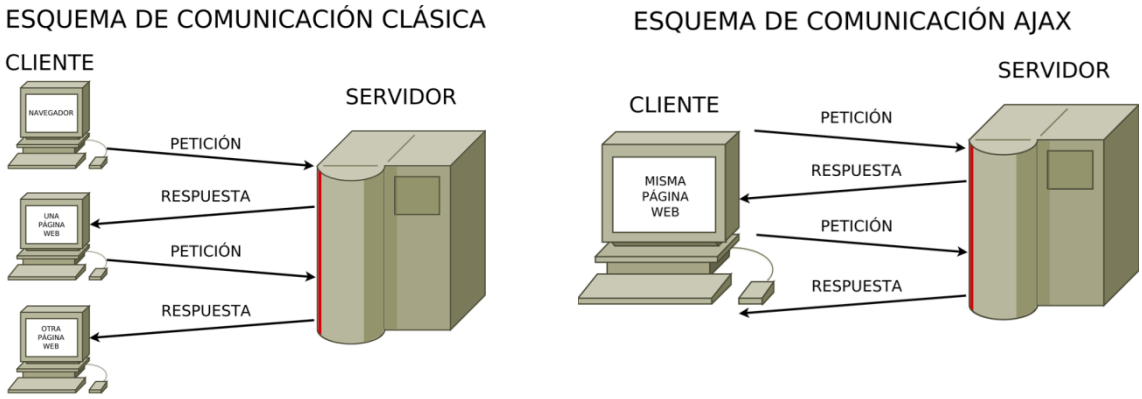
TEMA 3. Ajax

Introducción

Ajax es el acrónimo de *Asynchronous JavaScript and XML* y hace referencia a un conjunto de técnicas que nos permiten enviar y recibir información del servidor desde un documento HTML sin tener que volver a cargarlo. Obviamente esto supone una mejora de la experiencia de usuario sobresaliente, pues el usuario tendrá la impresión de estar contemplando un documento "vivo" en el que los contenidos se van actualizando en tiempo real; piense por ejemplo en la portada de un diario de economía en el que las cotizaciones se van actualizando automáticamente sin que el usuario tenga que recargar una y otra vez el documento.



El término Ajax fue acuñado por Jeese James Garret en 2005, pero en realidad las técnicas a las que se refiere son muy anteriores, casi tan antiguas como la propia Web. Dentro de estas técnicas destaca el objeto XMLHttpRequest (abreviado frecuentemente como XHR), que fue introducido por Internet Explorer 5 en 1999. Este objeto es el núcleo de la mayoría de aplicaciones Ajax actuales y ha sido estandarizado por el W3C dentro del DOM. A pesar de su nombre, XMLHttpRequest permite intercambiar con el servidor datos en cualquier formato, no sólo en XML.

Comunicación asíncrona actual: el objeto XMLHttpRequest

El procedimiento general para comunicarse mediante el objeto XMLHttpRequest consta de las siguientes fases:

1. Crear una instancia del objeto XMLHttpRequest como si de cualquier otro objeto intrínseco se tratara. Por ejemplo: `var xhr = new XMLHttpRequest();`.
2. Si se desea, añadir oyentes de evento al objeto XMLHttpRequest. Podemos configurar oyentes para distintos eventos que se producen durante la comunicación; el más importante de estos eventos (por su universal aceptación) es `readystatechange`, que se dispara cada vez que cambia el estado de la petición (y que se explica con más detalle en esta misma sección). No obstante, existen otros (que se explican en la sección siguiente), como `progress`, que

sólo podrán ser debidamente "escuchados" si asignamos el oyente antes de ejecutar el método `open` (siguiente paso) del objeto XMLHttpRequest.

3. Llamar al método `open` del objeto XMLHttpRequest enviándole como argumentos el tipo de petición que deseamos realizar (generalmente 'GET' o 'POST'), el url al que queremos remitir la petición, y si queremos realizarla asíncronamente (true) o síncronamente (false); por ejemplo, `xhr.open('GET','ajax.php',true);`. Si realizamos una petición síncrona, la ejecución de nuestro código JavaScript se detendrá hasta que se reciba la respuesta. Por el contrario, si la petición es asíncrona, el código podrá seguir su flujo de ejecución. En cualquier caso (síncrono o asíncrono), al recibir la respuesta, se disparará un evento `readystatechange` sobre el objeto XMLHttpRequest.

Nota: ¿GET o POST? Generalmente las peticiones GET se utilizan cuando queremos solicitar información al servidor enviándole unos datos como criterio de selección de esa información; por el contrario, las peticiones POST se utilizan habitualmente para enviar información al servidor que queremos que almacene (por ejemplo, en una base de datos, o en un archivo). Las peticiones GET tienen restringido su tamaño, mientras que las POST no. Las peticiones POST pueden enviar archivos pero las GET no. Las peticiones GET son más rápidas que las POST. Existen otros métodos como PUT o DELETE, pero su uso aún no está ampliamente aceptado (incluso algunos firewalls bloquean este tipo de peticiones).

