

Páginas Web más dinámicas gracias a AJAX

# PODER AJAX

La tecnología AJAX añade elementos dinámicos para mejorar las páginas más sosas. Todo lo que necesitamos es un programa en Perl en el lado del servidor y algo de código JavaScript en la parte del cliente. **POR MICHAEL SCHILLI**

Los desarrolladores Web despertaron abruptamente cuando Google presentó su servicio Google Maps. De repente los usuarios podían mover los mapas de manera dinámica, como si la aplicación se ejecutara en una interfaz gráfica de usuario, en lugar de en un navegador. Súbitamente, los tiempos de retardo cliente-servidor eran difíciles de apreciar, dado que la página no tenía que recargarse para reflejar cambios en el estado de la aplicación. Hoy día las aplicaciones AJAX brotan por doquier por toda Internet. La beta de Yahoo! Webmail, por ejemplo, se parece bastante a una apli-

cación de escritorio. Hay que fijarse con mucho detenimiento para apreciar que es nuestro navegador el que está detrás del espectáculo.

AJAX (JavaScript Asíncrono y XML) se basa en HTML dinámico y JavaScript en la parte cliente. El Objeto *XMLHttpRequest* fue añadido originalmente por Microsoft y anduvo revoloteando durante un tiempo hasta que Google lo lanzó al estrellato. AJAX permite que un script escrito en JavaScript y descargado de una página web intercambie información de manera asíncrona con el servidor web. Pasa de esta manera información “de contrabando” de forma dinámica hasta la página HTML, ya que en principio sólo van a ocurrir cambios pequeños en la página.

La Figura 1 muestra una aplicación de ejemplo que administra fragmentos de texto usados habitualmente en correos electrónicos y los sirve en un campo de texto para cortar y pegar. Podemos seleccionar *Add new topic* para añadir un fragmento de texto y que se guarde en el servidor. *Update* envía el texto correspondiente al servidor y *Remove* lo elimina.

La página se carga sólo una vez por el navegador. Los enlaces subrayados pueden parecer hiperenlaces, y se pueden pulsar, pero no provocan que el navegador salte hasta una nueva URL. En vez de esto, simplemente se ejecuta el código JavaScript especificado por el manejador del evento *OnClick*, comunicándose de esta manera con el servidor tras bastidores. El módulo de Perl *CGI::Ajax* escrito por Brent Pedersen simplifica enormemente la implementación de este mecanismo. Dicho módulo define un protocolo cliente-servidor (en Java-Script y Perl), que puede usar el código en JavaScript de la parte cliente para llamar a las funciones Perl de la parte servidor, referenciando sus nombres y su lista de parámetros. Un objeto JavaScript *XMLHttpRequest* (*ActiveXObject("Microsoft.XMLHTTP")* en IE) habilita al navegador para que envíe peticiones GET al script CGI de la parte servidor. La petición activa una función Perl previamente especificada que devuelve uno o más valores. El código JavaScript los recoge y refresca los campos predefinidos en el explorador.

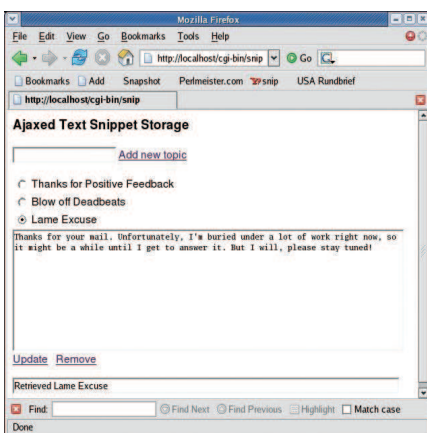


Figura 1: El administrador de fragmentos de texto en el navegador.

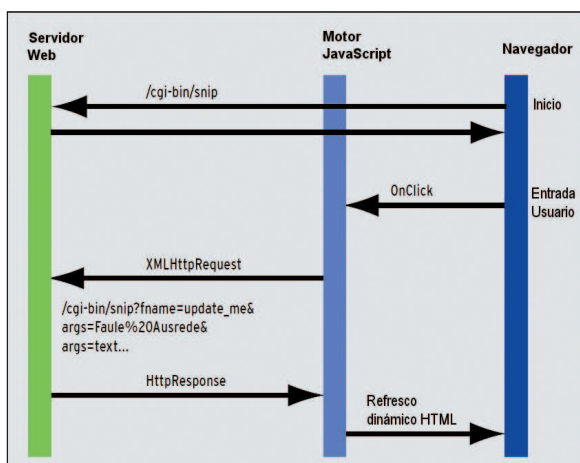


Figura 2: Comunicación entre el navegador, el motor de JavaScript y el servidor web.

