

Programar páginas Web con AJAX

MÁS RÁPIDO, MÁS ALTO, MÁS LEJOS



AJAX ofrece un método rápido y eficiente para generar páginas Web interactivas. Os mostramos cómo podemos invocar sus poderes en nuestras propias creaciones Web. **POR OLIVER FROMMEL**

La Web 2.0 no se limita únicamente a la parte de usuario. Algunas importantes y novedosas herramientas ayudan a crear páginas Web más rápidas y más eficientes. Una de las características más destacadas de la nueva Internet es la tecnología conocida como AJAX.

En la vieja Internet, al rellenar un formulario con varios campos y enviarlo al servidor, debíamos aguardar mientras éste evaluaba los datos y respondía con una nueva página (véase la Figura 1). Una página Web basada en AJAX proporciona una solución mucho más elegante. En lugar de volver a pedir la página entera, el navegador sólo pide un fragmento de la misma. El usuario continúa trabajando mientras el servidor da cuenta de la petición, y el navegador continúa mezclando la información con la página existente. De cara al usuario, la página Web es casi tan rápida como una aplicación de escritorio.

El nombre AJAX fue acuñado por Jesse James Garrett [1] en su ensayo "AJAX: Un nuevo camino para las aplicaciones Web". Aunque Garrett sostiene que AJAX no es un acrónimo, la mayoría de la gente lo ha adoptado como abreviación de Javascript Asíncrono y XML. Las páginas Web hechas con AJAX no se generan desde archivos HTML estáticos (y hojas de estilo CSS), sino que se componen de código Javascript que se ejecuta cuando el usuario pulsa sobre un enlace o dispara cualquier otro tipo de evento. Las funciones de Javascript piden información al servidor, cuya respuesta viene en formato XML (lo que explica la X) y HTML, junto con otros formatos.

Este artículo ofrece un vistazo que puede ser muy útil si queremos que AJAX comience a funcionar en nuestra propia página Web.

Asíncrono

Debido a que la petición de AJAX es asíncrona (esto explica la A), el usuario puede continuar interactuando con otras partes de la página HTML sin bloquear el navegador. Una petición asíncrona independiza la petición de la respuesta. Una vez que el código Javascript del navegador ha formulado una petición, simplemente continúa ejecutándose. Cuando la respuesta llega al navegador, llama a una función específica de Javascript, que a cambio mezcla los datos recibidos con la página HTML ya existente. El resultado puede ser tan sencillo como un único valor numérico en una tabla HTML, información estructurada en XML o JSON (Javascript Object Notation), o un valor de un formulario.

Esta técnica permite a Google Mail la capacidad de usar el espacio de trabajo en el centro del navegador para visualizar un editor donde el usuario puede componer un correo, visualizarlo o mostrar una lista de correos (véase la Figura 2). Otros excelentes ejemplos de AJAX son las aplicaciones Web 2.0 Flickr y Del.icio.us.

Guía Rápida

Para comenzar, vamos a asomarnos a una aplicación Web muy sencilla. Cuando el usuario pulsa, queremos que el navegador descargue un número aleatorio desde el servidor y lo muestre en una posición predefinida de la página Web.

La aplicación comprende tres componentes: la página de portada en HTML estático, el código Javascript y un script en el lado servidor que devuelve el resultado (el número aleatorio). Una cosa que vamos a necesitar necesariamente es un servidor Web (generalmente Apache) que soporte un lenguaje de script, que en nuestro caso va a ser PHP. Sin embargo, para probar AJAX podemos ejecutar el servidor y el navegador en la misma máquina (cualquier instalación de escritorio nos valdrá).

Si la instalación y configuración de Apache y PHP con las herramientas estándar de nuestra distribución es demasiado compleja, sólo tenemos que elegir uno de los populares paquetes LAMP o XAMPP. Estos paquetes incluyen Apache y PHP, junto con la base de datos MySQL, que siendo estrictos no necesitaríamos, pero que puede ser de utilidad para las aplicaciones dinámicas en AJAX.

