Construindo sites dinâmicos e rápidos

O poder do Ajax

www.sxc.hu - Ula Kapala

A tecnologia Ajax traz recursos para melhorar muito o desempenho de sites lentos e pesados. Tudo que ela requer é um script Perl no servidor e um pouco de JavaScript no cliente.

Por Michael Schilli

Desenvolvedores web foram acordados aos tapas quando o Google lançou o serviço *Maps* [1], em abril de 2005. De repente, os usuários passaram a poder mover mapas online de maneira rápida, como se o aplicativo estivesse instalado localmente.

Nesse serviço, todo o trânsito de pacotes na rota cliente-servidor começou a passar despercebido, já que a página não precisa ser recarregada para refletir as mudanças no aplicativo. Hoje, aplicativos *Ajax* (Asynchronous JavaScript e XML) estão pipocando por toda a web. A versão beta do *Yahoo! Webmail* (aberta apenas para testadores beta), por exemplo, se parece muito com um programa desktop. Se você não olhar com atenção, nem vai perceber que é uma página rodando no navegador.

A técnica de desenvolvimento web Ajax se baseia em *HTML* dinâmico e *JavaScript* (no cliente). Além disso, o objeto *XMLHttpRequest* – criado pela Microsoft e ignorado até o Google levá-lo à fama – permite que um JavaScript baixado de um site troque dados de modo assíncrono com o servidor web. Assim, ele "contrabandeia" dinamicamente esses dados para a página HTML, fazendo com que pequenas mudanças sejam atualizadas na página sem a necessidade de recarregá-la inteira.

Figura 1: O *Respostas*, nossa aplicação de exemplo para gerenciar respostas prontas de email.

A **figura 1** mostra nossa aplicação de exemplo, que gerencia textos usados freqüentemente em respostas de email e os disponibiliza em um campo de texto para que sejam copiados e colados nas mensagens.

Clique em *Acrescentar novo tópico* para adicionar um novo texto a ser armazenado no servidor. Já o comando *Atualizar* salva o texto correspondente. O link *Remover* apaga o item selecionado.

A página é carregada apenas uma vez no navegador. Os links sublinhados até podem parecer links normais, mas não fazem o navegador saltar para nenhuma outra página. Em vez disso, simplesmente executam o JavaScript especificado pelo manipulador (*handler*) `OnClick`, "conversando" com o servidor nos bastidores.

O módulo Perl *CGI:Ajax*, de Brent Pedersen, facilita bastante a implementação desse tipo de mecanismo. Ele define um protocolo cliente-servidor (em JavaScript e Perl), que o JavaScript no cliente pode usar para chamar funções Perl no servidor por referenciamento de seus nomes e listas de parâmetros.

Um objeto JavaScript XMLHttpRequest (chamado *ActiveXObject ("Microsoft.XMLHTTP")* no Internet Explorer) permite que o navegador envie uma solicitação *GET* ao script *CGI* no servidor. O pedido ativa uma função Perl previamente especificada, que retorna um ou mais valores. O código JavaScript pega esses dados e os insere em campos pré-definidos na página carregada.

O objeto *CGI::Ajax* – criado com `CGI::Ajax -> new('display' = > \&display, ...)` – no script CGI garante que o HTML enviado de volta contenha uma seção JavaScript. O aplicativo web usa uma função JavaScript chamada `mostrar` para preencher a área de texto no navegador com determinado texto. Correspondentemente, uma função Perl chamada `mostrar()` é definida no servidor. O manipulador JavaScript mais tarde chama essa função usando uma solicitação HTTP.

Um botão de opção definido no HTML com o manipulador (*handler*) `OnClick` (`OnClick="display(['Desculpa esfarrapada'],['tarea', 'statusdiv'])"`) chama a função JavaScript `mostrar ()` e passa as duas matrizes (*arrays*) especificadas para a função.

A primeira matriz (array) contém o atributo *id* do botão de opção que está selecionado no momento. O seu valor de texto ("Desculpa esfarrapada") é passado para a função no servidor. A segunda contém os atributos id das tags HTML, que o manipulador atualiza com o valor de retorno da função do servidor, após completar a solicitação. Dessa maneira, tanto a área de texto quanto o campo de status são atualizados.

O botão de seleção com o id "Desculpa esfarrapada" também tem a propriedade `VALUE="Desculpa esfarrapada"`, o que significa que a função Perl `mostrar()` no servidor (linha 12 da **listagem 1: respostas**) recebe esse conjunto de caracteres como seu primeiro parâmetro. A função `mostrar()` não faz nada mais do que receber a resposta correspondente à opção "Desculpa esfarrapada" do cache do servidor e a enviar ao navegador, junto com uma mensagem de status.