El objeto `CGI::Ajax` creado por `CGI::Ajax->new('display' => \&display...)`, en el script CGI asegura que el HTML devuelto contiene una sección JavaScript. La aplicación web usa una función llamada `display` para rellenar las áreas de texto del navegador con el fragmento de texto seleccionado en ese momento. En correspondencia, se define en el servidor una función Perl llamada `display()`. El manejador JavaScript la llama posteriormente usando una petición HTTP request. Un

botón radio que se definió en HTML con el manejador del evento `OnClick`, `OnClick = "display(['Lame Excuse'], ['tarea', 'status-div'])"`, hace la llamada a la función JavaScript `display()` y le pasa los dos arrays señalados a la función. El primer array contiene el atributo `id` del botón radio seleccionado. Su texto seleccionado por `CGI::Ajax`, que envía el código JavaScript necesario al navegador, y configura el manejador para que pueda acceder a las funciones Perl.

refrescadas por el manejador con el valor de retorno de la función del servidor tras completar la petición. De esta manera, tanto el área de texto como el estado del campo de texto se actualizan con información fresca del servidor.

En nuestro caso, el botón de radio con la `id` "Lame Excuse", tiene también la propiedad `VALUE = "Lame Excuse"`, lo que quiere decir que a la función Perl `display()` de la parte servidor (línea 12 en el Listado `snip`) se le pasa esta cadena de texto como primer parámetro.

`display()` no hace otra cosa, aparte de recoger el bloque de texto que coincide con "Lame Excuse" desde la caché de la parte servidor y de enviarlo de vuelta hasta el navegador junto con un mensaje de estado. Aquí es donde el manejador de eventos de JavaScript aparece de nuevo, refrescando el campo de texto grande (`id = 'tarea'`) y el campo de estado de la parte inferior (`id = 'status-div'`) con las cadenas devueltas por `display()`. Todo esto se abstrae pulcramente por `CGI::Ajax`, que envía el código JavaScript necesario al navegador, y configura el manejador para que pueda acceder a las funciones Perl.

En la parte cliente, `snip` hace mucho más que simplemente refrescar los campos de texto. Si el usuario elimina uno de los títulos al pulsar `Remove`, no sólo borra el botón de radio con el título, si no que también selecciona el primer título en la lista restante y carga el bloque de texto desde el servidor. `CGI::Ajax` no puede tratar el entramado de la parte cliente de esta manera: funciones adicionales en JavaScript vendrán en su ayuda.

Como nota positiva destacar que `CGI::Ajax` es extremadamente fácil de usar. Como muestra el Listado 1, sólo

Listado 1: snip

```

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 $cache =
09 Cache::FileCache->new();
10
11 #####
12 sub display {
13 #####
14 my ($topic) = @_;
15
16 return $cache->get($topic),
17 "Retrieved $topic";
18 }
19
20 #####
21 sub remove_me {
22 #####
23 my ($topic) = @_;
24
25 $cache->remove($topic);
26 return "Deleted $topic";
27 }
28
29 #####
30 sub update_me {
31 #####
32 my ($topic, $text) = @_;
33
34 $cache->set($topic, $text);
35
36 my $disptext =
37 $disptext =
38 substr($text, 0, 60)
39 . "...";
40 if length $text > 60;
41 return
42 "Topic '$topic' updated "
43 . "with '$disptext'";
44 }
45
46 #####
47 sub show_html {
48 #####
49 my $template =
50 Template->new();
51
52 my @keys =
53 sort $cache->get_keys();
54
55 $template->process(
56 "snip.tmpl",
57 { topics => \@keys },
58 \my $result)
59 or die $template->error();
60
61 return $result;
62 }
63
64 #####
65 # main
66 #####
67 my $cgi = CGI->new();
68 $cgi->charset("utf-8");
69
70 my $pjax = CGI::Ajax->new(
71 'display' => \&display,
72 'update_me' => \&update_me,
73 'remove_me' => \&remove_me
74 );
75 print $pjax->build_html($cgi,
76 \&show_html);

