Prototype y Script.aculo.us

Web

Prototype

Qué es

Prototype es una librería Open Source creada para extender las funcionalidades de JavaScript y reducir la tarea de codificación a los programadores. Tiene la gran ventaja de distribuirse en un solo archivo .js, es muy liviana y en se ha convertido un estándar de facto en el mercado al inicio del lanzamiento de AJAX. Su uso ha caído desde la salida de jQuery al mercado que realiza tareas similares de manera más simple.

Instalación

Su instalación es muy sencilla. Sólo basta con visitar el sitio web www.prototypejs.org y descargar la última versión, que no es más que un archivo prototype.js que se debe incorporar en nuestro proyecto y, luego, incluirlo en un tag script (fig. 1).

En el sitio web se puede consultar toda la documentación de las API que incorpora la librería. A continuación se detallarán los agregados más utilizados.

Utilitarios

$

Dentro de la gama de utilitarios, hay muchos y muy útiles. El primero que se va a analizar es el signo $. Aunque hasta el presente no se usaba para mucho, es posible que una variable (y, por lo tanto, también una función) en JavaScript comience con el signo $, incluso que su nombre sea sólo ese símbolo. Es por eso que Prototype ha creado

una función llamada $ que reemplaza a una de las funciones más utilizadas en una aplicación AJAX: document.getElementById. Es así que con hacer:

```
$("divContenido")
```

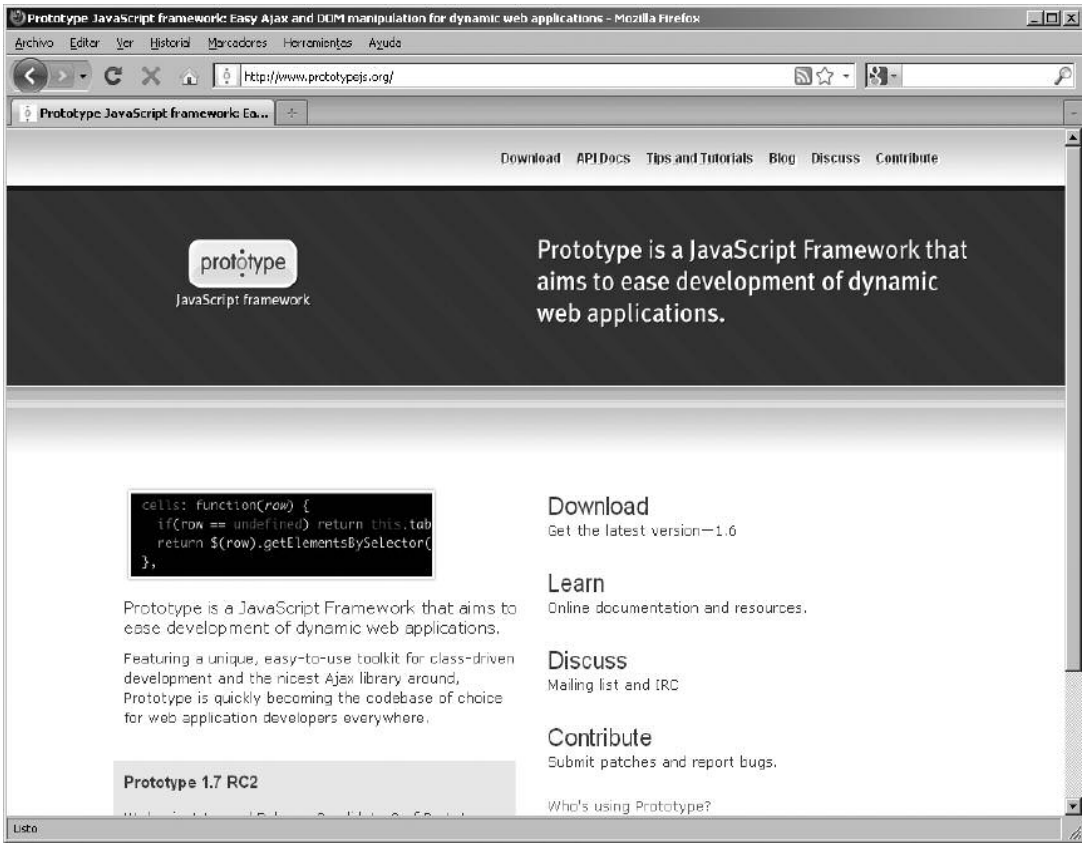

Fig. 1. Sitio web oficial de la librería Prototype.

se estaría recibiendo un objeto que represente al elemento XHTML que tenga id divContenido. Queda muy simple y sencillo de leer, además de la obvia reducción de código a escribir y a transferir desde el servidor al cliente. De esta forma, si se quiere cambiar el color de fondo de una celda que posee id, se podría hacer:

```
$("celda1").bgColor = "red";
```

o si se quiere definir la función onclick de un botón:

```
$("btnEnviar").onclick = function() { alert("Gracias por presio-
narme") };
```

no obstante, la función $ no es sólo una mera traducción de document.getElement ById. La función recibe un número dinámico de parámetros y, cuando se envía más de uno, lo que se obtiene es un vector cuyas posiciones son cada uno de los elementos pedidos. Por ejemplo:

```
$("btnEnviar", "btnCancelar", "btnSalir")
```

Este código devolvería un vector de 3 posiciones con tres objetos de tipo botón, con los ids solicitados.

$$

La función $$ recibe uno o más selectores de tipo CSS (de clase, de etiqueta o de id) y devuelve un vector con todos los elementos que cumplen con el o los selectores dados. Hay combinaciones completas que se pueden hacer para encontrar con facilidad y rapidez elementos HTML que cumplan con alguna condición. Ejemplos:

```
// Devuelve todos los divs
$$("div");

// Devuelve todos los span de clase titulo
$$("span.titulo");

// Devuelve todos los links que abran en ventana nueva
$$("a[target='_blank']");

// Devuelve todos los td y th de la tabla clientes
$$("#clientes td", "#clientes th");
```