El archivo HTML en la aplicación de ejemplo en AJAX tiene una sencilla estructura (véase el Listado 1). La etiqueta script en la sección de cabecera apunta a un archivo Javascript externo, denominado *ajax.js*, que contiene el código de la aplicación AJAX. Como alternativa, podemos insertar funciones Javascript entre las etiquetas de apertura y cierre en el cuerpo del archivo HTML.

La línea 6 del Listado 1 enlaza el código y el HTML. El atributo de enlace *onclick* guarda el nombre de la función Javascript llamada por el navegador (*getRandom*) cuando el usuario lo pulsa. Esto va seguido de un *span* HTML con una ID, a la cual referirá Javascript posteriormente.

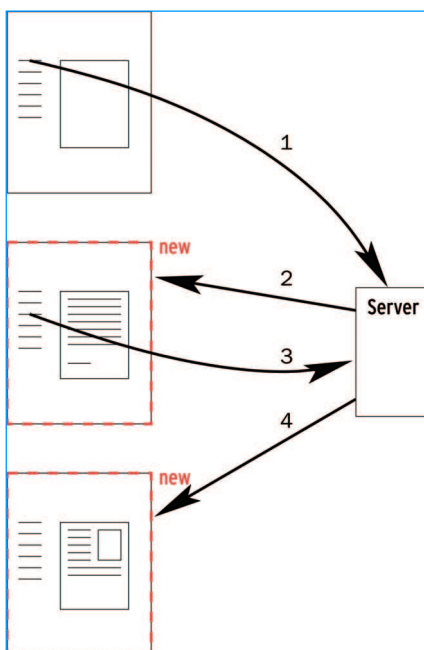


Figura 1: El método antiguo: el navegador vuelve a cargar la página completamente cada vez que ocurre un cambio.

Si queremos verificar que la conexión HTML/Javascript está funcionando sólo tenemos que añadir el siguiente fragmento de código al archivo:

```
function getRandom()
{
  alert("Clicked");
}
```

Suponiendo que *index.html* y *ajax.js* se ubican en el mismo directorio del servidor, *ajaxtest* por ejemplo, la URL para la página HTML sería *http://Nombreservidor/ajaxtest/*. Si el servidor y el navegador se ejecutan en la misma máquina usaremos *localhost* como nombre del servidor. Ahora, cuando pulsamos el enlace, el navegador muestra un cuadro en el que se visualiza un mensaje de texto de *Clicked*.

Si esto no funciona, al menos practicaremos cómo solucionar determinados problemas. El navegador Firefox es una buena elección para el desarrollo con AJAX, ya que incluye un conjunto de prácticas herramientas. Por ejemplo, el menú *Tools* tiene un apartado *JavaScript Console* que lanza una pequeña ventana para los mensajes de error de Javascript.

Una aplicación tan sencilla como esta no tiene muchas fuentes potenciales de error. El servidor puede que no sea capaz de localizar el archivo Javascript, o puede haber algún error de sintaxis.

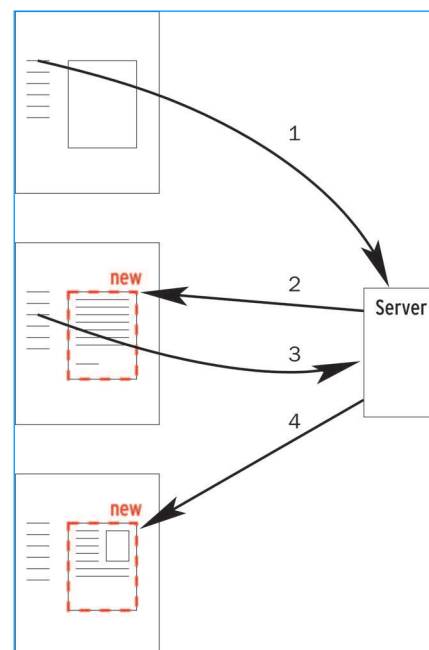


Figura 2: El modelo AJAX: las peticiones sólo recargan las partes de la página que han cambiado realmente.

