Crie websites eficientes com AJAX

AJAX: ágil e a jato

No princípio, páginas web se comportavam como livros. Graças ao AJAX, os sites modernos são mais semelhantes a aplicativos de verdade.

por Carsten Zerbst

Os padrões para uma boa apresentação na Internet são bem diferentes hoje do que eram quando Tim Berners-Lee criou as primeiras páginas da Web. Os sites se assemelham, cada vez mais, a aplicativos interativos de desktop, deixando para trás o antiquado visual de material impresso. O AJAX é uma tecnologia baseada no *JavaScript* que adiciona conveniência por meio de menus *drop-down*, tabelas ordenáveis e páginas de entrada de dados interativas. O principal avanço é a ausência de atrasos geralmente experimentados enquanto as páginas eram recarregadas.

Longo caminho

Antes de renderizar um website, o navegador e o servidor web cumprem diversas etapas (**figura 1**):

- O navegador envia uma requisição de página ao servidor.
- O servidor processa a requisição e serve o texto HTML e as imagens. Isso pode levar alguns segundos se a carga for pesada. A velocidade de transmissão da rede decide a rapidez de entrega do conteúdo. O tempo necessário ainda é perceptível, mesmo em intranets rápidas.
- Por último, o navegador lê a resposta e exibe a página. A mesma seqüência se repete para cada imagem antes de o navegador conseguir renderizar a versão final da página.

Esses três passos costumam levar vários segundos. No caso de páginas HTML sem tecnologia AJAX, os passos são repetidos até para as menores alterações.

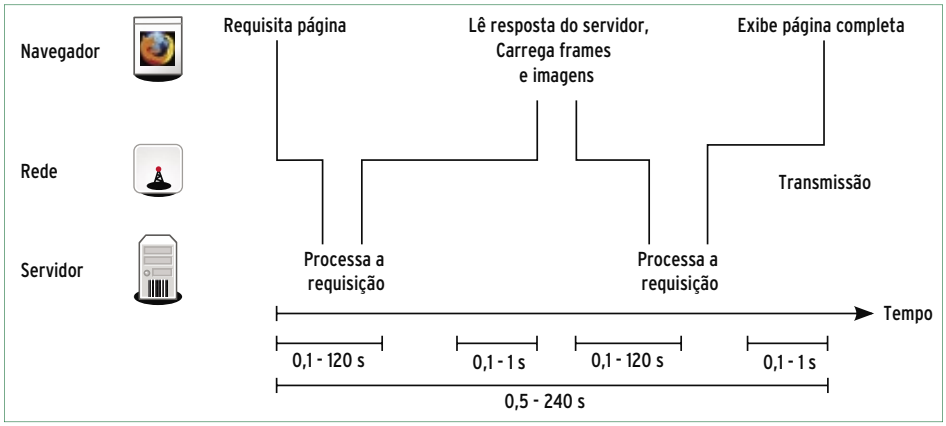

Figura 1 Pingue-pongue: a troca entre o navegador e o servidor web que ocorre para cada alteração da página sem AJAX dura vários segundos.

Diferentemente das *Rich Internet Applications*, isso afeta consideravelmente a experiência do usuário: menus que se abrem sem atraso, ordenação de tabelas com cliques do mouse ou arrastar-e-soltar não são fáceis de implementar em razão das lentas recargas de página. As páginas HTML que oferecem esses tipos de recursos precisam ser autônomas, como programas locais; ou seja, não se deve depender de uma conexão com o servidor.

Sem requisições lentas

Para melhorar a experiência do usuário, mais e mais aplicativos web estão começando a processar a entrada dos usuários diretamente no navegador, reduzindo assim as requisições lentas ao servidor.

Somente duas das várias técnicas de processamento de dados no lado do navegador alcançaram grande sucesso: JavaScript e *Flash*. Ambas estão disponíveis em mais de 90% de todos os computadores. Isso significa que desenvolvedores web não precisam ter medo de usá-las.

Outras soluções, à exceção do relativamente difundido plugin *Java*, foram incapazes de atingir um sucesso multi-plataforma tão significativo. Mas Flash e JavaScript adotam técnicas completamente diferentes entre si.

Flash

O plugin proprietário do Flash executa aplicativos Flash binários no navegador. O plugin é embutido na página da mesma forma que uma imagem *bitmap*, exceto pelo fato de que oferece ao usuário uma interface interativa. O plugin do Flash possui excelentes capacidades gráficas e praticamente não oferece restrição à criatividade do desenvolvedor. Entretanto, por falta de uma alternativa de código aberto equivalente, não há alternativas às ferramentas da Adobe para sua criação.