$F

Devuelve el valor (la propiedad value) del elemento HTML dado, por lo general una etiqueta de formulario (un textbox, o un select).

```
var nombre = $F("txtNombre");
var país = $F("lstPaises");
```

$w

Copiada del lenguaje Ruby, esta función recibe un string con palabras separadas por espacio y devuelve un vector con cada palabra.

```
var arrayPaises = $w(”Argentina Brasil Chile Uruguay”);
```

Try.these

Esta función recibe una lista de funciones por parámetro y devuelve el resultado de la primera de ellas que no genere una excepción (error). Por ejemplo, se podría mejorar nuestra librería de AJAX cuando se instancia el objeto XMLHttpRequest evitando ifs:

```
function obtenerXHR() {
  return Try.these(
    function() { return new XMLHttpRequest() },
    function() { return new ActiveXObject('Msxml2.XMLHTTP') },
    function() { return new ActiveXObject('Microsoft.XMLHTTP') }
  ) || false;
}
```

Otros

Hay otros utilitarios menos usados, como $R, $A y $H. El primero permite generar un objeto ObjectRange, el segundo, convertir cualquier objeto enumerable a un Array y el último, cualquier objeto a un HashTable (array asociativo). Dejamos para revisar en la documentación oficial de Prototype su uso y ejemplos. $R permite obtener una secuencia de objetos de tipo enumerables, por ejemplo, para recibir una lista con los números 1 a 1000, se puede hacer $R(1, 1000) y las letras del alfabeto $R('a', 'z'). Si se quieren convertir esos objetos a Array, sólo resta aplicarle también $A (fig. 2).

Agregados a objetos

Cuando importamos prototype.js en nuestra página web, de manera instantánea muchos objetos adquieren características nuevas, como propiedades y métodos agregados por la librería.

Strings

Todos los strings ahora poseen, entre otros, los métodos siguientes:

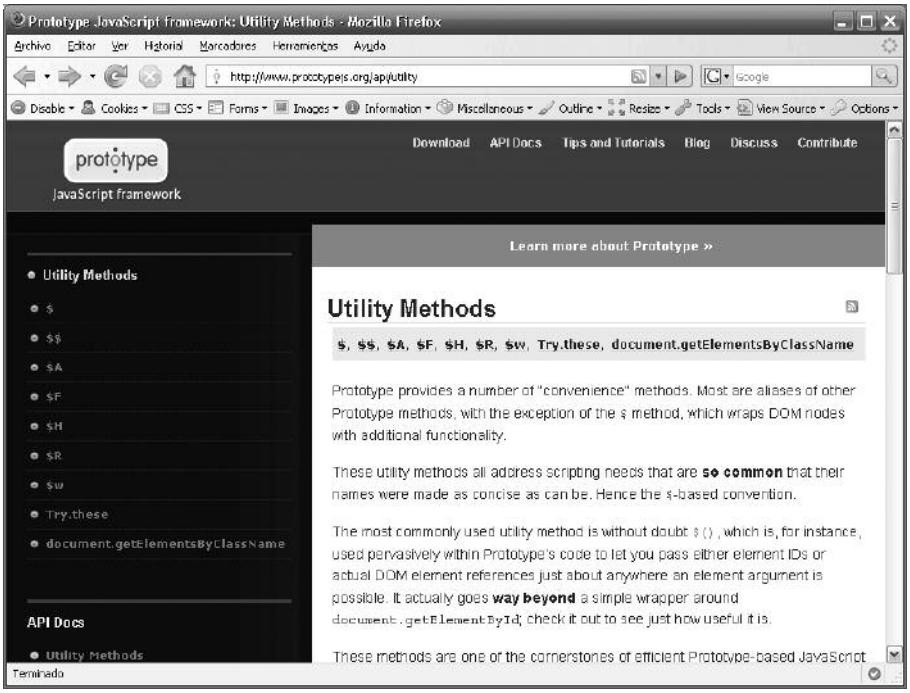

Fig. 2. La documentación de Prototype incluye ejemplos de cada función y está disponible en la web, en PDF y en formato Ayuda de windows.

Método	Descripción
blank	Devuelve true si el string está vacío o sólo contiene espacios
camelize	Devuelve una versión en formato camello del string en cuestión. Por ejemplo: nombre-cliente devuelve nombreCliente
capitalize	Pone la primera letra en mayúscula
desherize	Reemplaza todos los guiones bajos (_) por medios (-)
empty	Devuelve true si el string está vacío por completo (no tiene ni siquiera espacios)
endsWith	Recibe una cadena e indica si el string finaliza con ella. Por ejemplo, es útil para comprobar extensiones de nombre de archivo
escapeHTML	Convierte caracteres especiales a HTML
evalJSON	Evalúa el string como un JSON y devuelve un objeto. Recibe un parámetro lógico opcional e indica si queremos comprobar que no sea código malicioso

Método	Descripción
evalScripts	Ejecuta todos los tag SCRIPT que hubiera en el string
startsWith	Recibe una cadena e indica si el string comienza con ella
strip	Elimina todos los espacios antes y después del texto (equivalente a un Trim en otros lenguajes)
stripScripts	En caso que el string fuera HTML, elimina todos los tags script que hubiera
stripTags	En caso que el string fuera HTML elimina todos los tags (aunque no el texto que tuvieran dentro)
times	Recibe un parámetro numérico y devuelve una versión con el string repetido n veces.
toArray	Convierte el string a un array de caracteres.
toJSON	Convierte el string a un JSON compatible.
toQueryParams	En caso que el string tuviera un formato estilo QueryString (clave=valor&clave2=valor2) devuelve un objeto con las propiedades clave y clave2.
truncate	Recibe una cantidad y un sufijo opcional. Trunca el string a la cantidad dada e incorpora el sufijo al final. Por defecto el sufijo es "...". Permite mostrar textos largos en espacios reducidos.
underscore	Convierte un string en notación camello a separado por guiones bajos.
unescapeHTML	Devuelve una versión en texto normal de un string con caracteres en formato Escape HTML.