Aqui é onde o manipulador de eventos JavaScript entra de novo, atualizando o campo de texto maior (`id='tarea'`) e o campo de status na parte de baixo (`id='statusdiv'`) com as strings retornadas por `mostrar()`. Tudo isso é manipulado tranqüilamente com o CGI::Ajax, que envia o código JavaScript requerido para o navegador e prepara o manipulador no servidor para acessar as funções Perl.

Contudo, o script Perl `respostas` faz mais do que apenas atualizar os campos de texto. Se o usuário apagar alguns dos tópicos clicando em *Remover*, isso não apenas apaga o botão de opção, mas também seleciona o primeiro tópico da lista que sobra, além de carregar a resposta equivalente. Isso porque o módulo CGI::Ajax ainda não pode fazer truques como esse no cliente. Mas funções de JavaScript vão nos ajudar a chegar lá.

Um lado bem positivo é que o CGI::Ajax é fácil de usar. Como mostra a **listagem 1**, você apenas precisa definir funções para as diversas ações no cliente (remover/atualizar/mostrar) e criar uma função `mostrar_html`, que gera a aplicação HTML no cliente durante o carregamento inicial. ➠

Listagem 1: respostas

```
01 #!/usr/bin/perl -w
02 use strict;
03 use CGI;
04 use CGI::Ajax;
05 use Cache::FileCache;
06 use Template;
07
08 my $cache =
09   Cache::FileCache->new();
10
11 ##############################
12 sub mostrar {
13 ##############################
14  my ($topico) = @_;
15
16  return $cache->get($topico),
17    "$topico obtido";
18 }
19
20 ##############################
21 sub me_remova {
22 ##############################
23  my ($topico) = @_;
24
25  $cache->remove($topico);
26  return "$topico apagado";
27 }
28
29 ##############################
30 sub me_atualize {
31 ##############################
32  my ($topico, $texto) = @_;
33
34  $cache->set($topico, $texto);
35
36  my $disptext = $texto;
37  $disptext =
38    substr($texto, 0, 60)
39    . "..."
40    if length $texto > 60;
41  return
42    "Tópico '$topico' atualizado"
43    . "with '$disptext'";
44 }
45
46 ##############################
47 sub mostrar_html {
48 ##############################
49  my $modelo =
50    Template->new();
51
52  my @keys =
53    sort $cache->get_keys();
54
55  $modelo->process(
56   "snip.tmpl",
57   { topics => \@keys },
58   \my $resultado)
59    or die $modelo->error();
60
61  return $resultado;
62 }
63
64 ##############################
65 # main
66 ##############################
67 my $cgi = CGI->new();
68 $cgi->charset("utf-8");
69
70 my $pjx = CGI::Ajax->new(
71  'mostrar'   => \&mostrar,
72  'me_atualize' => \&me_atualize,
73  'me_remova' => \& me_remova
74 );
75 print $pjx->build_html($cgi,
76  \&show_html);
```