---

4. Definir los encabezados de la petición a través del método `setRequestHeader('`*`nombreEncabezado`*`','`*`valorEncabezado`*`')` del objeto XMLHttpRequest. Una petición HTTP está compuesta por el contenido en sí y los encabezados. En los encabezados el navegador envía información adicional al servidor, como cookies, formato del contenido de la petición (XML, JSON, …), …

5. Enviar la petición al servidor a través del argumento del método `send` del objeto XMLHttpRequest; por compatibilidad con versiones antiguas de navegadores, si no queremos enviar ningún dato en la petición conviene enviar a este método el argumento `null`. Si la petición era síncrona se detendrá la ejecución del código JavaScript hasta que se reciba la respuesta del servidor y después podremos saber cuál ha sido el tipo de respuesta a través de la propiedad `status` del objeto XMLHttpRequest (esta propiedad está disponible tanto en las peticiones síncronas como en las asíncronas, y contiene el código de estado devuelto por el protocolo HTTP; por ejemplo, el valor 200 indica que la transacción se ha tramitado correctamente, el valor 304 indica que el recurso solicitado no ha sido modificado desde la última vez que fue solicitado, por lo que puede obtenerse de la caché del navegador, el valor 403 indica que no estamos autorizados a acceder al recurso solicitado, el valor 404 indica que el recurso solicitado no existe, y el valor 500 indica que se ha producido un error interno en el servidor al procesar el recurso solicitado (puede consultar el listado completo de códigos de estado HTTP en `http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html`). ¿Y si el servidor no contesta? El navegador se quedará eternamente bloqueado. Obviamente esto resultaría nefasto, y ése es el principal motivo por el que las peticiones Ajax síncronas (conocidas como Sjax) apenas se utilizan. Por el contrario, en las peticiones asíncronas seguirá el flujo de

ejecución normal y, si lo deseamos, podremos incluso controlar el progreso de la transacción a través de distintos eventos; el evento más importante es `readystatechange`, que se dispara cada vez que se actualiza el valor de la propiedad `readyState` del objeto XMLHttpRequest (no confundir con la propiedad readyState del objeto document). En la siguiente tabla se describen los valores que puede adoptar esta propiedad de sólo lectura.

Valor	Descripción
0	Sin inicializar. Aún no se ha llamado al método `open` del objeto XMLHttpRequest.
1	Abierto. Se ha llamado al método `open` del objeto XMLHttpRequest, pero no al método send.
2	Enviado. Se ha llamado al método `send` del objeto XMLHttpRequest, pero aún no se ha recibido la respuesta del servidor.
3	Recibiendo. Se ha empezado a recibir la respuesta del servidor.
4	Completado. Se ha recibido la respuesta completa del servidor. Tenga en cuenta que esto no quiere decir que la petición se haya tramitado correctamente; por ejemplo, puede ser que el servidor haya contestado con un código 404. Para tener la certeza de que una petición se ha tramitado correctamente tendremos que tener el valor 4 en `readyState` y el valor 200 en `status`.

Nota: Además de la propiedad `status`, el objeto XMLHttpRequest también posee la propiedad `statusText`, que en lugar del código de estado contiene el mensaje completo de estado; por ejemplo: '200 OK'.

6. Gestionar la respuesta del servidor. El contenido de la respuesta estará disponible en la propiedad `responseText` del objeto XMLHttpRequest, pero también podemos acceder a los encabezados de la respuesta a través de los métodos `getResponseHeader(`*`nombreEncabezado`*`)` y `getAllResponseHeaders()`.

Por ejemplo, copie los siguientes dos archivos en la carpeta de proyectos de su servidor web, acceda al primero con su navegador web y pulse el botón **ENVIAR PETICIÓN** para comprobar cómo se reciben datos nuevos del servidor sin volver a recargar la pagina. El archivo ajax.php está escrito en PHP y simplemente escribe en la respuesta el mensaje 'La fecha del servidor es:', deja pasar 10 segundos, y escribe la fecha/hora actual del sistema, que es lo que recibirá nuestro objeto xhr en su propiedad `responseText`.

index.html

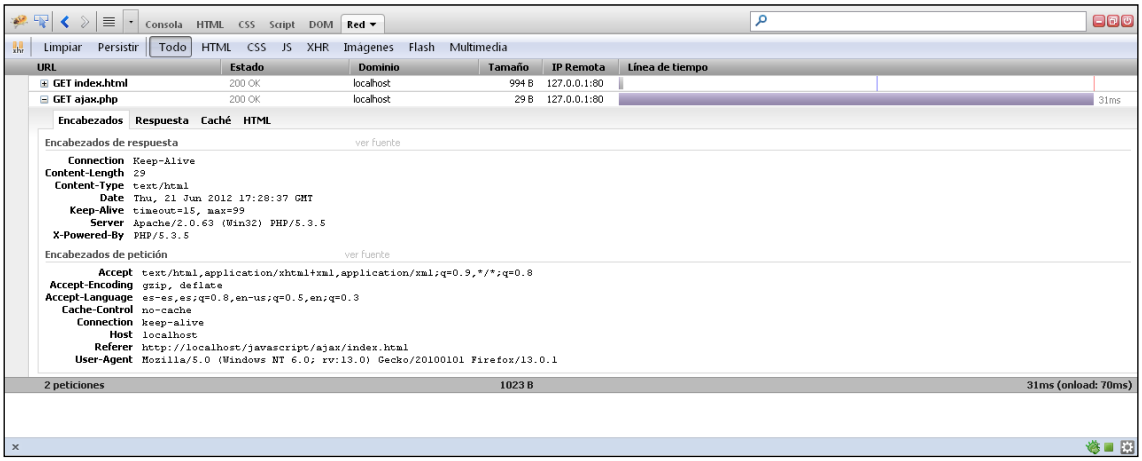
```
001  <!DOCTYPE html>
002  <html>
003   <head>
004    <title>AJAX</title>
005    <style>
006    </style>
007    <script>
008     var xhr;
009     function enviarPeticionAJAX(evento){
```

```
010       evento.target.disabled=true;
011       xhr = new XMLHttpRequest();
012       xhr.addEventListener('readystatechange',gestionarRespuesta);
013       xhr.open('GET','ajax.php',true);
014       xhr.send(null);
015      }
016      function gestionarRespuesta(evento){
017       if(evento.target.readyState == 4 && evento.target.status == 200){
018        document.getElementById('encabezados').innerHTML =
    evento.target.getAllResponseHeaders();
019        document.getElementById('contenido').innerHTML = evento.target.responseText;
020        document.getElementById('boton').disabled = false;
021       }
022      }
023    </script>
024   </head>
025   <body>
026    <button id='boton' type='button' onclick='enviarPeticionAJAX(event);'>ENVIAR
    PETICI&Oacute;N</button>
027    <h1>Encabezados respuesta</h1>
028    <div id='encabezados'></div>
029    <h1>Contenido respuesta</h1>
030    <div id='contenido'></div>
031   </body>
032  </html>
```