También aparece una nueva clase en JavaScript llamada Template, que permite definir cierta plantilla y un objeto estilo JSON, así como reemplazar cada propiedad del objeto en la plantilla en las ocurrencias que se indiquen. Por ejemplo:

```
// ésta es la plantilla
var plantilla = new Template('El curso de #{curso} es dictado por
#{profesor}.');

// los datos a reemplazar
var cursoAjax = {
curso: 'AJAX y JS Avanzado',
profesor: 'Maximiliano R. Firtman'
};
// aplica la plantilla
alert(plantilla.evaluate(cursoAjax));
```

Arrays

Todos los arrays reciben los métodos siguientes. Cuando el objeto que tenemos no es estrictamente un array, sino una colección (p. ej., document.getElementsByTagName) podemos convertirlo a array en forma explícita para tener estos métodos utilizando $A.

Método	Descripción
clear	Limpia todo el array.
clone	Devuelve una copia exacta del array.
compact	Devuelve una versión comprimida del array, eliminando todas las posiciones nulas o indefinidas.
each	Ejecuta una función recibida por parámetro en cada posición del array. Equivale a un for que recorra todo el array y ejecute la función a cada elemento.
first	Devuelve el primer elemento.
flatten	En el caso de un vector multidimensional (p. ej., una matriz), lo convierte a una sola dimensión.
indexOf	Devuelve la posición de la primera ocurrencia de un elemento recibido por parámetro, o -1 si no existe.
last	Devuelve el último elemento.
reduce	Si el array contiene una sola posición, devuelve el único elemento.
reverse	Invierte la posición de los elementos.
toJSON	Convierte el vector a un string estilo JSON.
without	Recibe una lista de valores y devuelve el vector, pero sin los valores recibidos.

Si trabajamos con Prototype, debemos dejar de utilizar for in para recorrer los vectores. Sin entrar en demasiados detalles técnicos, los métodos que Prototype tienen problemas con el for in. La solución es cambiar el código siguiente:

```
for (var i in vector) {
      alert(vector[i]);
}
```

por el que sigue:

```
vector.each(function(item) {
      alert(item)
}
```

También aparecen métodos nuevos que se pueden aplicar a todos los elementos que sean enumerables, que los podemos ver en la documentación. Algunos de estos métodos son en realidad muy útiles.

Elementos DOM

Cuando se acceda a un elemento DOM por medio de las funciones propias de Prototype, como $ o $$, entonces se estará accediendo a un elemento nodo extendido, que posee más propiedades y métodos.

Event

Prototype incorpora a JavaScript un manejo de eventos algo superior al clásico que trae el estándar del lenguaje. Con Prototype se puede hacer uso del patrón de diseño Observer para agregar o eliminar un observador a un evento de un elemento HTML. Esto permite una forma más sencilla de agregar y eliminar funciones que escuchan a que cierto evento ocurra.

Para ello, hay una clase Event que posee los métodos observe y stopObserving. Veamos un ejemplo:

```
Event.observe(window, 'load', function() {
  Event.observe('btnEnviar', 'click', mostrarMensaje);
});

function mostrarMensaje() {
      alert("Gracias");
      // Dejamos de "observar" el click del botón
      Event.stopObserving('btnEnviar', 'click', mostrarMensaje);
      // El próximo click no ejecutará esta función
}
```

Form

Si a un formulario HTML le incluimos un ID (p. ej., "form1") y lo accedemos a través de $("form1"), tendremos los métodos siguientes:

Método	Descripción
disable	Desactiva de manera funcional todos los elementos que están dentro del formulario.
enable	Vuelve a activar en forma funcional todos los elementos.
findFirstElement	Devuelve el primer control que esté activado y no sea de tipo hidden.
focusFirstElement	Pone el foco en el primer control del formulario.
getElements	Devuelve una colección con todos los controles de ingreso del formulario.
getInputs	Devuelve una colección con todos los controles INPUT. Es posible restringir cuáles queremos mediante los parámetros.
reset	Limpia todo el formulario.
serialize	Devuelve todos los controles en formato string URL para enviarlos en una petición AJAX vía GET o POST.
serializeElements	Devuelve una serie de controles enviados por parámetro en formato string URL.

Así, si se desea convertir un clásico formulario a un formulario AJAX, sólo se debe realizar la petición al servidor, enviando los parámetros recibidos del método serialize. Además, se debe cambiar el botón de Enviar de tipo submit por uno de tipo button, si no lo estaremos enviando al viejo estilo (fig. 3). Veamos un ejemplo:

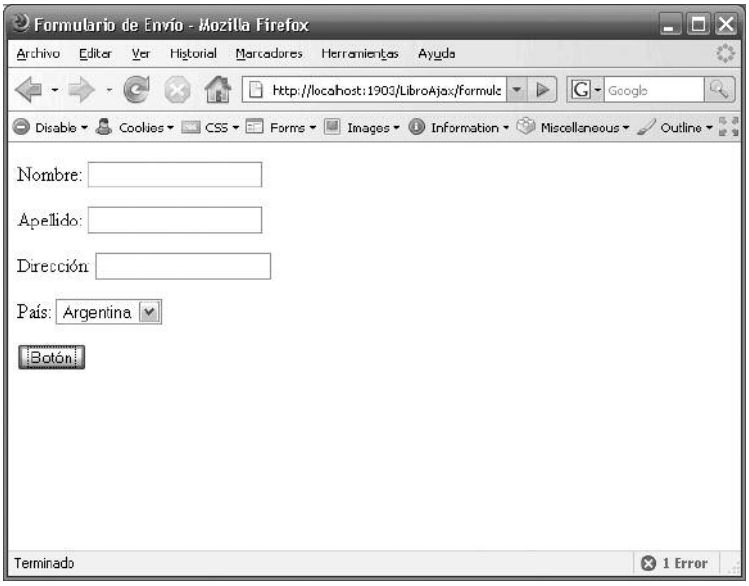

Fig. 3. Se puede transformar cualquier formulario para que envíe los datos vía AJAX sin refrescar la página.