Conectando con el Servidor

Hasta el momento, el navegador ha enviado una petición para la página Web y otra para el archivo Javascript hacia el servidor. Con esto se completa la comunicación. Éstas no contactan entre sí cuando un usuario pulsa un enlace. Para recabar más información del servidor, el código Javascript ha de crear previamente un objeto de petición. Este proceso es controlado por *XMLHttpRequest()*, de ahí la X de AJAX. El siguiente comando asigna el nuevo objeto de petición a la variable *request*:

```
request = new XMLHttpRequest();
```

El objeto pedido posee varios métodos que son importantes para comunicarse con el servidor más tarde. En primer lugar, *request.open()* configura los parámetros de conexión.

El primer parámetro especifica el método HTTP (GET o POST). Le sigue la dirección Web de contacto (URL). El siguiente parámetro es para el acceso asíncrono, con el valor *true* en nuestro caso. Dos parámetros opcionales pueden pasar el nombre de usuario y la contraseña para las páginas protegidas por contraseña. Suponiendo la siguiente dirección para la variable *url* de Javascript,

`http://localhost/~oliver/ajax/test.php`, la línea que sigue configuraría la conexión:

```
request.open("GET", url, true);
```

Antes de enviar la petición al servidor, tenemos que especificar previamente la función a la que va a llamar el navegador cuando reciba la respuesta. Debemos recordar que el navegador no espera la respuesta del servidor, ya que las comunicaciones entre el navegador y el servidor son asíncronas. El campo de objeto de petición `onreadystatechange` se usa para especificar la función de callback.

Como sugiere el nombre, el navegador no sólo llama a la función de callback cuando recibe una respuesta, sino también cada vez que cambia el estado del objeto de petición. Se definen cinco estados para este objeto, que van desde no usado (0) a finalizado (4).

Esperando Respuesta

Finalmente, vamos a usar el método `send` para enviar la petición al servidor. El método puede controlar la información extra que podamos tener como parámetros opcionales, dando la posibilidad de insertar entradas del usuario en los formularios, por ejemplo.

Nuestro sencillo ejemplo no tiene información extra. Esta es la razón por la cual pasamos un `null` como parámetro a `send`.

El Listado 2 muestra todo el código de nuestra sencilla aplicación en AJAX. La función de callback aparece en la línea

Listado 1: HTML con AJAX

```
01 <html>
02   <head>
03     <script language="JavaScript"
04       type="text/javascript"
05       src="ajax.js"/>
06   </head>
07   <body>
08     <a href="#"
09       onclick="javascript:
10         getRandom()">AJAX-Test</a>
11     <span id="random">
12     </span>
13   </body>
14 </html>
```

Listado 2: El Listado Completo

```
01 function checkResult()
02 {
03   alert("New state: " +
04     request.readyState);
05 }
06 function getRandom() {
07   request = new
08     XMLHttpRequest();
09   var url =
10     "http://localhost/~oliver/ajax/
11     index.html";
12   request.open("GET",
13     url,true);
14   request.onreadystatechange=
15     checkResult;
16   request.send(null);
17 }
```

10, `checkResult()`, mostrando el estado actual de la petición en un cuadro cada vez que se le llama.

Debido a que el navegador realmente no hace nada con la respuesta del servidor, no es necesario meterse en el lío de escribir un script en PHP. A efectos de demostración, podemos simplemente usar la página de índice para realizar la petición, como se puede ver en el Listado 2. Esta solución no influye en la manera en la que se usan el objeto de petición o la función de callback.

Si no nos aparece un cuadro, debemos volver a la resolución de problemas: ¿error con la transcripción de la variable URL? Copiamos la cadena (sin las comillas), la insertamos en el recuadro de direcciones en otra ventana del navegador, y presionamos Enter. Si el servidor responde con un mensaje de error, debemos comparar la variable con el nombre de archivo en el servidor una vez más.