ajax.php

```
001  <?php
002   echo 'La fecha del servidor es: <br />';
003   sleep(10);
004   echo date(DATE_RFC822);
005  ?>
```

Nota: Para depurar las peticiones y respuestas que se producen en AJAX es muy útil recurrir al panel **Red** de Firebug. Por ejemplo, en la siguiente figura se muestra la información que nos aporta este panel al ejecutar el ejemplo anterior.



EJERCICIO

Cambie en la línea 13 el url ajax.php por otro que no exista y modifique el código del oyente gestionarRespueta para que indique que se ha producido un error en caso de que el estado (`status`) de la respuesta no sea 200 o 304 (recuerde que el código 304 indica que el recurso no ha cambiado y se está sirviendo desde la caché del navegador en lugar de solicitarlo completamente al servidor).

Controlar el progreso de las peticiones Ajax

En las transacciones Ajax asíncronas podemos controlar su progreso a través de los eventos asociados al objeto XMLHttpRequest. El más importante de estos eventos es `reeadystatechange`, que ya explicamos en la sección anterior, pero además disponemos de los siguientes (aunque su implementación no está completamente homogeneizada en los navegadores):

- `abort`. Se produce al cancelar la transacción con el método `abort()` del objeto XMLHttpRequest. Es excluyente de los eventos `error` y `load`.
- `error`. Se produce cuando se detecta un error en la transacción. Es excluyente de los eventos `abort` y `load`.
- `load`. Se produce cuando la transacción se ha realizado correctamente. No debemos confundir que la transacción sea correcta con que la respuesta sea correcta. Por ejemplo, podemos tener una transacción correcta con una respuesta de código 404 (documento no encontrado). Por este motivo siempre hay que consultar el valor de la propiedad `status` del objeto XMLHttpRequest. Este evento excluyente de los eventos `abort` y `error`.
- `loadstart`. Se produce al recibir el primer byte de la respuesta.
- `progress`. Es junto a readystatechange el único evento que puede producirse varias veces durante la transacción. Indica que se ha producido algún progreso en la recepción de la respuesta.
- `timeout`. Se produce si la transacción ha superado su plazo de ejecución y va a cancelarse. El plazo de ejecución de un objeto XMLHttpRequest se establece a través de su propiedad `timeout`, y su valor se expresa en milisegundos.

Todos estos eventos, excepto `readystatechange`, poseen las siguientes propiedades:

- `lengthComputable`. Es un valor booleano que nos indica si se conoce el tamaño en bytes total de la respuesta.
- `loaded`. Indica cuántos bytes de la respuesta se han cargado ya.
- `total`. Si `lengthComputable` es true, indica el tamaño total en bytes de la respuesta.

Por ejemplo, vamos a completar el código anterior añadiendo algunos oyentes (el código adicional se ha resaltado en negrita). Pruebe este código varias veces y en distintos navegadores (Firefox, Opera y Chrome, por ejemplo) alternando el valor de la propiedad `timeout` de 5000 a 15000, para comprobar que la implementación de los eventos distintos a `readystatechange` es ligeramente desigual.

```
001    <!DOCTYPE html>
```

```
002    <html>
003     <head>
004      <title>AJAX</title>
005      <style>
006      </style>
007      <script>
008       var xhr;
009       function enviarPeticionAJAX(evento){
010        evento.target.disabled=true;
011        xhr = new XMLHttpRequest();
012        xhr.addEventListener('readystatechange',gestionarRespuesta,false);
013
       xhr.addEventListener('abort',function(){console.log('abort');evento.target.disab
      led = false;});
014
       xhr.addEventListener('load',function(){console.log('load');evento.target.disable
      d = false;});
015
       xhr.addEventListener('error',function(){console.log('error');evento.target.disab
      led = false;});
016        xhr.timeout = '15000';
017
       xhr.addEventListener('timeout',function(){console.log('timeout');evento.target.d
      isabled = false;});
018        xhr.addEventListener('progress',gestionarProgreso,false);
019        xhr.open('GET','ajax.php',true);
020        xhr.send(null);
021       }
022       function gestionarRespuesta(evento){
023        console.log('readystatechange: ' + evento.target.readyState);
024        if(evento.target.readyState == 4 && evento.target.status == 200){
025         document.getElementById('encabezados').innerHTML =
      evento.target.getAllResponseHeaders();
026         document.getElementById('contenido').innerHTML =
      evento.target.responseText;
027         document.getElementById('boton').disabled = false;
028        }
029       }
030       function gestionarProgreso(evento){
031        console.log('progress: ' +
      (evento.lengthComputable?evento.loaded/evento.total * 100 + '%':'desconocido'));
032       }
033      </script>
034     </head>
035     <body>
036      <button id='boton' type='button' onclick='enviarPeticionAJAX(event);'>ENVIAR
      PETICI&Oacute;N</button>
037      <h1>Encabezados respuesta</h1>
038      <div id='encabezados'></div>
039      <h1>Contenido respuesta</h1>
040      <div id='contenido'></div>
041     </body>
042    </html>
```

