AJAX

Jordi Sánchez Cano

PID_00172706

UOC
Universitat Oberta de Catalunya

Índice

Introducción

Una vez que se conocen las diferentes tecnologías utilizadas por AJAX (HTML, XML, HTTP, Java-Script, DOM, etc.), es el momento de aprender cómo hay que utilizarlas para desarrollar aplicaciones web dinámicas.

En primer lugar, se estudian las diferentes técnicas de comunicación asíncrona con un servidor HTTP que permitirán intercambiar datos y actualizar partes de una página sin interferir con la interacción del usuario.

El formato predilecto de intercambio de datos utilizado por AJAX es XML. En este módulo se estudiará cómo obtener y manipular documentos XML por medio de su representación mediante el DOM. Una vez obtenido un documento XML, aprenderemos a transformarlo al formato HTML mediante XSLT, y posteriormente, a actualizar parcialmente la página actual.

Por último, se estudiará el formato JSON, alternativo a XML. Este formato representa la información de forma más ligera y accesible desde Java-Script.

Objetivos

Los objetivos de este módulo didáctico son los siguientes:

1. Saber qué es el HTML dinámico y crear webs con contenidos actualizables una vez que se ha cargado la página.

2. Conocer las diferentes técnicas de comunicación asíncrona con el servidor empleadas por AJAX y cuándo hay que utilizar cada una de ellas.

3. Estudiar a modo de introducción qué es la accesibilidad web y cómo se puede hacer accesible una página.

4. Conocer la estructura de un documento representado por el DOM.

5. Aprender a cargar documentos XML y acceder a ellos mediante el DOM.

6. Seleccionar partes de un documento XML mediante XPath.

7. Realizar transformaciones XSL y mostrar el documento obtenido de forma dinámica.

8. Intercambiar y manipular información con el servidor mediante JSON.

1. Técnicas de comunicación AJAX

En este apartado veremos las diferentes técnicas de comunicación con un servidor HTTP[1].

(1)HTTP es la sigla de *hypertext transfer protocol* ('protocolo de transferencia de hipertexto').

1.1. Introducción

El modelo de una aplicación web tradicional está basado en la navegación entre páginas, de forma que cada vez que se requiere mostrar nueva información, la aplicación solicita una nueva página, que sustituye a la actual.

AJAX posibilita la creación de aplicaciones web interactivas mediante el uso de diferentes técnicas y tecnologías. Por un lado, permite intercambiar información con el servidor utilizando peticiones asíncronas, sin que esto influya en la interacción del usuario y sin la necesidad de navegar hacia otra página. Por otro lado, AJAX también posibilita la actualización de partes de una página con la información intercambiada durante la comunicación asíncrona.

Ved también

Ved la diferencia entre el modelo tradicional y una aplicación web que hace uso de AJAX en el apartado 2 del módulo "Introducción a AJAX" de esta asignatura.

1.1.1. Comunicación asíncrona

Inicialmente, se utilizaron algunas particularidades de los marcos HTML[2] para el intercambio de información asíncrona. El uso de estas técnicas se popularizó de tal forma que los navegadores empezaron a integrar un nuevo objeto, que se llamó XMLHttpRequest, implementado especialmente para esta tarea.

(2)HTML es la sigla de *hypertext markup language* ('lenguaje de marcado de hipertexto').

XMLHttpRequest

XMLHttpRequest es una interfaz que implementan la mayoría de navegadores. Permite el intercambio de información de forma asíncrona con un servidor HTTP.

Aunque una de las utilidades más importantes de AJAX es la actualización parcial de una página, también existen muchas otras que se limitan a la comunicación asíncrona.

Ejemplo de comunicación asíncrona

Existen clientes de correo basados en web (también llamados *clientes de webmail*) que utilizan AJAX para enviar el texto de un nuevo mensaje de forma periódica mientras se escribe. El servidor almacenará el mensaje para poder recuperarlo en caso de error y así continuar con su edición.

1.1.2. Actualización parcial de una página

Una vez cargada una página HTML, es posible modificar su contenido de forma dinámica con el uso de DHTML. Se trata de un conjunto de técnicas combinadas con HTML, Java-Script, CSS y DOM[3]. Mediante código *script*, se puede acceder a un documento HTML y modificar su estructura con el DOM. Estos

(3)DOM es la sigla de *document object model* ('modelo de objetos del documento' o 'modelo en objetos para la representación de documentos').

cambios se muestran al instante en el navegador. El uso de HTML dinámico es uno de los pilares básicos de AJAX, ya que permite mostrar los datos que se obtienen de manera asíncrona.

