver problemas que talvez existiriam daqui longos anos. Os browsers estavam apenas interessados em criar uma massa de usuários suficiente para ser o primeiro no ranking. Os desenvolvedores, por sua vez, queriam ganhar seu dinheirinho, fazer o trabalho, entregar pro cliente e acabar com o problema.

Ninguém deu atenção quando começamos a desenvolver com tabelas, fazendo com que os sites ficassem mais pesados, aumentando o tempo de desenvolvimento e o custo do projeto.

Este ciclo defeituoso foi praticado durante muito tempo. Por incrível que pareça, foram os desenvolvedores que começaram a fazer o ciclo funcionar novamente como deveria. Eles acordaram os fabricantes de browsers e também o W3C.

O W3C passou a pensar mais em problemas presentes. Solucionou problemas iminentes e que entregavam valor para os projetos.

Os browsers suportaram o mais rápido possível essas mudanças, atualizando seus engines, e fortalecendo as bases para novas soluções e flexibilizações posteriores.

Já os desenvolvedores estagnaram. Desculpe-me, serei um pouco revoltado daqui para frente. Os desenvolvedores dormiram. Quando o W3C e os browsers apresentaram soluções para problemas como transparência, bordas arredondadas, backgrounds inteligentes, utilização de fonts remotas e etc, os desenvolvedores resolveram que não era a hora dessas soluções por causa da retrocompatibilidade com browsers antigos, como o IE6. Na verdade estou sendo meio injusto aqui. Não foram todos os desenvolvedores que criaram caso com a retrocompatibilidade, foram somente os idiotas.

Nós reclamávamos que precisávamos de recursos mais inteligentes para trabalhar. Que precisávamos de idéias realmente inovadoras, que transformassem os projetos e facilitassem o desenvolvimento. Protestamos, escrevemos manifestos, postamos em nossos blogs revoltados com a falta de visão do W3C e com a pobreza do suporte dos browsers. E quando conseguimos o que queríamos, demos para trás. Amarelamos. Pedimos arrego.

Infelizmente, via-se muito disso aqui no Brasil do que no resto do mundo. Ouvi muitas desculpas como: – “Mas meu cliente usa IE6.” ou “Mas isso não

funciona em IE6.” Você é desenvolvedor. Você trata com seu cliente todos os dias. Você tem o poder de educar e convencer. E se você acha que não tem poder nenhum, por que ainda trabalha com web?

Talvez isso seja trauma do passado. Talvez não. O fato é que não podemos mais nivelar por baixo. Isso é atrasar uma “evolução” inteira.

Tenho incansavelmente falado sobre Graceful Degradation, Enhanced Progressive, HTML 5, as maravilhas do CSS 3 e etc, pois esse é o assunto que rola lá fora. Há desenvolvedores e empresas, que acham que devemos ter uma autorização especial dos gringos para utilizarmos novas tecnologias e principalmente para deixar os browsers antigos no passado.

Esse assunto me faz pensar em outra pergunta: Estamos (você está?) preparados para o ritmo alucinante do W3C e dos browsers?
Até a Microsoft está cumprindo com a palavra de ter um browser atualizado. Eles já anunciaram o novíssimo Internet Explorer 10, com uma série de atualizações que promete trazer o IE para o patamar de browsers atuais.

O mercado de client-side se transformará rapidamente nos próximos anos. Muitas mudanças no HTML e no CSS serão publicadas com o intuito de tornar a web mais uniforme, flexível e portável. O HTML 5 não é só uma coleção de novas tags e APIs. O CSS 3 não ganhou só bordas arredondadas. E nem só de iPhone vive o homem. Existem milhares de Nokias, Blackberrys, Windows Phones e Androids por aí.

Os desenvolvedores precisam acordar.