Envío de datos al servidor

Existen dos técnicas para configurar los datos que queremos enviar al servidor en las peticiones GET y POST de Ajax:

- El método clásico que consiste en adjuntar los datos como cadena de búsqueda, bien en el propio url de la petición en las peticiones GET, o bien como argumento del método `send` del objeto XMLHttpRequest en las peticiones POST. En este último caso, el de las peticiones POST, si queremos que el servidor reciba los datos como si procediesen de un formulario (el método POST se usa generalmente para enviar datos de formulario) tendremos que configurar en encabezado `Content-Type` con el valor `application/x-www-form-urlencoded`. El mayor inconveniente del método clásico es que nos obliga a

recurrir al DOM para ir recopilando todos los datos de los controles que queremos enviar, y esto puede resultar bastante tedioso en algunos casos; por el contrario, su mayor ventaja es que es compatible con todos los navegadores. Por ejemplo, en este primer código se realiza una petición GET, y el en siguiente se resaltan en negrita las modificaciones que habría que introducir para realizar la misma petición con el método POST; tras ellos, el tercer código corresponde a la aplicación del lado del servidor, que simplemente responde con la calificación del alumno y materia elegidos (es válida para los métodos GET y POST porque utiliza la superglobal $_REQUEST).

index.html para petición GET

```
001  <!DOCTYPE html>
002  <html>
003   <head>
004    <title>AJAX</title>
005    <style>
006    </style>
007    <script>
008     var xhr;
009     var alumno;
010     var materia;
011     var calificacion;
012     function enviarPeticionAJAX(evento) {
013      if (alumno.value !='' && materia.value != ''){
014       alumno.disabled = true;
015       materia.disabled = true;
016       xhr = new XMLHttpRequest();
017       xhr.addEventListener('readystatechange', gestionarRespuesta, false);
018       xhr.open('GET', 'ajax.php?alumno=' + alumno.value + '&materia=' +
     materia.value, true);
019       xhr.send(null);
020      }
021     }
022     function gestionarRespuesta(evento) {
023      if (evento.target.readyState == 4 && evento.target.status == 200) {
024       alumno.disabled = false;
025       materia.disabled = false;
026       calificacion.value = evento.target.responseText;
027      }
028     }
029     document.addEventListener('readystatechange', inicializar, false);
030     function inicializar() {
031      if (document.readyState == 'complete') {
032       alumno = document.getElementById('alumno');
033       materia = document.getElementById('materia');
034       calificacion = document.getElementById('calificacion');
035       alumno.addEventListener('change', enviarPeticionAJAX, false);
036       materia.addEventListener('change', enviarPeticionAJAX, false);
037      }
038     }
039    </script>
040   </head>
041   <body>
042    <label for='alumno'>Alumno: </label>
043    <select id='alumno'>
044     <option value='' selected='selected'>--Elija un alumno--</option>
045     <option>Juan F&eacute;lix Mateos</option>
046     <option>Ana Irene Palma</option>
047    </select>
048    <label for='materia'>Materia: </label>
049    <select id='materia'>
050     <option value='' selected='selected'>--Elija una materia--</option>
051     <option>Lenguaje</option>
052     <option>Matem&aacute;ticas</option>
053    </select>
054    <label for='calificacion'>Calificaci&oacute;n: </label>
055    <input type='text' readonly='readonly' id='calificacion'/>
056   </body>
057  </html>
```

index.html para petición POST

```
<!DOCTYPE html>
<html>
 <head>
  <title>AJAX</title>
  <style>
  </style>
  <script>
   var xhr;
   var alumno;
   var materia;
   var calificacion;
   function enviarPeticionAJAX(evento) {
    if (alumno.value !='' && materia.value != ''){
     alumno.disabled = true;
     materia.disabled = true;
     xhr = new XMLHttpRequest();
     xhr.addEventListener('readystatechange', gestionarRespuesta, false);
     xhr.open('POST', 'ajax.php', true);
     xhr.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');
     xhr.send('alumno=' + alumno.value + '&materia=' + materia.value);
    }
   }
   function gestionarRespuesta(evento) {
    if (evento.target.readyState == 4 && evento.target.status == 200) {
     alumno.disabled = false;
     materia.disabled = false;
     calificacion.value = evento.target.responseText;
    }
   }
   document.addEventListener('readystatechange', inicializar, false);
   function inicializar() {
    if (document.readyState == 'complete') {
     alumno = document.getElementById('alumno');
     materia = document.getElementById('materia');
     calificacion = document.getElementById('calificacion');
     alumno.addEventListener('change', enviarPeticionAJAX, false);
     materia.addEventListener('change', enviarPeticionAJAX, false);
    }
   }
  </script>
 </head>
 <body>
  <label for='alumno'>Alumno: </label>
  <select id='alumno'>
   <option value='' selected='selected'>--Elija un alumno--</option>
   <option>Juan F&eacute;lix Mateos</option>
   <option>Ana Irene Palma</option>
  </select>
  <label for='materia'>Materia: </label>
  <select id='materia'>
   <option value='' selected='selected'>--Elija una materia--</option>
   <option>Lenguaje</option>
   <option>Matem&aacute;ticas</option>
  </select>
  <label for='calificacion'>Calificaci&oacute;n: </label>
  <input type='text' readonly='readonly' id='calificacion'/>
 </body>
</html>
```