```

## Listado 2: snip.js

```

001 // #####
002 function topic_add(topic) {
003 // #####
004 var itemTable =
    document.getElementById("topic
    s");
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("No topic name
    specified.");
012 return false;
013 }
014
015 input.name = "r";
016 input.type = "radio";
017 input.id = topic;
018 input.value = topic;
019 input.onclick = function() {
020 display([topic],[ 'tarea',
    'statusdiv' ]);
021 };
022 input.checked = 1;
023 newCol1.appendChild(input);
024
025 var textnode =
    document.createTextNode(topic)
    ;
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
    ('new_topic').value = "";
034
035 return false;
036 }
037
038 // #####
039 function topic_update() {
040 // #####
041 if(!id_selected()) {
042 alert("Create a new topic
    first");
043 return;
044 }
045 update_me( [
    id_selected(), 'tarea' ],
046 'statusdiv');
047 }
048
049 // #####
050 function topic_remove() {
051 // #####
052 var sel = id_selected();
053
054 if(!sel) { alert("No topic
    available");
055 return;
056 }
057
058 remove_me([sel], 'statusdiv');
059
060 var node =
    document.getElementById(sel);
061 var row =
    node.parentNode.parentNode;
062 row.parentNode.removeChild
    (row);
063 select_first();
064 }
065
066 // #####
067 function select_first() {
068 // #####
069 var form =
    document.getElementById("form"
    );
070 if(! form.r) { return; }
071 if(! form.r.length) {
072 form.r.checked = 1;
073 if(!
    document.getElementById(id_sel
    ected()) {
074 document.getElementById
    ('tarea').value = "";
075 return;
076 }
077 display([id_selected()],
    ['tarea', 'statusdiv']);
078 }
079
080 for(var i = 0; i <
    form.r.length; i++) {
081 form.r[i].checked = 1;
082 break;
083 }
084 display([id_selected()],
    ['tarea', 'statusdiv']);
085 }
086
087 // #####
088 function id_selected() {
089 // #####
090 sel =
    id_selected_first_pass();
091
092 if(!
    document.getElementById(sel) )
    {
093 document.getElementById
    ('tarea').value = "";
094 return;
095 }
096 return sel;
097 }
098
099 // #####
100 function
    id_selected_first_pass() {
101 // #####
102 var form =
    document.getElementById("form"
    );
103 if(! form.r) { return 0; }
104 if(! form.r.length) { return
    form.r.id; }
105
106 for(var i = 0; i <
    form.r.length; i++) {
107 if(form.r[i].checked) {
108 return form.r[i].id;
109 }
110 }
111 alert("Selected ID is
    unknown");
112 return 0;
113 }

```

```

<HTML><HEAD>
<style> body {font-family: arial;} </style>
<SCRIPT LANGUAGE="JavaScript" src="/snip.js"></SCRIPT>
</HEAD><BODY>

<H3>Ajaxed Text Snippet Storage</H3>

<P> <INPUT type="text" id="new_topic" value="">
<A ONCLICK="topic_add(
  document.getElementById('new_topic').value);
  return false;"
  HREF="foo">Add new topic</A>

<BR> <FORM id="form">
  <TABLE id="topics">
    [% FOREACH topic = topics %]
      <TR>
        <TD> <INPUT
          ONCLICK="display(['[% topic %]',
            ['tarea', 'statusdiv'])"
          NAME="r" TYPE="radio" ID="[% topic %]"
          VALUE="[% topic %]">
        </TD>
        <TD> [% topic %] </TD>
      </TR>
    [% END %]
  </TABLE>

  <TEXTAREA NAME="text" ID="tarea" COLS=80 ROWS=10>
</TEXTAREA>

  <BR> <A HREF="foo" ID="update"
    ONCLICK="topic_update(); return false">Update</A>
  &nbsp;&nbsp;&nbsp;<A HREF="foo" ID="remove"
    ONCLICK="topic_remove(); return false">Remove</A>

<P> <INPUT TYPE="text" SIZE=80 NAME="status"
  ID="statusdiv" VALUE="Welcome.">
</FORM>
<SCRIPT LANGUAGE=JavaScript>select_first();</SCRIPT>
</BODY> </HTML>

```

Figura 3: Plantilla HTML que procesa snip usando el conjunto de herramientas de plantillas.

necesitamos definir funciones para las distintas acciones del cliente (eliminar/actualizar/mostrar) y proporcionar una función *show\_html*, que devuelve el HTML de la aplicación cliente en la carga inicial.