DHTML

DHTML es la sigla de *dynamic HTML* ('HTML dinámico'). Es un término usado para definir el uso conjunto de HTML, CSS y DOM con un lenguaje de *script*.

Ved también

Los ejemplos numerados que se presentan a lo largo de todo este subapartado están también disponibles en línea.

Ejemplo 1

El ejemplo 1 consta de una página que, una vez cargada, permite modificar su contenido al presionar un botón. Esta acción crea dinámicamente una etiqueta `<h3>` con el texto "Texto a mostrar", que se añade al documento actual.

```
<html>
   <head>
      <title>Ejemplo DOM 1</title>
      <script type="text/javascript">
         sContenido = 'Texto a mostrar';
         function mostrarTexto(){
            //Se crea un elemento <h3>
            var elH3 = document.createElement('h3');
            //Se crea un nodo de texto con el texto contenido
            //en sContenido
            var elText = document.createTextNode(sContenido);
            //Se añade el nodo de texto al nodo <h3>
            elH3.appendChild(elText);
            //Se añade el nodo <h3> al cuerpo de esta misma página
            document.body.appendChild(elH3);
         }
      </script>
   </head>

   <body>
      <button onclick="mostrarTexto()">
         Haga clic para mostrar el texto
      </button>
   </body>
</html>
```

El botón de la página tiene asignado al evento `click` una función Java-Script. La primera línea de dicha función crea un nodo encabezado `<h3>` con el método `document.CreateElement(nombreTag)`. La segunda línea crea un nodo de tipo texto a partir de la variable `sContenido`. Posteriormente, añade el nodo de texto creado al elemento `<h3>` para formar el texto de la cabecera. Para mostrarlo, se añade el elemento `<h3>` al nodo `<body>` del documento. Esto cambia el modelo del documento y fuerza al navegador a refrescar la vista de la interfaz de usuario, en la que aparecerá la nueva cabecera. Este método es poco práctico y requiere mucho desarrollo.

Existe una forma más sencilla de actualizar una página sin tener que crear HTML dinámico. Las etiquetas `<div>` y `<span>` son agrupaciones lógicas que permiten diferenciar partes del documento para facilitar el acceso al mismo y así cambiar su contenido y/o estilo. Ambas disponen de la propiedad `innerHTML`, que permite establecer el texto HTML de su contenido.

Propiedad *innerHTML*

Al establecer texto HTML a la propiedad `innerHTML`, se actualiza el modelo de datos del documento de forma automática.

Ejemplo 2

El ejemplo 2 muestra el mismo contenido que el ejemplo anterior, pero utilizando el elemento `<div>`.

```
<html>
   <head>
      <title>Ejemplo DOM con una división</title>
      <script type="text/javascript">
         sContenido = '<h3>Texto a mostrar</h3>';
         function mostrarTexto(){
           var div = document.getElementById('seccionActualizable');
            div.innerHTML = sContenido;
         }
      </script>
   </head>
   <body>
      <button onclick="mostrarTexto()">
         Haz clic para mostrar el texto
      </button>
      <div id="seccionActualizable"></div>
   </body>
</html>
```

En este ejemplo, la función `mostrarTexto ()` obtiene el objeto que representa la división definida en el `<body>`. Posteriormente, se establece en la propiedad `innerHTML` el código HTML `<h3>Texto a mostrar</h3>`. De forma automática, el navegador muestra los cambios. En los ejemplos posteriores se utiliza el contenedor `<div>` para mostrar nuevos contenidos.