formulario.html

```
<!DOCTYPE html PUBLIC ”-//W3C//DTD XHTML 1.0 Transitional//EN”
”http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd”>
<html xmlns=”http://www.w3.org/1999/xhtml”>

<head>
   <meta http-equiv=”Content-Type” content=”text/html; charset=iso-
8859-1” />
   <title>Formulario de Envío</title>
   <script type=”text/javascript” src=”prototype.js”></script>
   <script type=”text/javascript” src=”ajax.js”></script>
   <script type=”text/javascript” src=”formulario.js”></script>
</head>

<body>
<form id=”form1” name=”form1”>
  <p>Nombre:
    <input name=”txtNombre” type=”text” id=”txtNombre” />
  </p>

  <p>Apellido:
    <input name=”txtNombre2” type=”text” id=”txtApellido” />
  </p>

  <p>Direcci&oacute;n:
    <input name=”txtNombre3” type=”text” id=”txtDireccion” />
  </p>

  <p>Pa&iacute;s:
    <select name=”lstPaises” id=”lstPaises”>
      <option value=”AR”>Argentina</option>
      <option value=”ES”>Espa&ntilde;a</option>
      <option value=”MX”>M&eacute;xico</option>
    </select>
  </p>

  <p>
    <input type=”button” onclick=”enviarPost()” value=”Bot&oacute;n”
/>
  </p>
</form>

</body>
</html>
```

formulario.js

```
function enviarPost() {
   var peticion = obtenerXHR();
   peticion.open(”POST”, ”recibirDatos.php”, true);
   peticion.onreadystatechange = procesarPeticion;

   // Definimos cabecera obligatoria para enviar POST
   peticion.setRequestHeader(”Content-Type”, ”application/x-www-form-
urlencoded”);

   // Enviamos los parámetros del formulario a través de Prototype
   var parametros = $(”form1”).serialize();
   peticion.send(parametros);
}

function procesarPeticion() {
    if (peticion.readyState==4) {
            // La petición terminó
            if (peticion.status==200) {
                  // Mostramos el texto en una alerta
                  alert(peticion.responseText);
            }
    }
}
```

Objeto AJAX

Prototype provee su propio mecanismo para realizar peticiones AJAX que nos abstrae de los problemas de incompatibilidad entre browsers, entre otras cosas.

Para ello, provee los objetos siguientes que se deben instanciar:

Ajax.Request(url, opciones)

Permite realizar una petición AJAX a la url indicada, definiendo distintos parámetros opcionales para partir del segundo parámetro, que se envía en formato JSON.

Ajax.Updater(elemento, url, opciones)

Realiza una petición AJAX, y el contenido recibido como texto HTML lo utiliza como contenido para reemplazar en el elemento DOM enviado por parámetro (p. ej., un DIV).

Ajax.PeriodicalUpdater(elemento, url, opciones)

Realiza una petición AJAX periódicamente y, sin mediar ningún otro código reemplaza el contenido del elemento recibido por parámetro (p. ej., un DIV) con el texto HTML recibido por la url en la petición.

Además, por medio de Ajax.Responders es posible registrar funciones que se ejecutarán cada vez que una petición AJAX de Prototype pase por cierto estado.

En todos los métodos expuestos las opciones en formato JSON pueden ser:

Opción	Valor por defecto	Descripción
asynchronous	true	Define si se usa una petición asincrónica o sincrónica
contentType	'application/x-www-form-urlencoded'	Define la cabecera que se enviará al servidor con el tipo de contenido que estamos enviando
encoding	'UTF-8'	La codificación de la petición
method	'post'	Puede ser get o post
parameters	''	Recibe los parámetros a enviar en la petición. Se puede enviar un string tipo URL, o un objeto (p. ej., un JSON)
postBody	null	En caso que se envíe vía post se puede definir aquí el cuerpo de la petición enviada
requestHeaders	Headers por defecto	Se puede enviar un objeto (p. ej., JSON) o un vector con los headers que se quieren enviar

También los tres tipos de peticiones en las opciones permiten definirle los eventos siguientes que podrán ser capturados por una función:

Propiedad	Descripción
onComplete	Se ejecuta cuando la petición termina
onException	Se ejecuta cuando se produce cualquier tipo de error al definir la petición
onFailure	Se ejecuta cuando hubo algún error en el servidor (status != 200)
onSuccess	Se ejecuta cuando la petición termina bien (equivaldría a cuando readyState==4 y status==200). Este evento recibe la respuesta recibida por parámetro (que tiene las propiedades responseText y responseXml)
on999	Siendo 999 algún código HTTP (p. ej., on404). Se ejecuta cuando ese código HTTP se produce en la petición

Por ejemplo, para enviar una petición al servidor y actualizar un DIV con el contenido de un archivo html, el código sería el siguiente:

```
new Ajax.Updater('divContenido', 'contenido.html', {
  onFailure: { alert('Ocurrió un error') },
  method: 'get'
});
```

Si se desea reemplazar nuestro ejemplo anterior de un formulario POST utilizando el objeto Ajax de Prototype se cambiaría el formulario.js por el siguiente:

```
function enviarPost() {
   new Ajax.Request("recibirDatos.php", {
       onSuccess: function(respuesta) {
            // En respuesta tenemos la devolución del servidor
            alert(respuesta.responseText);
       }
       parameters: $("form1").serialize();
   }
}
```

Como se puede apreciar, el código para hacer la petición AJAX se simplifica mucho. No obstante, siempre se debe recordar que igualmente la librería está haciendo lo mismo que hacíamos nosotros antes. De esta forma, ya no sería necesario utilizar nuestro ajax.js, aunque todavía no nos desharemos de él.

Incluso si se envía desde el servidor un encabezado que indique algún tipo MIME de tipo JSON (p. ej., application/javascript), Prototype convertirá en forma automática la respuesta a JSON y se recibirá como segundo parámetro del evento onSuccess (fig. 3-10).

Otros agregados

Prototype también agrega comportamiento a los objetos siguientes:

- **Class**. Aparece un método Class.create que devuelve una función al estilo definición de clases en el lenguaje Ruby.
- **Date**. Aparece el método toJSON que convierte la fecha.
- **Enumerable**. Aparecen varios métodos nuevos en todos los objetos que sean enumerables (colecciones).
- **Function**. Aparecen los métodos bind y bindAsEventListener, que permiten enlazar las funciones cuando hay problemas con el ámbito (scope) de éstas.