La configuración de seguridad para Javascript en nuestro navegador es otra potencial fuente de errores: el objeto de petición sólo puede contactar con el servidor que le sirvió la página HTML original. Si las direcciones del servidor difieren, el navegador supone que existe una infracción de la seguridad y genera un mensaje de "Permiso denegado".

Existe una extensión para Firefox denominada Firebug que puede ser de

utilidad para resolver problemas con las aplicaciones AJAX [2]. No sólo muestra los errores de Javascript, sino que también nos puede mostrar cada `XMLHttpRequest` con los campos de cabecera y el código de respuesta.

Carga extra

Evidentemente, estos ejemplos no son exactamente lo que pretendía el inventor de AJAX. Las peticiones al servidor en segundo plano se diseñan para mezclar contenido dinámico con la página Web actual. El script PHP del Listado 3 implementa un servicio de prueba para este propósito, generando un número aleatorio entre 1 y 100 en cada petición.

Si guardamos el script como `random.php` en el mismo directorio del servidor que los demás archivos, tenemos que cambiar la variable `url` en `ajax.js` para que coincidan. Esto nos proporciona la siguiente línea:

```
var url = "http://localhost/~oliver/ajax/random.php";
```

Para indicarle al navegador que cargue el código Javascript modificado, sólo tenemos que pulsar en el botón de recargar. Ahora, cuando pulsemos en el enlace, las ventanas de cuadro de diálogo mostrarán el estado del objeto de petición.

Por supuesto, tenemos que completar la transferencia para acceder a la información del servidor (este es el estado 4 del objeto de petición), por lo que puede que queramos que el manejador del callback verifique el estado del terminal en cada cambio de estado y no procese la carga extra hasta que el objeto alcance este estado.

Podemos usar la variable de los objetos `readyState` para leer la información. `responseText` guarda la información extra en ASCII plano. La función `checkResult()` ahora tiene ese aspecto, como puede apreciarse en el Listado 4.

En este punto, cuando recargamos la

Listado 3: Script PHP para Número Aleatorio

```
01 <?
02   srand(time());
03   $random = (rand()%100);
04   print $random;
05 ?>
```

página en nuestro navegador y pulsamos el enlace, aparece un cuadro con el número aleatorio generado por el servidor.

Actualizar la Página

Con esto se completan dos de los tres pasos para crear una aplicación AJAX. Hemos usado un objeto de petición para enviar una petición al servidor, recibido una respuesta y procesado la respuesta. Esto deja a la página Web actualizándose a sí misma. Queremos visualizar la información extra en la posición asignada y formatearla de manera atrayente, en lugar de mostrarla sin más en un cuadro.

El encargado de ello será de nuevo el código Javascript. El diseño de la página se representa del lado del navegador según el Document Object Model (DOM, véase la Figura 3). La estructura en árbol del documento nos ofrece la capacidad de referenciar, leer y modificar cualquier elemento HTML en Javascript.

Podemos suministrar código para insertar elementos dentro del árbol. Los elementos se muestran en la página HTML. La función *Firefox Tools | DOM Inspector* nos permite inspeccionar el árbol de documentos (véase la Figura 3). De momento, sólo vamos a modificar un elemento existente para ver el resultado de la petición AJAX. Es sencillo modificar un elemento si éste tiene una ID específica, la cual podemos usar para referenciarla en Javascript.

Hemos asignado una ID de *random* al elemento span en el archivo HTML mostrado en el Listado 1.

La función *The getElementById()* de Javascript para el documento completo nos proporciona una referencia al elemento HTML. El comando siguiente fija adecuadamente los elementos HTML *innerHTML* al valor aleatorio pedido.