## HTML y Perl Separados

El script Perl *snip* toma el HTML a enviar de la plantilla *snip.tmpl*, que se muestra en la Figura 3. El conjunto de herramientas de plantilla cargan la mencionada plantilla y proporcionan un buen número de términos que nos permiten insertar bucles *for* sencillos o condiciones.

Usa [% FOREACH topic = topics %] para iterar sobre el array @topics previamente proporcionado por *snip* con los títulos de los bloques de texto y genera un cierto número de botones radio, cada uno en una tabla de filas separadas. [% topic %] devuelve el valor de la variable de plantilla *topic* cada vez. La propiedad *id* de cada botón de radio se configura con la cadena de texto del título, y el manejador del evento *OnClick* llama a la función *display()* descrita previa-

mente, que existe tanto en la parte cliente JavaScript como en el universo Perl del servidor.

La función *select\_first()* selecciona la primera entrada en la lista de botones radio y hace la petición del bloque de texto para el título correspondiente. Llama primero al método *document.getElementById* para buscar la etiqueta HTML con la ID de *form* (el formulario HTML en *snip.tmpl*). Todos los botones de radio se llaman *r*, por lo que *form.r* debería ser un array que computa las entradas de cada botón de radio encontrado. Si la lista de títulos está vacía, *select\_first()* no selecciona nada, obviamente, ni se comunica con el servidor.

Los hipervínculos de etiquetas tipo *<A >* que se usan en *snip* deberían obtener un *return false* como última acción de sus manejadores *OnClick*. Esto asegura que el navegador ejecuta el código JavaScript asignado al enlace, en lugar de seguir el falso atributo *HREF*.

La plantilla carga la librería *snip.js* en primer lugar (véase Listado 2). La librería proporciona un buen número de funciones que permiten a la interfaz gráfica de usuario que funcione adecuadamente. La función de JavaScript *topic\_add* en *snip.js* espera una cadena del título y añade una nueva entrada con su nombre al final de la tabla de botones radio.

*topic\_remove()* elimina un título de la lista de botones de radio y envía una petición al servidor, que en respuesta borra el fragmento de texto de la caché.

*id\_selected()* genera la propiedad *id* del botón de radio seleccionado, es decir, el título de la entrada que quiere ver el usuario. Si la lista está vacía, Firefox puede confundirse y seguir devolviendo un valor. Para solucionar este fallo, *id\_selected()* vuelve a veri-

ficar el resultado de *id\_selected\_first\_pass()* y devuelve *undefined* si pilla a Firefox haciendo trampas. Si la lista de botones radio tiene dos o más entradas, *form.r.length* devuelve el tamaño de la lista. Si la lista tiene sólo una sola entrada, *form.r.length* devuelve un valor no definido. Si la lista está vacía, *form.r* no está definida. El flag *checked* puede ahora verificarse, bien a través de *form.r.checked*, o bien a través del elemento "i" del array: *form.r[i].checked*. Tras esto, *select\_first()* llama a la función *display()* para obtener desde el servidor el texto del título seleccionado.

## Asperezas

Este script sólo pretende ser un ejemplo. Para conservar la simplicidad se ha dejado fuera del script, el código para tratar las condiciones de error. Podrían también aparecer algunos problemas de compatibilidad con navegadores distintos de Firefox.

## Instalación

El script requiere los módulos *Class::Accessor*, *CGI::Ajax* y *Template* de CPAN. A continuación añadimos el script *snip* (ejecutable!) la plantilla *snip.tmpl* (en el *cgi-bin* del servidor web) y el script de JavaScript *snip.js* directamente bajo la raíz de documentos (usualmente *htdocs*).

Si el terminal utilizado para cortar y pegar no entiende UTF-8, deberíamos marcar como comentario la línea 68 de *snip* para indicarle al servidor web que envíe *charset = iso-8859-1* en la cabecera CGI en su lugar. Y si preferimos no tener la caché de documentos bajo */tmp*, tendremos que especificar nuestro directorio preferido en la línea 9: *my \$cache = Cache::File-Cache->new({cache\_root => "/ruta"});*. Finalmente, apuntamos nuestro navegador *http://server/cgi-bin/snip*, y, suponiendo que tenemos JavaScript activado, la función de fragmentos de texto debería funcionar, comunicándose con el servidor, y manteniendo el repositorio de fragmentos en el servidor actualizado. ■

## RECURSOS

- [1] Listados del artículo: <http://www.linux-magazine.com/Magazine/Downloads/62/Perl>