- **Hash**. Permite trabajar de mejor forma cuando se trata con objetos de tipo Hash (o vectores asociativos).
- **Number**. Contiene algunas funciones útiles para números, como times –que ejecuta una función n cantidad de veces– o toPaddedString –que devuelve el número con ceros a la izquierda–.
- **Object**. La librería agrega métodos a todos los objetos, entre ellos extend –que permite heredar del mismo– o toJSON –que lo convierte a un string con notación JSON–.
- **PeriodicalExecuter**. Permite ejecutar una función en forma periódica. Lo que hace es encapsular el funcionamiento de setInterval y clearInterval propios de JavaScript.

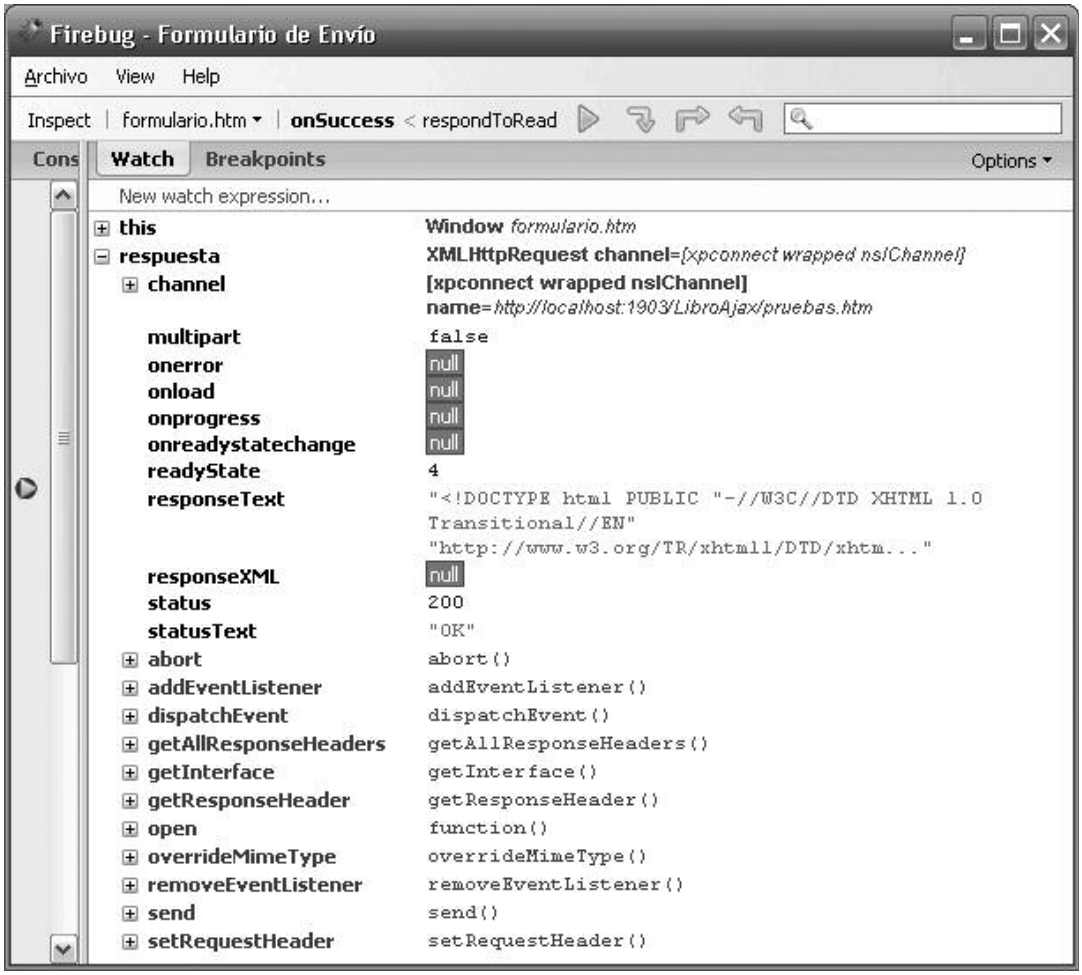

Fig. 4. Objeto respuesta recibido en el evento onSucess. Cada evento tiene sus propios atributos.

Script.aculo.us

Qué es

Script.aculo.us es una librería gratuita para la creación de aplicaciones ricas de Internet utilizando JavaScript, que necesita Prototype para ejecutar su funcionalidad. Es una de las librerías más sencillas y más utilizadas para crear controles ricos (fig. 5).

Dentro de las funcionalidades que agrega se pueden destacar:

- Framework de animación
- Soporte de Drag and Drop
- Controles Ajax (como Autocomplete)
- Más utilidades DOM, además de las de Prototype
- Testing de Unidad

Fig. 5. El sitio web de Script.aculo.us es un buen punto de inicio para encontrar ejemplos y documentación.

Instalación

Desde el sitio web de la librería (http://script.aculo.us) se puede descargar la última versión en formato ZIP. Dentro de este archivo comprimido en la carpeta lib se encontrará la última versión de Prototype y en la carpeta src, los archivos .js de la librería.

Esta librería presenta 7 archivos .js partiendo como base de scriptaculous.js. Este mismo JavaScript es el que carga el resto de los scripts.

Se deben instalar todos los js en la misma carpeta donde se tendrán los HTML y los JavaScript, y si se desea insertarlos en otra subcarpeta, se debe modificar prototype.js con el camino relativo hasta llegar a ellos.

Para la utilización primero es necesario que se incluya Prototype, por ejemplo:

```
<script src="javascripts/prototype.js" type="text/javascript"></script>
<script src="javascripts/scriptaculous.js"
type="text/javascript"></script>
```

Si se desean limitar las funcionalidades disponibles y así reducir la cantidad de JavaScript que se cargarán, se puede utilizar la sintaxis siguiente:

```
<script src="scriptaculous.js?load=lista"
type="text/javascript"></script>
```

donde lista es una secuencia separada por comas de alguno de los módulos siguientes: builder, effects, dragdrop, controls, slider.