1.2. Técnica de los marcos ocultos

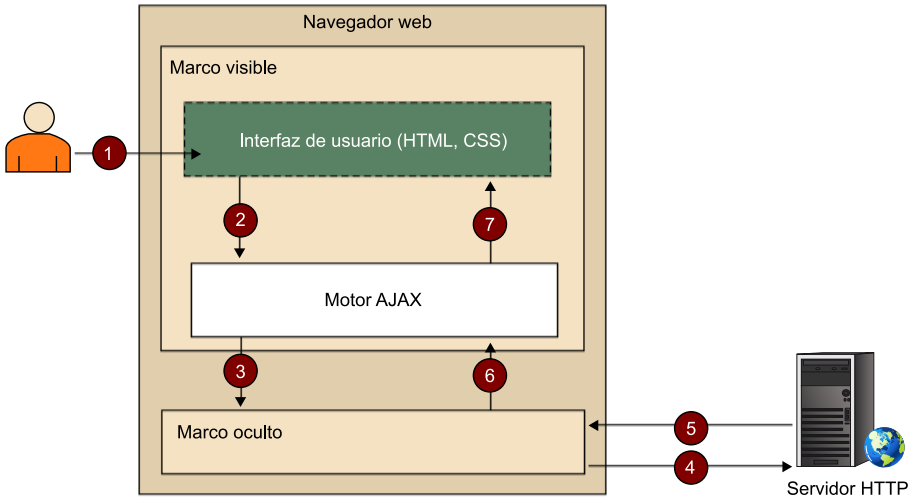
La primera técnica utilizada para la comunicación asíncrona se basó en el uso de marcos HTML. Este procedimiento consiste en definir un conjunto de marcos en una página de los que al menos uno de ellos es invisible para el usuario.

Cada vez que sea necesaria la obtención de nuevos datos, se cambiará la localización del contenido del marco oculto para generar una nueva petición HTTP. Una vez recibidos los nuevos datos en el marco oculto, éstos se podrán tratar y mostrar dentro de marcos visibles.

Ocultar un marco

Para ocultar un marco a la vista del usuario, se debe establecer el alto y el ancho a 0 puntos. Aun así, algunos navegadores antiguos muestran los bordes del marco.

Figura 1. Comunicación con marcos ocultos



A continuación, explicamos los pasos que se han de llevar a cabo en una comunicación por medio de marcos ocultos para actualizar parte del documento actual (dichos pasos pueden verse gráficamente en la figura 1).

1) El usuario realiza una acción sobre un elemento HTML (por ejemplo, seleccionar un elemento de una lista o pulsar un botón).

Acciones sobre un elemento HTML

Algunas veces el desencadenante de la petición no es el usuario. Por ejemplo, podría darse el caso de un proceso Java-Script que generara una petición de forma periódica.

2) El elemento HTML notifica la acción a una o más funciones *script* contenidas en la página (también llamadas *motor AJAX*).

3) El motor AJAX recarga el contenido del marco oculto hacia un nuevo URL en el que se encuentra el contenido que debe descargarse.

4) El marco oculto realiza la petición al servidor HTTP.

5) El servidor procesa y genera los nuevos datos, y los envía de vuelta al navegador. En este caso, el destinatario es el marco oculto.

6) La página cargada en el marco oculto dispondrá de una función Java-Script que será ejecutada y que notificará al motor AJAX la recepción del contenido.

7) El motor AJAX procesa la información recibida en el marco oculto y realiza los cambios en el marco visible.

La **interfaz de usuario** que aparece en la figura 1 hace referencia a la parte visual de interacción con la aplicación, y está compuesta por un conjunto de páginas HTML y estilos CSS.

El **motor AJAX** es el componente formado por el conjunto de funciones Java-Script que controlan el intercambio de información y la actualización de la interfaz de usuario. En el caso de los marcos ocultos, el motor dispondrá de dos funciones básicas:

1) Una función que contiene la lógica necesaria para desencadenar las peticiones HTTP. Cada acción que requiera obtener nueva información llamará a esta función.

2) Una función para tratar los datos alojados dentro del marco oculto una vez obtenida la respuesta HTTP.

Los marcos ocultos permiten la generación de peticiones aisladas del marco principal, puesto que éstas se llevan a cabo en otro oculto. Según el propósito de la operación, las peticiones HTTP generadas pueden ser de dos tipos: **GET** y **POST**. A continuación, pasamos a explicar cómo se pueden realizar estas peticiones mediante marcos ocultos.

1.2.1. Peticiones GET

Marcos HTML

En HTML, los marcos corresponden a la etiqueta `<frame>`.

Para realizar una petición de tipo GET, se debe especificar, en la propiedad `location` de un marco HTML oculto, el URL con el recurso que se desea obtener. Para disponer de marcos ocultos, la aplicación web debe contar con una página que defina los marcos visibles y los marcos ocultos. En los marcos visibles, se mostrará la interfaz de usuario. Los marcos ocultos se reservan para llevar a cabo las peticiones que se vayan necesitando.

A continuación, explicamos cada uno de los pasos de esta técnica con dos ejemplos.