JavaScript

Em contraste, o JavaScript não é restrito a áreas isoladas da página. O interpretador integrado ao navegador executa o programa e converte a página inteira em uma interface modificável dinamicamente. Para isso, os scripts criam ou modificam o código HTML na página, modificam os estilos CSS e até desenham imagens.

Os scripts propriamente ditos são textos não compilados. Diferentemente do desenvolvimento em Flash, programar em JavaScript não requer ferramentas especiais. Um simples editor de texto é suficiente, ao menos para começar.

Boas ferramentas

Como sempre, boas ferramentas facilitam a programação. Um editor com suporte a HTML , CSS e JavaScript é bastante útil. Ele também deve ser capaz de lidar com código-fonte que mistura todos os três[1][2]. O plugin *Web Developer*[3], feito por Chris Pederick para o *Firefox*, é a escolha óbvia para analisar e depurar programas. Ele revela falhas em HTML, CSS e JavaScript, investiga cookies e exibe o código HTML modificado dinamicamente, não apenas a versão original entregue pelo servidor.

Tabelas

Um número crescente de aplicativos, como sistemas de processamento de pedidos ou de gestão empresarial (ERPs), usam interfaces web: listas de componentes e outras visões em listas são os que mais se vê.

Direitos	Tipo	Dono	Grupo	Tamanho	Alteração	Nome
lrwxrwxrwx	1	root	root	18	11.02.06	printcap
-rw-r--r--	1	root	root	149	26.11.06	hosts.deny
-rw-r--r--	1	root	root	188	26.11.06	hosts.equiv
-rw-r--r--	1	root	root	191	26.11.06	hosts.lpd
-rw-r--r--	1	root	root	530	27.01.07	group.YaST2save
-rw-r--r--	1	root	root	536	27.01.07	group.old
-rw-r--r--	1	root	root	551	27.01.07	group
-rw-r--r--	1	root	root	677	19.12.06	hosts.YaST2save
-rw-r--r--	1	root	root	715	27.12.06	hosts
-rw-r--r--	1	root	root	1357	02.10.06	passwd
-rw-r--r--	1	root	root	2639	26.11.06	hosts.allow

Figura 2 Nova ordem: Clicar no cabeçalho da tabela faz o *JavaScript* reordenar a tabela, sem obrigar o servidor a participar.

Um aspecto importante é a capacidade do usuário de ordenar essas listas. Se a interface web for baseada em HTML estático, o servidor precisará gerá-la novamente e servir a versão modificada. Obviamente, a ordenação baseada em código JavaScript no lado cliente torna o processo bem mais veloz.

A **figura 2** dá um exemplo de listagem de diretório implementada numa tabela HTML. Clicar no cabeçalho da tabela ordena-a pela coluna selecionada. A **figura 3** mostra como fazer isso com JavaScript. Uma tabela HTML compreende elementos `<tr>` e `<td>` aninhados, que podem ser referenciados via DOM[4].

O script inicia apagando dinamicamente todos os elementos `<tr>` (ou seja, as linhas da tabela) da página. As linhas só existem como um vetor no JavaScript. A função em JavaScript então ordena o vetor na coluna pedida e escreve a nova ordem entre as tags vazias `<table>` e `</table>`. Por último, o script desenha uma seta, um caractere de seta Unicode [5], antes do cabeçalho da coluna que funcionou como base para a ordenação.

Sistema modular

Esse método pode ser implementado com poucas centenas de linhas de código; porém, é mais fácil usar soluções já existentes. Há dezenas de bibliotecas JavaScript populares na Internet que incluem funções para tarefas freqüentemente necessárias,