Efectos visuales

Uno de los agregados más potentes en Script.aculo.us es la posibilidad de generar efectos visuales muy atractivos y con muy poco código. Para ello, sólo hace falta utilizar la sintaxis siguiente:

```
new Effect.NombreEfecto(id_elemento, opcionales);
```

De esta forma, se puede aplicar un efecto a un DIV, o a cualquier otro elemento del DOM. Cuando se instancie el efecto, éste se generará en pantalla (fig. 6).

Los efectos disponibles se dividen en:

- **Efectos de núcleo**: contiene efectos de base.
- **Efectos combinados**: contiene efectos visuales producto de la combinación de efectos de núcleo.

- **Librería de efectos**: es una lista de efectos generados a partir de los anteriores por la comunidad de usuarios disponibles en wiki.script.aculo.us/scriptaculous/show/EffectsTreasureChest.

Efectos de núcleo

Los Core Effects o efectos de núcleo disponibles son los siguientes:

- **Opacity**: cambia la opacidad (transparencia) del elemento.
- **Scale**: cambia el tamaño del elemento.
- **Morph**: cambia una propiedad CSS del elemento.
- **Move**: mueve el elemento en la página.
- **Highlight**: genera una animación de resaltado del fondo del elemento.
- **Parallel**: ejecuta un conjunto de efectos en paralelo.

Fig. 6. La API de efectos de Script.aculo.us es muy completa y en la web encontraremos ejemplos de cada uno.

Los parámetros opcionales se ingresan en formato de objeto JSON y los que se comparten entre todos los efectos de núcleo son:

Propiedad	Valor por defecto	Descripción
duration	1	Duración del efecto en segundos, es posible utilizar valores reales, por ejemplo: 2, 0.5
fps	25	Cantidad de cuadros refrescados por segundo
transition		Define una función que modifica el punto de animación
from	0	Valor entre 0 y 1 que define el punto de origen del efecto
to	1	Valor entre 0 y 1 que define el punto destino del efecto
queue		Permite definir, en modo avanzado, dónde aplicar el efecto en una cola de efectos
delay	0	Segundos de espera desde ahora antes del comienzo de la animación

En la documentación de la librería es factible encontrar sobre qué elementos se puede aplicar cada efecto para tener compatibilidad con todos los browsers. En general, a los DIV se les pueden aplicar todos los efectos.

Algunas propiedades específicas de cada efecto son:

Efecto	Propiedad	Descripción
Scale	scaleX	Valor lógico que define si se debe escalar en sentido horizontal.
Scale	scaleY	Valor lógico que define si se debe escalar en sentido vertical.
Scale	scaleContent	Define si también se debe escalar el contenido del elemento.
Scale	scaleFromCenter	Define si se escala desde el centro.
Scale	scaleMode	Puede ser 'box' o 'contents'.
Morph	style	Define un objeto JSON con los valores CSS nuevos que debe tener el elemento.
Move	x	Especifica la coordenada *x* del destino.
Move	y	Especifica la coordenada *y* del destino.
Move	mode	Puede ser 'absolute' (considera píxeles reales) o 'relative' (mueve el elemento desde donde está n cantidad de píxeles).
Highlight	startcolor	Color inicial del efecto.
Highlight	endcolor	Color final del efecto.
highlight	restorecolor	Valor lógico que indica si se debe devolver el color original al elemento.

Así, por ejemplo, con el código siguiente se puede crear un efecto que apaga de a poco un botón al ser presionado:

```
<input type="button" value="Apagar" id="btnApagar" onclick="new Ef-
fect.Opacity('btnApagar', {from: 1, to: 0, duration: 2})" />
```

Las transiciones permiten modificar el modo en que se aplica el efecto. Es recomendable probarlas todas para entender sus diferencias. Las opciones disponibles (mediante el atributo transition) son:

- Effect.Transitions.sinoidal (valor por defecto)
- Effect.Transitions.linear
- Effect.Transitions.reverse
- Effect.Transitions.wobble
- Effect.Transitions.flicker

El único distinto es Parallel, que recibe un vector de efectos que se deben aplicar, por ejemplo:

```
new Effect.Parallel( [ new Effect.Move('btn1', {x: 0, y:0}),
new Effect.Opacity('btn1') ] );
```

Por último, se expondrán las propiedades de tipo función (Eventos) disponibles para administrar mejor el efecto animado:

Evento	Descripción
beforeStart	Se ejecuta antes de comenzar el efecto.
beforeUpdate	Se ejecuta antes de que se realice cada redibujo del efecto (depende de la cantidad. de frames por segundo –fps– y de la duración).
afterUpdate	Se ejecuta después de actualizar cada redibujo.
afterFinish	Se ejecuta al finalizar toda la animación.

En todas estas funciones se recibirá un objeto por parámetro, que contendrá propiedades útiles para saber el estado del efecto, entre ellas element (referencia el elemento que se anima), currentFrame (el cuadro actualmente dibujado), startOn (milisegundos pasados desde el inicio) y finishOn (milisegundos restantes).

Efectos combinados

Los efectos combinados ofrecen una funcionalidad mayor y más simple, así como más atractivos. Los disponibles son:

- Appear
- Fade
- Puff
- DropOut
- Shake
- SwithOff
- BlindDown
- BlindUp
- SlideDown
- SlideDown
- SlideUp
- Pulsate
- Squish
- Fold
- Grow
- Shrink

En wiki.script.aculo.us/scriptaculous/show/CombinationEffects se encontrará un detalle y una descripción de cada uno, junto con las opciones adicionales de las que se pueden disponer en cada uno de ellos.

Toggle

Hay un efecto especial llamado Toggle que permite prender o apagar un elemento con rapidez (según su estado actual), aplicando algún tipo de efecto entre appear, blind o slide.