ajax.php

```
<?php
 $alumno = $_REQUEST['alumno'];
 $materia = $_REQUEST['materia'];
 switch ($alumno){
  case 'Juan Félix Mateos':
   switch ($materia){
    case 'Matemáticas':
     echo '7.5';
     break;
    case 'Lenguaje':
```

```
011         echo '9.5';
012         break;
013       }
014       break;
015     case 'Ana Irene Palma':
016       switch ($materia){
017         case 'Matemáticas':
018           echo '8.5';
019           break;
020         case 'Lenguaje':
021           echo '7.5';
022           break;
023       }
024       break;
025   }
026 ?>
```

- Utilizar el novedoso tipos de datos (u objeto intrínseco) `FormData`. Este objeto es capaz de generar automáticamente la cadena de búsqueda de un formulario, sin que tengamos que ir recabando el valor de cada control recurriendo al DOM, y además se encarga de configurar implícitamente los encabezados para hacer creer al servidor que esta cadena de búsqueda procede de un formulario que se ha enviado utilizando la codificación `multipart/form-data` (esto puede tener ciertas implicaciones que se resaltarán en la sección posterior "Cross-Origin Resource Sharing") Obviamente esto resulta muy cómodo, pero hay navegadores que no implementan este objeto aún. Además, el objeto FormData nos ofrece el método `append(nombre,valor)` por si queremos añadir un par que no procede del formulario a la petición. Por ejemplo, en el siguiente código se han englobado los controles de lista desplegables en un formulario para poder serializar directamente sus datos con el objeto FormData y también se utiliza el método `append` para ilustrar cómo se añadiría un par nombre/dato adicional a una petición (el archivo ajax.php es el mismo del ejemplo anterior, y se han resaltado en negrita las principales diferencias):

```
001 <!DOCTYPE html>
002 <html>
003   <head>
004     <title>AJAX</title>
005     <style>
006     </style>
007     <script>
008       var xhr;
009       var alumno;
010       var materia;
011       var calificacion;
012       var datos;
013       function enviarPeticionAJAX(evento) {
014         if (alumno.value !='' && materia.value != ''){
015           datos = new FormData(document.forms[0]);
016           alumno.disabled = true;
017           materia.disabled = true;
018           datos.append('curso','11/12');
019           xhr = new XMLHttpRequest();
020           xhr.addEventListener('readystatechange', gestionarRespuesta, false);
021           xhr.open('POST', 'ajax.php', true);
022           xhr.send(datos);
023         }
024       }
025       function gestionarRespuesta(evento) {
026         if (evento.target.readyState == 4 && evento.target.status == 200) {
027           alumno.disabled = false;
028           materia.disabled = false;
029           calificacion.value = evento.target.responseText;
030         }
031       }
```

```
032       document.addEventListener('readystatechange', inicializar, false);
033       function inicializar() {
034        if (document.readyState == 'complete') {
035         alumno = document.getElementById('alumno');
036         materia = document.getElementById('materia');
037         calificacion = document.getElementById('calificacion');
038         alumno.addEventListener('change', enviarPeticionAJAX, false);
039         materia.addEventListener('change', enviarPeticionAJAX, false);
040        }
041       }
042     </script>
043    </head>
044    <body>
045     <form id='formulario'>
046      <label for='alumno'>Alumno: </label>
047      <select id='alumno' name='alumno'>
048       <option value='' selected='selected'>--Elija un alumno--</option>
049       <option>Juan F&eacute;lix Mateos</option>
050       <option>Ana Irene Palma</option>
051      </select>
052      <label for='materia'>Materia: </label>
053      <select id='materia' name='materia'>
054       <option value='' selected='selected'>--Elija una materia--</option>
055       <option>Lenguaje</option>
056       <option>Matem&aacute;ticas</option>
057      </select>
058      <label for='calificacion'>Calificaci&oacute;n: </label>
059      <input type='text' readonly='readonly' id='calificacion'/>
060     </form>
061    </body>
062   </html>
```

Nota: Tenga en cuenta que los controles de los formularios que están deshabilitados (atributo disabled) no se envían al servidor. ¿Qué hubiera ocurrido en el código anterior si las líneas 16 y 17 se hubieran colocado delante de la 15? Que como los campos estarían deshabilitados al crear la instancia de FormData no se incluirían en él.