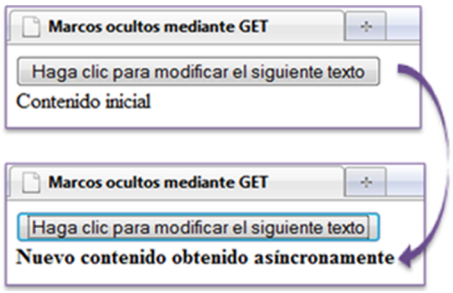
Primer ejemplo de petición GET

Este primer ejemplo (ejemplo 3) consiste en una página con un texto y un botón.

Ejemplo 3

Al hacer clic sobre el botón, se obtiene un texto del servidor, que se representa en la página, tal como se muestra en la figura 2.

Figura 2. Ejemplo simple de marcos ocultos



Acto seguido, mostramos el contenido completo de los diferentes ficheros que forman este ejemplo y, más adelante, explicaremos paso a paso cómo podemos actualizar el texto:

- *index.html*:

```
<html>
   <frameset rows="100%,0" frameborder="no">
      <frame name="visibleFrame" src="ejemplo1.html" noresize="noresize" />
      <frame name="hiddenFrame" src="about:blank" noresize="noresize" />
   </frameset>
</html>
```

Ésta es la página inicial de la aplicación, y en ella se definen los marcos `visibleFrame` y `hiddenFrame`. La proporción de los dos marcos es de 100% y 0% respectivamente sobre el alto disponible, de manera que el segundo marco será invisible para el usuario. El atributo `noresize`, establecido a `"noresize"`, evita que el usuario pueda cambiar el tamaño de los marcos.

En el marco visible, se establecerá la página siguiente:

- *ejemplo1.html*:

```
<html>
   <head>
      <script type="text/javascript">
         function requestNewContent(){
            top.frames['hiddenFrame'].location = 'nuevocontenido.html';
         }
         function updateNewContent(sDataText){
            var division = document.getElementById('divUpdatable');
            division.innerHTML = sDataText;
         }
      </script>
      <title>Marcos ocultos mediante GET</title>
   </head>
   <body>
      <input type="button" onclick="requestNewContent();"
      value="Haga clic para modificar el siguiente texto" />
      <div id="divUpdatable">
         Contenido inicial
      </div>
   </body>
</html>
```

Esta página, que se cargará dentro del marco visible, dispone de la interfaz de usuario y del motor AJAX: la interfaz está formada por un botón y por un texto encapsulado en una división a la que se le ha asignado el identificador `divUpdatable`. El botón solicitará una nueva página al motor AJAX y el nuevo contenido se establecerá dentro de esta división.

El motor AJAX está formado por las funciones siguientes:

1) `requestNewContent()`. Esta función se encarga de solicitar el nuevo contenido. Localiza el marco oculto y modifica la propiedad `location` con el URL del documento que se desea obtener, iniciando el diálogo con el servidor.

2) `updateNewContent(sDataText)`. Esta función modifica la interfaz de usuario estableciendo el contenido textual pasado por parámetro dentro de la división `divUpdatable`. Para ello, localiza la división y establece el código HTML en la propiedad `innerHTML` de la división.

Uso del DOM

Ambas funciones hacen uso del DOM para acceder a los diferentes elementos de la página actual y modificarlos.

- *nuevocontenido.html*

```
<html>
<head>
<script type="text/javascript">
   window.onload =
   function()
   {
      var division = document.getElementById('divNuevoContenido');
      top.frames['visibleFrame'].updateNewContent(division.innerHTML);
   };
</script>
</head>
<body>
   <div id="divNuevoContenido">
      <b>Nuevo contenido obtenido asíncronamente</b>
   </div>
</body>
</html>
```

Esta página es la obtenida por el marco oculto. Contiene el texto que se desea mostrar encapsulado en una división para facilitar el acceso a él. La función asignada al evento `window.onload` localiza la división `divNuevoContenido` y pasa el código HTML que contiene la función `updateNewContent(sDataText)` del motor AJAX.

Evento *onload*

El evento `onload` asignado al objeto `window` se ejecutará cuando la página se haya cargado por completo en el marco oculto.

Una vez presentados los ficheros que intervienen en el ejemplo, explicaremos paso a paso cómo se actualiza el contenido de la página.