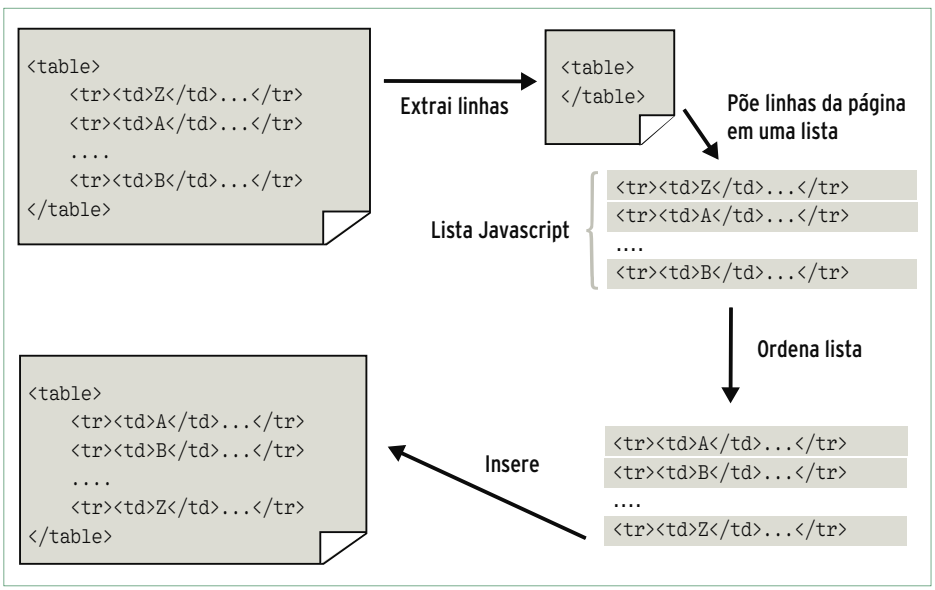

Figura 3 O *JavaScript* ordena tabelas removendo dinamicamente as linhas da tabela, fazendo *cache* dos resultados num vetor e reinserindo os dados na nova ordem.

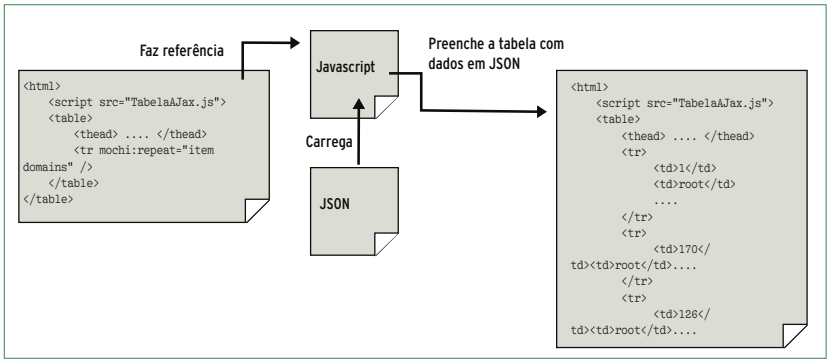

Figura 4 Mais dinâmico: se o *JavaScript* receber do servidor os dados em formato JSON, eles poderão ser ordenados e a tag <table> atualizada sem recarregar a página.

como ordenação de tabelas, criação de cantos arredondados em HTML e desenho de diagramas em árvore.

As vantagens das bibliotecas, em comparação com soluções do tipo "faça você mesmo", são o enorme escopo de funções e a compatibilidade garantida com navegadores populares. Apesar de seguir o padrão ECMA[6], as implementações de JavaScript em navegadores para Linux, Mac OS X e Windows diferem em aspectos muito importantes. A maioria das bibliotecas atuais abstraem essas diferenças para facilitar o trabalho dos desenvolvedores.

O **exemplo 1** mostra o código HTML de uma tabela ordenável baseada na biblioteca aberta *Mochikit*[7]. Além dos dois `includes` na **linha 6** que chamam o Mochikit, o código-fonte HTML é um pouco diferente de uma tabela estática. A tag `table` precisa de um ID único (`tabela Ordenavel`) assim como no exemplo de menu. Os atributos específicos do Mochikit `mochi:format="int"` e `mochi:format="gdate"` nas tags `<th>` ajudam o Mochikit a ordenar colunas numéricas e de datas corretamente.

Dados dinâmicos

O exemplo anterior ordena o conteúdo da tabela no código-fonte HTML; o exemplo a seguir vai um passo além: o aplicativo obtém o conteúdo da tabela sem recarregar a página do servidor. Links, botões e os itens do menu permitem que o usuário preencha a tabela com valores diferentes. Além disso, também é possível exibir resultados de busca sem recarregar a página.

A **figura 4** ilustra a tecnologia exibida aqui. O servidor fornece os dados em *JavaScript Object Notation* (JSON), um formato de texto que utiliza colchetes e vírgulas como separadores.