Por ejemplo:

```
<a href="javascript:new Effect.Toogle('divContenido',
'blind')">Titulo</a>

<div id="divContenido">Contenido</div>
```

Ejemplos

Veamos un ejemplo en el que al presionar un botón aparece en pantalla una especie de ventana que estaba oculta, que tiene una opción para cerrarse. Las operaciones de abrir y cerrar se realizan con una animación, y al finalizar el cierre aparece una alerta (figs. 7 a 10).

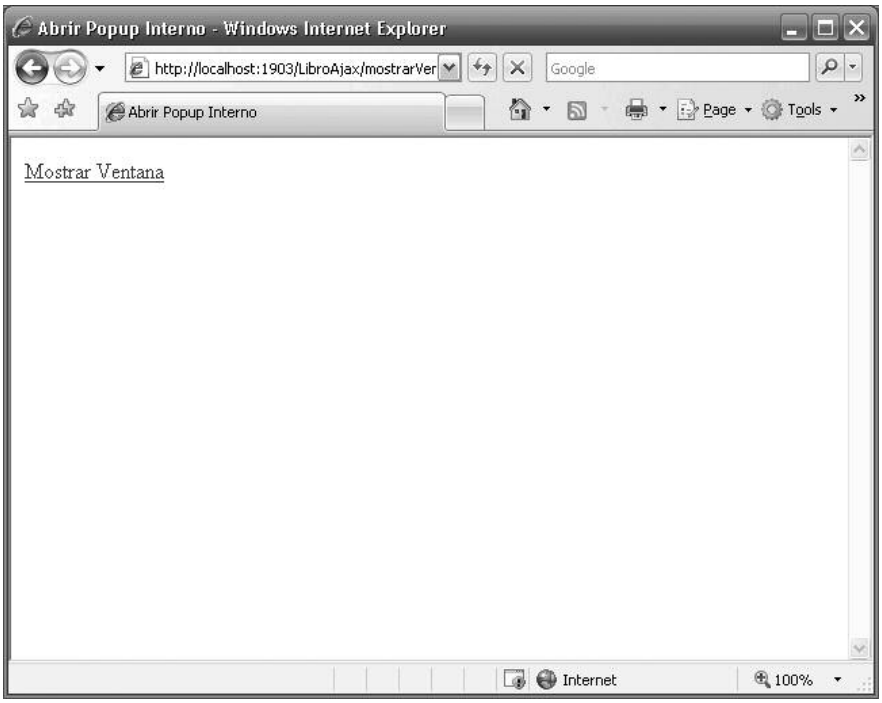

Fig. 7. Nuestra página web recién abierta. Nótese que el divVentana no se muestra.

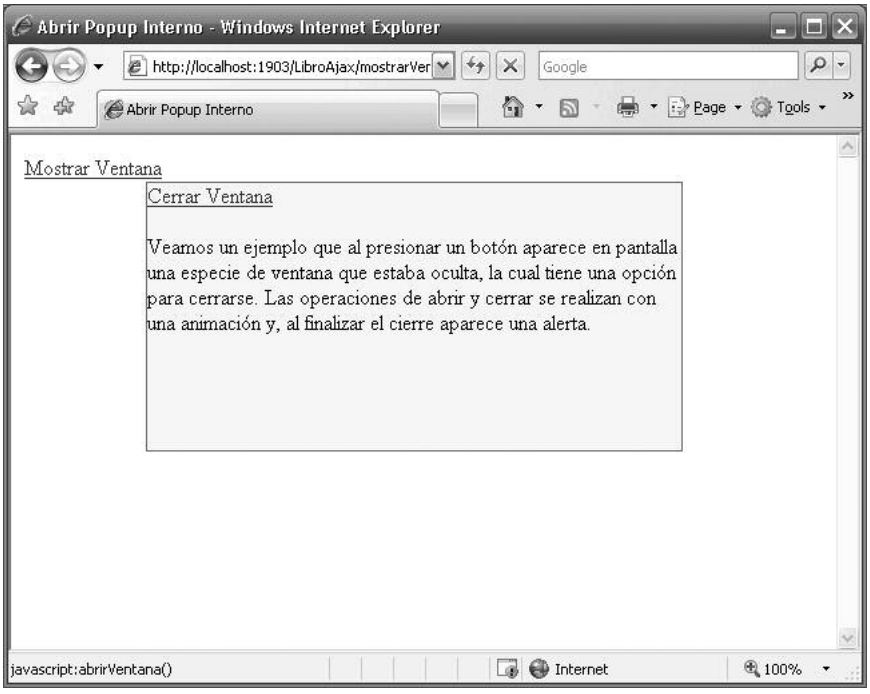

Fig. 8. Al utilizar el link “Abrir Ventana”, ésta se abre con un efecto de transición y el resultado es éste.

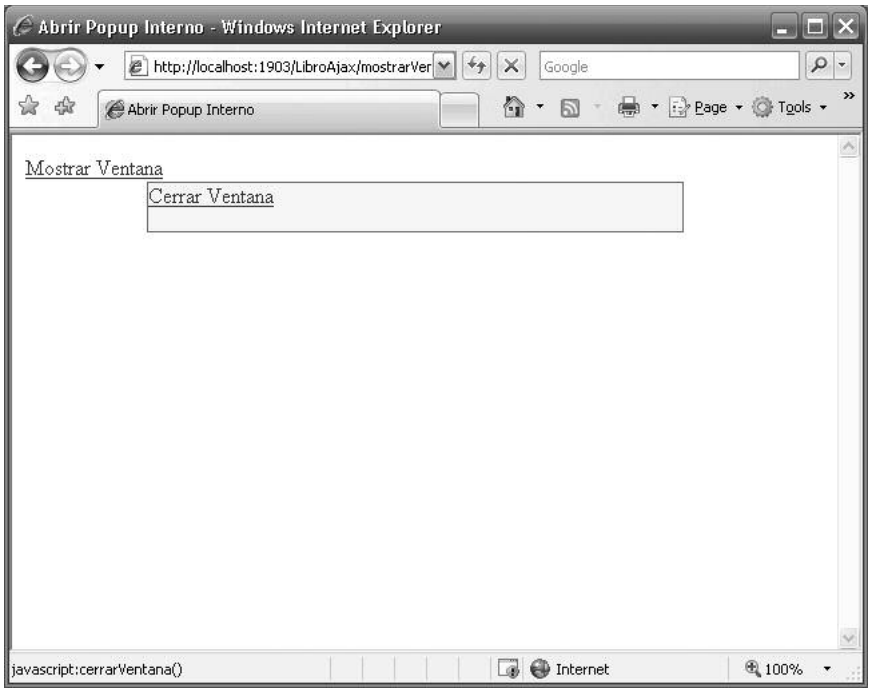

Fig. 9. Al cerrar se ve cómo se aplica un efecto de cierre y se abre una alerta al terminar el efecto.