Listagem 2: respostas.js

```
001 // ###################################################
002 function adicionar_topico(topico) {
003 // ###################################################
004   var itemTable = document.getElementById("topicos");
005   var newRow    = document.createElement("TR");
006   var newCol1   = document.createElement("TD");
007   var newCol2   = document.createElement("TD");
008   var input     = document.createElement("INPUT");
009
010   if(topic.length == 0) {
011       alert("Nenhum nome de tópico especificado.");
012       return false;
013   }
014
015   input.name    = "r";
016   input.type    = "radio";
017   input.id      = topico;
018   input.value   = topico;
019   input.onclick = function() {
020       mostrar([topico], ['tarea', 'statusdiv']);
021   };
022   input.checked = 1;
023   newCol1.appendChild(input);
024
025   var textnode = document.createTextNode(topico);
026   newCol2.appendChild(textnode);
027
028   itemTable.appendChild(newRow);
029   newRow.appendChild(newCol1);
030   newRow.appendChild(newCol2);
031
032   document.getElementById('tarea').value = "";
033   document.getElementById('novo_topico').value = "";
034
035   return false;
036 }
037
038 // ###################################################
039 function atualizar_topico() {
040 // ###################################################
041   if(!id_selected()) {
042     alert("Crie um novo topico antes");
043     return;
044   }
045   me_atualize( [ id_selected(), 'tarea' ],
046              'statusdiv');
047 }
048
049 // ###################################################
050 function remover_topico() {
051 // ###################################################
052   var sel = id_selected();
053
054   if(!sel) { alert("Nenhum tópico disponível");
055     return;
056   }
057   remove_me([sel], 'statusdiv');
058
059   var node = document.getElementById(sel);
060   var row  = node.parentNode.parentNode;
061   row.parentNode.removeChild(row);
062   select_first();
063 }
064
065 // ###################################################
066 function selecionar_primeiro() {
067 // ###################################################
068   var form = document.getElementById("form");
069   if(! form.r) { return; }
070   if(! form.r.length) {
071     form.r.checked = 1;
072     if(! document.getElementById(id_selected()) ) {
073        document.getElementById('tarea').value = "";
074       return;
075     }
076     display([id_selected()], ['tarea', 'statusdiv']);
077   }
078
079   for(var i = 0; i < form.r.length; i++) {
080     form.r[i].checked = 1;
081     break;
082   }
083   display([id_selecionado()], ['tarea', 'statusdiv']);
084 }
085
086 // ###################################################
087 function id_selecionado() {
088 // ###################################################
089   sel = id_selecionado_prim_pass();
090
091   if(! document.getElementById(sel) ) {
092     document.getElementById('tarea').value = "";
093     return;
094   }
095   return sel;
096 }
097
098 // ###################################################
099 function id_selecionado_prim_pass() {
100 // ###################################################
101   var form = document.getElementById("form");
102   if(! form.r) { return 0; }
103   if(! form.r.length) { return form.r.id; }
104
105   for(var i = 0; i < form.r.length; i++) {
106     if(form.r[i].checked) {
107       return form.r[i].id;
108     }
109   }
110   alert("ID selecionado desconhecido");
111   return 0;
112 }
```

HTML e Perl

O script Perl respostas pega o HTML que vai ser enviado no modelo `respostas.tmpl`, mostrado na **figura 3**. A biblioteca de modelos carrega esse modelo e fornece diversas opções para inserir simples laços (*loops*) ou condições `for` no HTML. Ele faz isso sem usar uma linguagem de programação completa de propósito, para evitar a mistura de lógicas de aplicação e renderização.

Ele usa a expressão `[% FOREACH topico = topicos %]` para fazer uma iteração na matriz `@topicos` (previamente fornecida pelo script `respostas` com as respostas dos tópicos) e fornece como saída diversos botões de opção, cada um em uma linha de tabela separada.

`[% topico %]` retorna o valor da variável `topico`. A propriedade `id` de cada botão de opção é associada à string de texto do tópico. E o manipulador `OnClick` chama a função `mostrar()`, já descrita antes, que está tanto no JavaScript do cliente quanto no ambiente Perl do servidor.

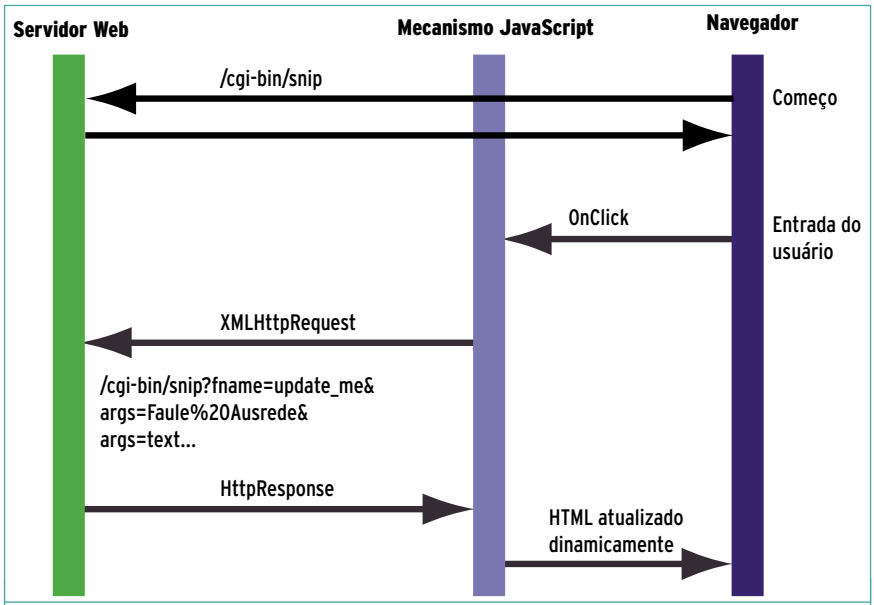

Figura 2: A comunicação entre o navegador, o mecanismo *JavaScript* e o servidor web.