Exemplo 1: Tabela ordenável

```
01 <!DOCTYPE HTML PUBLIC "- /W3C//DTD HTML 4.01 Transitional//EN">
02 <html>
03   <head>
04     <title>Tabela ordenável</title>
05     <meta http-equiv="Content-Type" content="text/html;charset=UTF-8">
06     <link href="tabelle.css" rel="stylesheet" type="text/css" />
07     <script type="text/javascript" src="lib/MochiKit/MochiKit.js"></script>
08     <script type="text/javascript" src="Tabela ordenável.js"></script>
09   </head>
10   <body>
11     <TABLE id="Tabela ordenável"class="datagrid">
12       <THEAD>
13         <TH>Direitos</TH> ....
<TH mochi:format="int">Tamanho</TH><TH mochi:format="gdate">Alteração</TH>
14       </THEAD>
15       <TBODY>
16         <TR>
17           <TD>-rw-r--r</TD><TD>1</TD><TD>root</TD><TD>root</TD><TD>551
  </TD><TD>27.01.07</TD><TD>group</TD>
18         </TR>
19         <TR>
20           ...
21         </TR>
22       </TBODY>
23     </TABLE>
24   </body>
25 </html>
26
```

JSON

XML é uma alternativa popular ao JSON. A vantagem desse último é o menor *overhead* quando comparado ao desnecessariamente verboso XML. A função JavaScript `eval()` converte o código JSON para objetos JavaScript normais.

O **exemplo 2** exibe o código HTML. Em vez do conteúdo da tabela, o código possui apenas um elemento `tbody` para marcar o lugar. O arquivo JavaScript referido no cabeçalho da tabela, `TabelaAjax.js`, substitui o `tbody` pelo novo conteúdo com valores obtidos do arquivo JSON (**exemplo 3**).

Se as listas forem mais compridas, fica mais fácil carregar apenas as primeiras 25 linhas. Essa técnica reduz o tempo de espera do usuário, assim como a carga do servidor.

Imagens

Os exemplos até aqui se restringiram a elementos HTML simples, como listas e tabelas. Widgets e funções não fornecidos nativamente pelo HTML podem ser programados em JavaScript em combinação com CSS.

Consultar a posição do mouse permite que o desenvolvedor implemente textos auxiliares e também recursos como arrastar-e-soltar. A maioria das bibliotecas JavaScript inclui implementações.

Isso não elimina as capacidades do HTML dinâmico, incluindo imagens que o HTML legado só conseguiria implementar embutindo bitmaps.

A **figura 5** mostra uma estrutura de árvore desenhada com JavaScript. Imagens dinâmicas no lado cliente têm várias vantagens quando comparadas a bitmaps: a resolução não é fixa e portanto pode ser modificada para refletir o tamanho pré-configurado de fonte do navegador.

Dados entrados pelo usuário podem ser visualizados em interação com o servidor, reduzindo a carga sobre este. Normalmente, boa parte da banda de um servidor web é consumida pelo serviço de imagens.

Extensões HTML

Existem basicamente duas técnicas para se usar JavaScript na criação de imagens. A linguagem SVG[8], baseada em XML, é a que oferece o maior número de recursos, em comparação com a *Vector Markup Language* (VML)[9]. Elas podem ser embutidas em HTML como imagens normais, mas também podem acrescentar elementos gráficos típicos – como linhas, áreas ou texto em tempo de execução. Portanto, elas podem responder interativamente à entrada de dados pelo usuário, da mesma forma que o HTML dinâmico modificado por JavaScript.

Programas de desenho, como o *Inkscape*[10] ou o *Karbon*[11], podem ser úteis na criação de rascunhos.

Infelizmente, não há como garantir a compatibilidade multi-navegador: nenhum dos navegadores mais populares suporta todos os três formatos.

O Firefox suporta SVG e *Canvas*[12], o Safari apenas Canvas, e o Internet Explorer apenas VML.

O Google possui duas bibliotecas JavaScript: uma para SVG[13] e outra para Canvas[14]; entretanto, não é garantido o suporte na plataforma Windows.

Ferramentas atestadas

Muitos diagramas e imagens podem ser criados com JavaScript e as ferramentas HTML e CSS padrão, como o diagrama em árvore da **figura 5**.

As caixas são compostas por elementos CSS `div` livremente posicionados. Se for usado `em` em vez de `px` (pixel) como unidade para localização e tamanho, as dimensões e a posição da caixa serão baseadas no tamanho da letra *m*.