Ejercicio: Cuadros de lista con opciones sensibles al contexto

Partimos de un documento con un formulario con dos cuadros de lista desplegables. En el primero se ofrecen los nombres de las comunidades autónomas de España y se deben cargar mediante Ajax en el segundo cuadro las provincias de la comunidad elegida en el primer cuadro. Este segundo cuadro de lista deberá permanecer deshabilitado hasta que se elija una comunidad en el primero, y si se cambia el valor de la comunidad también deberá deshabilitarse hasta que se obtengan los datos necesarios mediante Ajax.

Ajax con otros tipos de datos: XML, JSON y archivos

Hasta ahora, en todos los ejemplos de este capítulo nos hemos limitado a responder desde el servidor con fragmentos de código HTML o texto (si se fija en los encabezados de respuesta en Firebug verá que su Content-type es siempre text/html o text/plain). Sin embargo, en la práctica es más frecuente utilizar respuestas que contengan datos estructurados, bien en XML o en JSON, pues JavaScript posee propiedades y métodos que permiten aprovechar la estructura de esos datos. En el caso de XML, recuerde que el DOM es válido para HTML y XML, de modo que podremos aprovechar todos los métodos que ya conocemos (como getElementsByTagName o getAttribute) para acceder directamente a los datos. En el caso de JSON, los datos recibidos son directamente interpretados como objetos de JavaScript, de modo que podremos acceder a sus propiedades con la notación de punto o la literal.

XML

XML es un lenguaje de marcas que sirve para estructurar datos; su sintaxis es muy similar a la de HTML. Por ejemplo:

alumno_1.xml

```
001  <?xml version="1.0" encoding="UTF-8" ?>
002  <estudiante>
003    <nombre>Juan Félix Mateos</nombre>
004    <calificaciones curso="11/12">
005      <calificacion materia="Matemáticas" nota="7.5"/>
006      <calificacion materia="Lenguaje" nota="9"/>
007    </calificaciones>
008  </estudiante>
```

Inicialmente el objeto XMLHttpRequest fue diseñado para trabajar con datos XML, pero actualmente este formato está dejando paso a otros menos verbosos, como JSON. El hecho de que XML sea tan verboso es que lentifica las transacciones.

Para trabajar con datos XML en Ajax utilizaremos la propiedad `responseXML` en lugar de `responseText`, pues en ella dispondremos del documento DOM de los datos XML recibidos, y podremos explotarlos con todo el arsenal de técnicas de que disponemos.

Nota: Para que la propiedad `responseXML` contenga el documento XML, es decir un objeto de la clase Document del DOM (recuerde que el DOM es válido para HTML y XML), es necesario que el content-type declarado por el servidor en la respuesta sea text/xml. Si no es así, la propiedad `responseXML` contendrá el valor null. Si el servidor no declara correctamente el content-type podemos suplantarlo en el cliente llamando al método `overrideMimeType` del objeto XMLHttpRequest (se recomienda hacerlo antes de llamar al método `send()`. Por ejemplo, si el servidor devuelve un fragmento de un documento XML pero declara su content-type como text/plain, podríamos usar en el cliente el método `overrideMimeType('text/xml')` para que la propiedad `responseXML` lo adquiera correctamente.

Por ejemplo, cree el archivo del listado anterior con el nombre alumno_1.xml y otro llamado alumno_2.xml con el código que se muestra a continuación.

alumno_2.xml

```
001  <?xml version="1.0" encoding="UTF-8" ?>
002  <estudiante>
003    <nombre>Ana Irene Palma</nombre>
004    <calificaciones curso="11/12">
005      <calificacion materia="Matemáticas" nota="8"/>
006      <calificacion materia="Lenguaje" nota="9.5"/>
007    </calificaciones>
008  </estudiante>
```

Y, a continuación, cree el archivo index.xml con el código que se muestra a continuación.

```
001  <!DOCTYPE html>
002  <html>
003    <head>
004      <title>AJAX</title>
```

```
005     <style>
006     </style>
007     <script>
008      var xhr;
009      var alumno;
010      var materia;
011      var calificacion;
012      var datos;
013      function enviarPeticionAJAX(evento) {
014       if (alumno.value !='' ){
015        alumno.disabled = true;
016        materia.disabled = true;
017        calificacion.value = '';
018        materia.selectedIndex = 0;
019        datos = new FormData(document.forms[0]);
020        xhr = new XMLHttpRequest();
021        xhr.addEventListener('readystatechange', gestionarRespuesta, false);
022        xhr.open('POST', 'alumno_' + alumno.value + '.xml', true);
023        xhr.send(null);
024       }else{
025        calificacion.value = '';
026        materia.selectedIndex = 0;
027        materia.disabled = true;
028       }
029      }
030      function gestionarRespuesta(evento) {
031       if (evento.target.readyState == 4 && evento.target.status == 200) {
032        alumno.disabled = false;
033        materia.disabled = false;
034        datos = evento.target.responseXML;
035       }
036      }
037      function actualizarCalificacion(){
038       if (materia.value !=''){
039        var i;
040        var calificaciones = datos.getElementsByTagName('calificacion');
041        for (i=0;i<calificaciones.length;i++){
042         if(calificaciones[i].getAttribute('materia') == materia.value){
043          calificacion.value = calificaciones[i].getAttribute('nota');
044         }
045        }
046       }
047      }
048      document.addEventListener('readystatechange', inicializar, false);
049      function inicializar() {
050       if (document.readyState == 'complete') {
051        alumno = document.getElementById('alumno');
052        materia = document.getElementById('materia');
053        calificacion = document.getElementById('calificacion');
054        alumno.addEventListener('change', enviarPeticionAJAX, false);
055        materia.addEventListener('change', actualizarCalificacion, false);
056       }
057      }
058     </script>
059    </head>
060    <body>
061     <form id='formulario'>
062      <label for='alumno'>Alumno: </label>
063      <select id='alumno' name='alumno'>
064       <option value='' selected='selected'>--Elija un alumno--</option>
065       <option value='1'>Juan F&eacute;lix Mateos</option>
066       <option value='2'>Ana Irene Palma</option>
067      </select>
068      <label for='materia'>Materia: </label>
069      <select id='materia' name='materia' disabled='disabled'>
070       <option value='' selected='selected'>--Elija una materia--</option>
071       <option>Lenguaje</option>
072       <option>Matem&aacute;ticas</option>
073      </select>
074      <label for='calificacion'>Calificaci&oacute;n: </label>
075      <input type='text' readonly='readonly' id='calificacion'/>
076     </form>
077    </body>
078   </html>
```