Listado 4: La Función `checkResult()`

```
01 function checkResult()
02 {
03   if (request.readyState == 4)
04   {
05     alert("Response: " +
06       request.responseText);
07   }
```

```
var randomDiv =
document.getElementById(
"random");
randomDiv.innerHTML =
request.responseText;
```

Sólo tenemos que añadir estas dos líneas a *checkResult()*, en lugar de la función *alert*, para completar la aplicación AJAX: el código Javascript escribe el resultado directamente en la página HTML en vez de recargar la página completa.

Incompatibilidad

Esto es lo que pasa al menos en teoría, ya que lo que ocurre en realidad en las labores de desarrollo puede ser bastante distinto. Cada navegador hace lo que quiere, de manera que los programadores AJAX tendrán que lidiar con cada una de sus peculiaridades.

Comencemos con Internet Explorer. Las versiones actuales no implementan el objeto XML Request, o al menos no de manera que nos devuelva los resultados requeridos al llamar a *XMLHttpRequest()*. Microsoft espera que implementemos un objeto ActiveX en su lugar, pero afortunadamente, al menos reacciona de la misma manera que el objeto de petición. De hecho, se requiere otra variante, ya que las diferentes versiones de Internet Explorer usan sintaxis diferentes.

Konqueror de KDE y el navegador Safari de Apple son dos candidatos más, ya que algunas construcciones de Javascript que se usan para controlar el árbol DOM pueden causar problemas. Por ejemplo, el procedimiento que acabamos de ver para acceder al elemento que queremos cambiar a través de la función de documentación *getElementById()* no funcionará. Para guardar el trabajo existen ejemplos en AJAX con soluciones para varios navegadores en [3].

La función *checkResult()* del listado muestra también cómo hacer la interacción más dinámica. Como se pudo ver en la última variante, espera hasta que la petición se ha completado (estado 4) antes de fijar el elemento span con el resultado. Para el resto, el estado cambia, y comienza cuando se

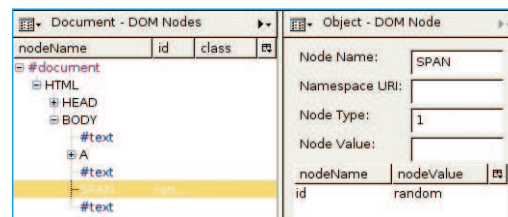


Figura 3: Seleccionamos Tools | DOM Inspector en Firefox para inspeccionar el Document Object Model de una página HTML.

establece la conexión. Cambia el texto a *Loading...* para dar al usuario cierta retroalimentación de lo que está pasando.

Ayuda en la Web

Como se puede apreciar, las aplicaciones Web en AJAX requieren un ligero esfuerzo adicional respecto a las páginas tradicionales, para empezar, con el diseño. No todas las páginas son susceptibles de "ajaxizarlas". Una planificación adecuada es incluso más importante que el desarrollo a la vieja usanza. Finalmente, necesitamos un lado servidor para cada sección AJAX de una página. Además, el código Javascript para cada aplicación AJAX tiene que saber exactamente cómo están estructuradas las páginas. Las Hojas de Estilo en Cascada (CSS) pueden proporcionar una estructura mejorada, a pesar de que esto signifi que haya que añadir incluso más ficheros.

Las numerosas librerías de Javascript y AJAX que podemos encontrar en Internet pueden ser de gran utilidad. Sajax [4] abstrae las peticiones AJAX y el proceso de respuesta, eliminando las preocupaciones acerca de compatibilidad de navegadores. Rico [5], y la Yahoo User Interface Library (YUI) [6] añaden más comodidad. Ambos incluyen funciones para crear interfaces HTML dinámicas. ■

RECURSOS

- [1] AJAX: A New Approach to Web Applications: <http://adaptivepath.com/publications/essays/archives/000385.php>
- [2] Firebug: <http://www.joehewitt.com/software/firebug>
- [3] Listados de este artículo: <http://www.linux-magazine.es/Downloads/XXXX/ajax>
- [4] Sajax: <http://www.modernmethod.com/sajax>
- [5] Rico: <http://openrico.org>
- [6] YUI: <http://sourceforge.net/projects/yui>