Exemplo 2: Tabela gerada dinamicamente

```
01 <!DOCTYPE HTML PUBLIC " //W3C//DTD HTML 4.01 Transitional//EN">
02 <html>
03   <head>
04     <link href="tabelle.css" rel="stylesheet" type="text/css" />
05     <script type="text/javascript" src="lib/ MochiKit/MochiKit.js"></script>
06     <script type="text/javascript" src="TabelaAjax.js "></script>
07   </head>
08   <body>
09     <a href="index.html">Exemplos</a>
10     <hr>
11     <h4>Ajax Table</h4>
12     <p>
13       <a href="top.json" mochi:dataformat="json">Recarregar Tabela 1</a><br>
14       <a href="top2.json" mochi:dataformat="json">Recarregar Tabela 2</a>
15     </p>
16     <table id="sortable_table" class="datagrid">
17       <thead>
18         <tr>
19           <th mochi:sortcolumn="PID int">PID</th>
20           <th mochi:sortcolumn="USER str">USUÁRIO</th>
21           [...]
22           <th mochi:sortcolumn="COMMAND str">COMANDO</th>
23         </tr>
24       </thead>
25       <!   substituído pelo conteúdo do arquivo JSON   >
26       <tbody class="mochi template">
27         <tr mochi:repeat="item domains">
28           <td mochi:content="item.PID"></ td>
29           <td mochi:content="item. USER"></td>
30           [...]
31           <td mochi:content="item. COMMAND"></td>
32         </tr>
33       </tbody>
34     </table>
35   </body>
36 </html>
```

Exemplo 3: Dados JSON

```
01 {
02   "columns": [ "PID", "USER","PR","NI", "VIRT", ➥"RES", "SHR",
➥"S", "CPU","MEM", "TIME", "COMMAND"],
03   "rows": [
04     [ "6620","cz","15","0","166912","57344","40960",
➥"S","11.6","5.6", "0:02.40", "soffice.bin"],
05     [ "3701","root","15","0","241664","225280","17408",
➥"S","4","21.8", "4:23.50", "X"],
06     [ "4496","cz","15","0","63828","20480","15360",
➥"R","2","2", "0:16.02", "gnome-panel"],
07     [ "4506","cz","15","0","70480","18432","10240",
➥"R","2","1.8", "0:04.14", "gnome-terminal"],
08   ]
09 }
```

A figura inteira será redimensionada para refletir o tamanho do texto, sem qualquer esforço por parte do desenvolvedor.

Os usuários também podem redimensionar facilmente a figura, na maioria dos navegadores, com a roda do mouse.

Ligue as linhas

Desenhar linhas de conexão é um pouco mais difícil. Como o JavaScript não tem suporte a elementos gráficos como linhas ou círculos, o truque nesse ponto é desenhar os elementos gráficos, pixel por pixel, como elementos `div`. O *jsGraph* de Walter Zorn abstrai esse processo complexo, dando ao desenvolvedor formas básicas como linhas, círculos e polígonos.

A função `connect()` se baseia nelas para desenhar duas caixas com linhas. Ela assegura as posições da caixa em tempo de execução para permitir que a imagem seja redimensionada para o tamanho da fonte.

A função `displayHierarchy()`, em seguida, redesenha as linhas de conexão.

O *TextResizeDetector*[15] de Lawrence Carvalho chama essa função sempre que o usuário altera o tamanho da fonte. O resultado é um widget AJAX que pode ser aproximado sem perdas, feito para estruturas de árvore que não podem ser implementadas com bitmaps.

Como ele se baseia inteiramente em elementos HTML, não há necessidade de extensões como Canvas ou VML que não estejam disponíveis para alguns navegadores. Pelo menos a caixa de texto é legível em navegadores sem suporte JavaScript e CSS.

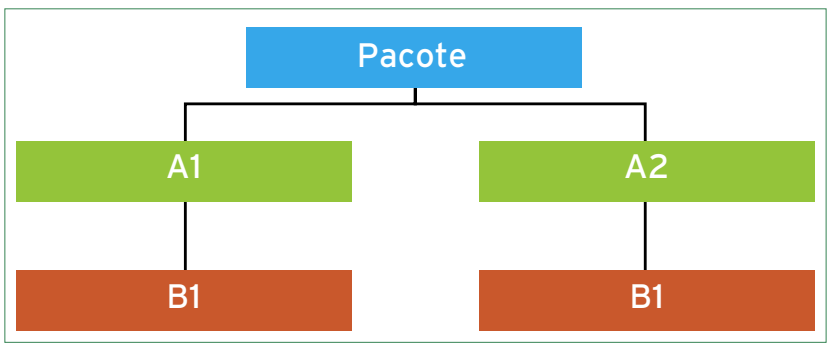

Figura 5 Melhor que bitmaps: A imagem em *JavaScript* reage dinamicamente a mudanças de tamanho de fontes e economiza banda.