1) Petición del nuevo contenido

La petición del nuevo contenido se inicia cuando el usuario hace clic sobre el botón del marco visible. Éste tiene asignado, en el evento `onclick`, la función `requestNewContent()`.

```
...
```

```
<body>
   <input type="button" onclick="requestNewContent();"
   value="Haga clic para modificar el siguiente texto" />
...
```

Esta función establece el URL 'nuevocontenido.html' en la propiedad location del marco oculto. Este URL es relativo al URL de la página que lo contiene. Al estar en la misma localización, basta con indicar sólo el nombre del documento.

```
function requestNewContent(){
   top.frames['hiddenFrame'].location = 'nuevocontenido.html';
}
```

Al modificar la propiedad location del marco, se envía una petición GET al servidor.

2) Obtención del nuevo contenido

El marco oculto recibe la página nuevocontenido.html. Al cargarla, llama a la función asignada al evento window.onload de este documento.

```
window.onload =
function()
{
   var division = document.getElementById("divNuevoContenido");
   top.frames["visibleFrame"].updateNewContent(division.innerHTML);
};
```

La primera línea obtiene, a partir del DOM, la división en la que se encuentra el contenido que se desea visualizar. La segunda localiza el marco visible y llama a la función updateNewContent(), pasando por parámetro el código HTML de la división localizada en la línea anterior.

3) Actualización del texto

La función updateNewContent() a la que se llama en el paso anterior realiza lo siguiente:

```
function updateNewContent(sDataText){
   var division = document.getElementById("divUpdatable");
   division.innerHTML = sDataText;
}
```

La primera línea obtiene la división en la que se debe visualizar el nuevo contenido. La segunda línea establece en la división el código HTML obtenido por parámetro. Hecho esto, la página ha sido modificada sin necesidad de que se recargue por completo.

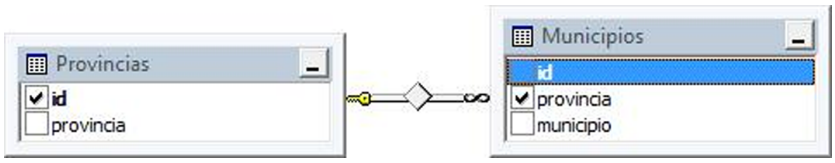
Segundo ejemplo de petición GET

Este ejemplo (ejemplo 4) consiste en una página HTML con dos listas que permiten al usuario escoger una provincia y una localidad respectivamente. Como las localidades dependen de las provincias, será necesario actualizar la segunda lista cada vez que se seleccione una provincia de la primera.

Ejemplo 4

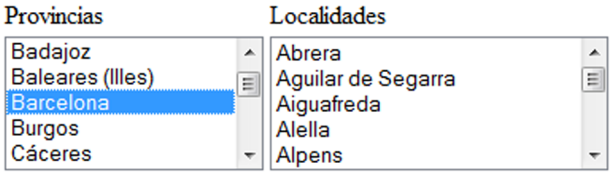
El servidor HTTP dispone de una base de datos con dos tablas que almacenan el conjunto de provincias y municipios.

Figura 3. Tablas de provincias y municipios



AJAX interviene en el momento en el que el usuario selecciona una provincia de la primera lista: se deberá realizar una petición GET al servidor HTTP añadiendo al URL un parámetro con el identificador de la provincia seleccionada. El servidor tendrá que devolver una página que contenga una lista de las diferentes localidades, que posteriormente se incorporará en una división del marco visible. Si, por ejemplo, se ha seleccionado la provincia de Barcelona, se actualizará la lista de localidades, de manera que quedará de la siguiente manera:

Figura 4. Captura del ejemplo



A continuación, mostramos el contenido completo de los diferentes ficheros que forman este ejemplo y, más adelante, explicaremos paso a paso cómo se puede actualizar la lista de municipios:

- *index.html*:

```
<html>
<frameset rows="100%,0" frameborder="no">
   <frame name="visibleFrame" src="ProvincesCities.php" noresize="noresize" />
   <frame name="hiddenFrame" src="about:blank" noresize="noresize" />
</frameset>
</html>
```

Se establecen dos marcos: uno visible y el otro oculto.