JSON

JSON es el acrónimo de *JavaScript Object Notation* y simplemente es una notación para expresar objetos JavaScript casi idéntica a la notación literal que aprendimos en el tema 1. La principal diferencia es que en JSON los nombres de las propiedades deben ser cadenas escritas entre comillas dobles (no está estandarizado el uso de comillas simples). Por ejemplo, el objeto JavaScript expresado en notación literal como `{edad: 38, peso: 79}` se expresaría en JSON como `{"edad": 38, "peso": 79}`.

Actualmente JSON es uno de los formatos más utilizados en transacciones Ajax debido a que es muy compacto (frente a la verbosidad de XML).

JavaScript nos ofrece el objeto JSON con los métodos `stringify(`*`objeto`*`)`, para convertir un objeto en una cadena expresada en JSON, y `parse(`*`cadena`*`)`, para interpretar una cadena expresada en JSON como un objeto de JavaScript.

Por ejemplo, en el siguiente código enviamos al servidor una cadena expresada en JSON creada a partir de un objeto con `JSON.stringify`, y recibimos de él una cadena que convertimos en un objeto mediante `JSON.parse`. En el archivo ajax.php simplemente se accede al cuerpo de la petición, en el que se encuentra la cadena enviada desde JavaScript, se convierte esa cadena en un array haciendo uso de la instrucción `json_decode`, y se devuelve una cadena expresada en JSON que JavaScript puede convertir en un objeto mediante `JSON.parse()`.

index.html

```
001  <!DOCTYPE html>
002  <html>
003   <head>
004    <title>AJAX</title>
005    <style>
006    </style>
007    <script>
008     var xhr;
009     var alumno;
010     var materia;
011     var calificacion;
012     var objetoPeticion = new Object();
013     var objetoRespuesta;
014     function enviarPeticionAJAX(evento) {
015      if (alumno.value !='' && materia.value !=''){
016       objetoPeticion.alumno = alumno.value;
017       objetoPeticion.materia = materia.value;
018       alumno.disabled = true;
019       materia.disabled = true;
020       xhr = new XMLHttpRequest();
021       xhr.addEventListener('readystatechange', gestionarRespuesta, false);
022       xhr.open('POST', 'ajax.php', true);
023       xhr.setRequestHeader("Content-Type", "application/json");
024       xhr.send(JSON.stringify(objetoPeticion));
025      }else{
026       calificacion.value = '';
027      }
028     }
029     function gestionarRespuesta(evento) {
030      if (evento.target.readyState == 4 && evento.target.status == 200) {
031       alumno.disabled = false;
032       materia.disabled = false;
033       objetoRespuesta = JSON.parse(evento.target.responseText);
034       calificacion.value = objetoRespuesta.calificacion;
035      }
036     }
037     document.addEventListener('readystatechange', inicializar, false);
038     function inicializar() {
039      if (document.readyState == 'complete') {
```

```
040         alumno = document.getElementById('alumno');
041         materia = document.getElementById('materia');
042         calificacion = document.getElementById('calificacion');
043         alumno.addEventListener('change', enviarPeticionAJAX, false);
044         materia.addEventListener('change', enviarPeticionAJAX, false);
045        }
046       }
047     </script>
048   </head>
049   <body>
050     <form id='formulario'>
051       <label for='alumno'>Alumno: </label>
052       <select id='alumno' name='alumno'>
053         <option value='' selected='selected'>--Elija un alumno--</option>
054         <option>Juan F&eacute;lix Mateos</option>
055         <option>Ana Irene Palma</option>
056       </select>
057       <label for='materia'>Materia: </label>
058       <select id='materia' name='materia'>
059         <option value='' selected='selected'>--Elija una materia--</option>
060         <option>Lenguaje</option>
061         <option>Matem&aacute;ticas</option>
062       </select>
063       <label for='calificacion'>Calificaci&oacute;n: </label>
064       <input type='text' readonly='readonly' id='calificacion'/>
065     </form>
066   </body>
067 </html>
```

ajax.php

```
001 <?php
002   $entrada = fopen('php://input','r');
003   $datos = fgets($entrada);
004   $datos = json_decode($datos,true);
005   switch ($datos['alumno']){
006     case 'Juan Félix Mateos':
007       switch ($datos['materia']){
008         case 'Matemáticas':
009           echo '{"calificacion":7.5}';
010           break;
011         case 'Lenguaje':
012           echo '{"calificacion":9.5}';
013           break;
014       }
015       break;
016     case 'Ana Irene Palma':
017       switch ($datos['materia']){
018         case 'Matemáticas':
019           echo '{"calificacion":8.5}';
020           break;
021         case 'Lenguaje':
022           echo '{"calificacion":7.5}';
023           break;
024       }
025       break;
026   }
027 ?>
```

Archivos

Tradicionalmente, para enviar archivos a un servidor mediante Ajax se ha recurrido a técnicas muy poco ortodoxas basadas en el uso de Flash o iframes ocultos. Sin embargo, gracias al objeto FormData ahora podemos enviar archivos con la misma facilidad que el resto de los datos de un formulario; los controles `input` de tipo `file` se codifican en el FormData de forma transparente.