Prós e contras

Muitos sites se beneficiam do uso de JavaScript e AJAX em relação à usabilidade. Menores tempos de resposta e a independência de recarregamento de páginas são muito bem recebidos pelos usuários. Entretanto, o uso de AJAX pode causar problemas que não ocorrem em páginas estáticas. Por exemplo, os usuários não podem simplesmente clicar nos botões *Avançar* e *Voltar* para navegar.

As páginas modificadas dinamicamente por JavaScript aumentam as expectativas dos usuários. O problema afeta até os simples menus dinâmicos citados acima. Se os usuários clicarem para abrir um submenu, o botão *Voltar* não os levará ao estado anterior da página; em vez disso, ele abrirá a página visitada anteriormente.

Isso talvez não seja um grande problema para um menu, mas se o script no lado cliente alterar substancialmente a página, fazendo-a parecer uma nova página do ponto de vista do usuário, é provável que haja confusão.

O navegador não consegue detectar as mudanças de estado que ocorrem numa página AJAX porque a URL continua a mesma. A lógica JavaScript do lado cliente aplica as mudanças sem recarregar, o que explica a incapacidade do navegador em capturar o status da página. Se o website utilizar a navegação baseada em AJAX, os bookmarks simplesmente levarão o usuário à página inicial.

A primeira coisa a ser levada em conta é o que é mais útil para o usuário: um histórico funcional e a capacidade de adicionar subpáginas aos favoritos ou a resposta

rápida. Apesar de ser sempre elegante o uso de código JavaScript no lado cliente para abrir menus ou reordenar tabelas sem recarregar páginas, a usabilidade de uma página de loja online ou catálogo com centenas de subpáginas seria seriamente afetada caso os usuários perdessem a capacidade de navegar com os botões de avanço e retrocesso.

Soluções

Há soluções para os problemas do histórico e dos favoritos. Por exemplo, o Google Maps fornece uma URL alternativa com os parâmetros do `GET` para a seção específica do mapa.

Os usuários não conseguem adicionar aos favoritos a página carregada no navegador. Em vez disso, um clique direito no link mostrado na página adiciona um favorito.

Outras soluções usam a âncora HTML normalmente usada para armazenar posições específicas de uma página em uma URL. A âncora é a parte da URL após o sinal de tralha (#).

Assim como a própria URL, a âncora pode ser modificada por meio do JavaScript sem recarregar a página. Se o navegador não conseguir encontrar uma tag de âncora para o texto após a tralha, a tela permanecerá inalterada. A âncora, portanto, é perfeita para guardar informações de estado: a parte da âncora é a única na URL que pode ser modificada sem exigir o recarregamento.

As capacidades de imagens em JavaScript são bem espartanas quando comparadas ao Flash: HTML e CSS só conseguem desenhar quadrados e texto. Bibliotecas como a jsGraphics adicionam outras formas, como círculos e polígonos. Imagens SVG ou Canvas, um elemento do futuro padrão web HTML 5, adicionam mais possibilidades quando embutidas na página. Porém, nenhuma delas é adequada para sites publicamente acessíveis, pois não estão disponíveis para todos os navegadores web. ■

Mais informações

[1] jEdit: http://www.jedit.org

[2] NetBeans: http://www.netbeans.org

[3] Plugin Web Developer para o Firefox: http://chrispederick.com/work/web-developer

[4] Document Object Model: http://www.w3.org/DOM

[5] Setas Unicode: http://www.alanwood.net/unicode/arrows.html

[6] Padrão ECMA: http://www.ecma-international.org/publications/standards/Ecma-262.htm

[7] MochiKit: http://www.mochikit.com

[8] SVG: http://www.w3.org/Graphics/SVG

[9] VML: http://www.w3.org/TR/1998/NOT-VML-19980513

[10] Inkscape: http://www.inkscape.org

[11] Karbon: http://www.koffice.org/karbon

[12] Elemento Canvas do HTML: http://www.w3.org/html/wg/html5/#the-canvas

[13] SVG2VML: http://code.google.com/p/svg2vml

[14] ExplorerCanvas: http://excanvas.sourceforge.net

[15] TextResizeDetector: http://www.alistapart.com/articles/fontresizing