A função JavaScript `selecionar_primeiro()` seleciona a primeira entrada na lista de botões de opção e solicita o texto

Figura 3: O modelo *HTML* que o script respostas processa com a biblioteca de modelos.

de resposta correspondente ao servidor para que possa exibi-lo no campo maior de texto. Primeiro, ela chama o método `document.getElementById` para procurar pela tag `HTML` com a ID `form` (o "form" HTML em `respostas.tmpl`). Todos os botões de opção são nomeados com a extensão *.r*, então `form.r` deve ser uma matriz, incluindo entradas para todos os botões de opção encontrados. Claro que se a lista de tópicos estiver vazia, a função `selecionar_primeiro()` não seleciona nada, nem se comunica com o servidor.

É vital que a tag de links `A` – usada no script `respostas` – tenha um `return false` como última ação em seu manipulador `OnClick`. Isso assegura que o navegador execute o JavaScript associado com o link, ao invés de seguir o falso atributo `HREF`.

O modelo carrega a biblioteca JavaScript `respostas.js` primeiro (**listagem 2**). Essa biblioteca fornece diversas funções para permitir que a interface rode corretamente. A função JavaScript `adicionar_topico()` espera a string de um tópico e adiciona uma entrada com esse nome no final da tabela de botões de opção. Para fazer isso, ela primeiro precisa localizar essa tabela, criar duas colunas, uma nova linha e um novo botão de opção. Ao final, o item é acrescentado.

A função `remover_topico()` apaga o tópico da lista de botões e envia uma solicitação ao servidor, que então apaga o texto de resposta de seu cache.

Já a função `id_selecionado()` fornece a propriedade `id` do botão selecionado, ou seja o tópico que o usuário quer ver. Se a lista estiver vazia, o *Firefox* pode se confundir e mesmo assim retornar um valor qualquer. Para contornar esse bug do navegador, a função `id_selecionado()` checa novamente o resultado de `id_selecionado_prim_pass()` e fornece `undefined` se flagrar o Firefox se atrapalhando.

Se a lista de botões de opção tiver duas ou mais entradas, `form.r.lenght` fornece o tamanho da lista. Se a lista tiver apenas uma única entrada, `form.r.lenght` fornece um valor não-definido. Se a lista estiver vazia, `form.r` fica como não-definido. A flag `checked` pode então ser verificada via `form.r.checked`, ou com o elemento "i" na matriz (array) `form.r[i].checked`. Depois disso, a função `selecionar_primeiro()` chama a função `mostrar()` para pegar o texto de resposta do tópico selecionado no servidor.

Arestas

Esse script foi feito apenas como um exemplo e poderia ser bastante aperfeiçoado. Entre os problemas, você pode ter dificuldades de compatibilidade com outros navegadores além do Firefox. Mas há alguns toolkits comerciais para garantir que até o *Internet Explorer 4.0* de sua avó abra sites em Ajax de maneira aceitável (apesar de diferente). Para manter o script simples, o código para a manipulação de erros foi deixado de lado. Você vai precisar cavar fundo nas profundezas lamacentas desse JavaScript e definir mensagens a serem exibidas para o usuário.

E muito cuidado. No Ajax, os detalhes também pregam peças.

Instalação

O script requer os módulos *Class::Accessor*, *CGI::Ajax* e *Template* do *CPAN*. Acrescente então o script `respostas` (como executável) e o modelo `respostas.tmpl` ao diretório *cgi-bin* do servidor web, e o JavaScript `respostas.js` logo abaixo da raiz dos documentos (normalmente `htdocs`).

Se o terminal usado para a ação de recortar e colar não suportar *UTF-8*, comente a linha 68 em `respostas`, para especificar que o servidor web deve usar o conjunto de caracteres *iso-8859-1* no cabeçalho CGI. Se você preferir não ter o cache de documentos em `/tmp`, é possível especificar isso corrigindo a linha "0" para `my $cache = Cache::FileCache->new({cache_root => "/path"})`. Finalmente, digite no navegador o endereço http://server/cgi-bin/respostas e – supondo-se que o JavaScript está habilitado – o aplicativo deve rodar, conversar com o servidor e manter o repositório de respostas atualizado. ■

Informações

[1] Google Maps: www.maps.google.com

[2] Códigos originais desse artigo (em inglês): www.linux-magazine.com/Magazine/Downloads/62/Perl