Más aún, los mismos eventos que se explicaron anteriormente para controlar el progreso de la recepción de una respuesta Ajax (`progress`, `load`, `timeout`, …), pueden

utilizarse también sobre la propiedad `upload` del objeto XMLHttpRequest para controlar el progreso del envío de archivos al servidor (pues esta propiedad es un objeto de tipo XMLHttpRequestUpload).

Nota: La propiedad `timeout` se establece sobre el objeto XMLHttpRequest y afecta a la transacción completa. No se puede asignar un timeout independiente para la propiedad `upload`, pero si un oyente para el evento timeout sobre `upload`, de modo que podamos indicar al usuario que su velocidad de subida es demasiado lenta..

Por ejemplo, en el siguiente código utilizamos un control `input` de tipo `file` para solicitar al usuario que elija una imagen de avatar. Esa imagen se envía al servidor mediante Ajax al pulsar el botón Enviar, y podemos seguir el avance de la transacción mediante el control de tipo `progress`, que se va actualizando con los eventos `progress` y `load` de la propiedad `upload` del objeto XMLHttpRequest. Debajo del control `progress` hay una imagen vacía, que se utilizará para mostrar la imagen enviada una vez concluida la transacción, evidenciando visualmente que se ha transmitido correctamente. El archivo ajax.php simplemente recibe el archivo de la imagen, lo mueve de la carpeta de recepción temporal de PHP a su misma carpeta, y devuelve el nombre del archivo como respuesta de la petición para que el documento index.html pueda utilizarlo como valor del atributo `src` de la imagen que estaba inicialmente vacío, demostrándose así que la imagen se ha transmitido correctamente.

index.html

```
001  <!DOCTYPE html>
002  <html>
003   <head>
004    <title>AJAX</title>
005    <meta charset="utf-8" />
006    <style>
007    </style>
008    <script>
009     var xhr;
010     function enviarPeticionAJAX(evento) {
011      var datos = new FormData(document.forms[0]);
012      xhr = new XMLHttpRequest();
013      xhr.timeout = 2000;
014      xhr.upload.addEventListener('progress',gestionarProgreso);
015      xhr.upload.addEventListener('load',cargaCompletada);
016      xhr.upload.addEventListener('timeout',subidaLenta);
017      xhr.addEventListener('readystatechange', gestionarRespuesta);
018      xhr.open('POST', 'ajax.php', true);
019      xhr.send(datos);
020     }
021     function gestionarRespuesta(evento) {
022      if (evento.target.readyState == 4 && evento.target.status == 200) {
023       var imagenAvatar= document.getElementById('imagenAvatar');
024       imagenAvatar.src = evento.target.responseText;
025      }
026     }
027     function gestionarProgreso(evento){
028      document.getElementById('progreso').max = evento.total;
029      document.getElementById('progreso').value = evento.loaded;
030     }
031     function cargaCompletada(evento){
032      document.getElementById('progreso').max = 1;
033      document.getElementById('progreso').value = 1;
034     }
035     function subidaLenta(evento){
```

```
036       alert('Cancelado. Su velocidad de subida es demasiado lenta.')
037      }
038    </script>
039   </head>
040   <body>
041    <form id='formulario'>
042     <label for='avatar'>Avatar: </label>
043     <input type='file' id='avatar' name='avatar' accept='image/*'/>
044     <br />
045     <button type='button' onclick='enviarPeticionAJAX(event);'>ENVIAR</button>
046     <br />
047     <progress id="progreso" value="0"></progress>
048    </form>
049    <img width='100' height='100' src='' id='imagenAvatar' />
050   </body>
051  </html>
```

ajax.php

```
001  <?php
002   header("Access-Control-Allow-Origin: *");
003   if ($_SERVER['REQUEST_METHOD'] == 'OPTIONS') {
004    exit;
005   }
006   $ext = strtolower(substr(strrchr($_FILES['avatar']['name'], "."), 1));
007   if (array_search($ext,array('jpg','gif','png'))  === false){
008    exit; //Evitar que se suban otros archivos.
009   }
010   $target_path = "./";
011   $target_path = $target_path . $_FILES['avatar']['name'];
012   if (move_uploaded_file($_FILES['avatar']['tmp_name'], $target_path)) {
013    echo utf8_encode($_FILES['avatar']['name']);
014   }
015  ?>
```

Nota: Tenga en cuenta que, por defecto, muchos servidores web tienen limitado el tamaño máximo de los archivos que pueden recibir por HTTP a 2 megabytes. En Apache puede aumentar este límite a través de las directivas upload_max_filesize y post_max_size del archivo php.ini.