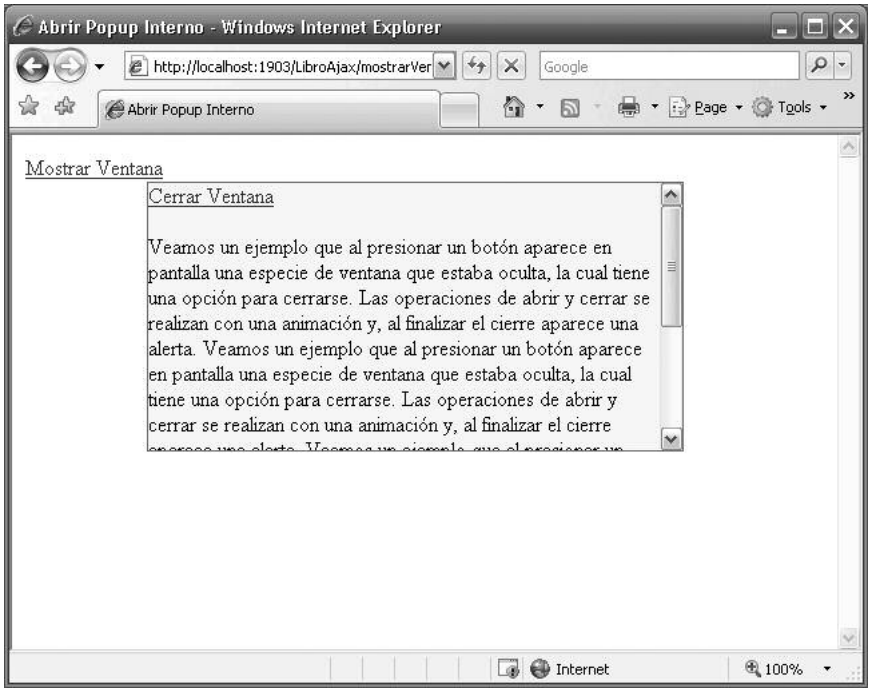

Fig. 10. Utilizar overflow:auto permite que un div con ancho y alto definidos tenga su propia barra de desplazamiento si el contenido no cabe.

mostrarVentana.html

```
<!DOCTYPE html PUBLIC ”-//W3C//DTD HTML 4.01 Transitional//EN”
”http://www.w3.org/TR/html4/loose.dtd”>
<html>
<head>
      <meta http-equiv=”Content-Type” content=”text/html;
      charset=ISO-8859-1”>
      <title>Abrir Popup Interno</title>
      <script type=”text/javascript” src=”prototype.js”></script>
      <script type=”text/javascript”
src=”scriptaculous.js?load=effects”></script>
      <script type=”text/javascript” src=”mostrarVentana.js”></script>
      <link rel=”stylesheet” type=”text/css” href=”mostrarVentana.css”
/> </head>
<body>

      <a id=”lnkMostrar”>Mostrar Ventana</a>

      <div id=”divVentana”>
            <a id=”lnkCerrar”>Cerrar Ventana</a>
            <br />
            <br />
```

mostrarVentana.css

```
#divVentana {
      width: 400px;
      height: 200px;
      position: absolute;
      top: 20px;
      right: 100px;
      background-color: yellow;
      border: 1px red solid;
      margin: 15px;
      overflow: auto;
}
```

mostrarVentana.js

```
        window.onload = function() {
        $("divVentana").hide();
        $("lnkMostrar").href="javascript:abrirVentana()";
        $("lnkCerrar").href="javascript:cerrarVentana()";
}

/** Abre una ventana (div) con un efecto de aparición
 *
 */
function abrirVentana() {
        new Effect.Appear($("divVentana"));
}

/** Cierra la ventana (div) con un efecto
 *  y muestra una alerta cuando éste termina
 */
function cerrarVentana() {
        new Effect.Fold($("divVentana"), {
              afterFinish: function() {
                    alert("Se terminó de cerrar la ventana");
              }
        });
}
```

Builder

Además del framework de efectos, Script.aculo.us agrega soporte de arrastrar y soltar (drag and drop), autocomplete, testing y otros controles que se verán más adelante en el libro.

Lo que se puede analizar ahora es un utilitario DOM adicional conocido como Builder.

Builder.node es una metodología rápida y sencilla para crear objetos de tipo DOM (tablas, divs, etc.) y definir sus propiedades con rapidez en cambio de utilizar document.createElement.

Su sintaxis es la siguiente:

```
var nodo = Builder.node("etiqueta");
var nodo = Builder.node("etiqueta", atributos);
var nodo = Builder.node("etiqueta", arrayHijos);
var nodo = Builder.node("etiqueta", atributos, arrayHijos);
```

En la versión más simple es el equivalente a document.createElement.

Los atributos se definen en un formato de objeto JSON, por ejemplo:

```
var tabla = Builder.node('table', {width:'100%', cellpadding:'3',
cellspacing:'0'});
```

Los nombres de las propiedades del JSON equivalen a los de HTML, salvo class, que se utiliza className y for, que se utiliza htmlFor.

La variable tabla ahora se puede usar para anexarla a otro element DOM, por ejemplo:

```
$("divContenido").appendChild(tabla);
```

El parámetro arrayHijos es un vector de otros nodos que se insertarán como hijos del elemento que se está creando. Por ejemplo, se crea un DIV con un link dentro:

```
$("div1").appendChild(Builder.node("div",
[ Builder.node("a",{id: "link1"})]);
```