- *provincesCities.php*:

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<script type="text/javascript">

/* Función que obtiene la lista de ciudades en función de la provincia
 * seleccionada. La información se alojará en el marco oculto */

function requestCities(){
   //Se obtiene el índice de la provincia seleccionada
   var index = document.getElementById('selectProvinces').selectedIndex;
   //Se obtiene el valor del ítem de la provincia seleccionada
   var id = document.getElementById('selectProvinces').options[index].value;
   //Se redirecciona el marco oculto a la página getCities.php
   //añadiendo el parámetro id al URL
   top.frames['hiddenFrame'].location = 'getCities.php?id=' + id;
}

/* Función a la que se llamará desde el marco oculto una vez que se disponga
 * de una nueva lista de ciudades. Permite establecer el código HTML
 * pasado por parámetro a una división en el marco visible. */

function updateNewContent(sDataText){
   var division = document.getElementById('divUpdatable');
   division.innerHTML = sDataText;
}

</script>
<title>Marcos ocultos GET</title>
</head>
<body>
<h3>Selecciona una provincia y una localidad:</h3>
<table>
   <tr><td>Provincias</td><td>Municipios</td></tr>
   <tr><td>
      <select id="selectProvinces" size="15" onclick="requestCities()">
         <?php
         //Se prepara la consulta SQL
         $db_nombre = 'ajax';
         $link = mysql_connect('localhost','ajax','J&J007')
            or die('Could not connect ' . mysql_errno());
         mysql_select_db($db_nombre ,$link) or die("Error selecting database.");
         $sqlQuery = 'Select id, provincia from provincias';
         //Si la consulta ha devuelto algún resultado...
         if($result = mysql_query($sqlQuery))
```

```
        {
           //Por cada registro devuelto se añade un <option> a la lista de provincias
           while($row = mysql_fetch_assoc($result))
           {?>
              <option value="<?php echo $row["id"]?>">
                 <?php echo $row["provincia"]?>
              </option>
           <?php }
        }
        mysql_close($link);
        ?>
     </select>
  </td>
  <td>
     <div id="divUpdatable"></div>
  </td>
  </tr>
</table>
</body>
</html>
```

Ésta es la página principal alojada en el marco visible en el que interactuará el usuario. Contiene las partes siguientes:

a) Un motor AJAX, con dos funciones similares a las del ejemplo anterior:

- `requestCities()`, que inicia una petición HTTP asíncrona modificando la propiedad `location` del marco oculto.

- `updateNewContent(sDataText)`, que muestra el texto HTML pasado por parámetro en la división `divUpdatable`.

b) Un *script* PHP, que genera la lista HTML de provincias obtenidas de la base de datos.

c) Una división con el identificador `divUpdatable`, que contendrá la segunda lista, con los municipios de la provincia que se seleccione en cada momento.

- *`getCities.php`*:

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<script type="text/javascript">

/* La siguiente función se ejecutará al cargarse esta página en el marco
```

```
 * oculto. Enviará parte del contenido de la página al motor AJAX
 * para que lo renderice en el marco visible. */
window.onload =
   function()
   {
      var divCities = document.getElementById('divisionCities');
      top.frames['visibleFrame'].updateNewContent(divCities.innerHTML);
   };
</script>
</head>
<body>
<div id="divisionCities">
   <select size="15">
      <?php
         // Se crea la sentencia SQL para obtener las ciudades a partir del
         // identificador de provincia pasado por parámetro en el URL
         $db_nombre = 'ajax';
         $idProvincia = $_GET["id"];
         $link = mysql_connect('localhost','ajax','J&J007')
         or die('Could not connect ' . mysql_errno());
         mysql_select_db($db_nombre ,$link) or die("Error seleccionando la base de datos.");
         $sqlQuery = "Select municipio from municipios Where provincia=\"" .
         $idProvincia."\"";
         //Si hay registros...
         if($result = mysql_query($sqlQuery))
         {
            //Por cada ciudad obtenida, se añade un <option> dentro de la lista
            while($row = mysql_fetch_assoc($result))
            {?>
               <option><?php echo $row["municipio"]?></option>
            <?php }
         }
      mysql_close($link);
      ?>
   </select>
</div>
</body>
</html>
```

Esta página será solicitada para obtener los municipios de una provincia. Ejecuta las acciones siguientes:

a) Recoge el identificador de provincia pasado por parámetro en el URL de la petición.

b) Obtiene de la base de datos la lista de municipios que pertenecen a la provincia.

c) Genera el código HTML con la lista de municipios obtenidos.

A continuación, explicamos paso a paso cómo se obtienen y se muestran las localidades.

1) Carga inicial de provincias

En el marco visible se carga la página `provincesCities.php`, que obtiene la lista de todas las provincias de la base de datos. Este proceso se realiza en el lado del servidor con un *script* PHP.

```
<select id="selectProvinces" size="15" onclick="requestCities()">
   ...
   <option value="<?php echo $row["id"]?>"> <?php echo $row["provincia"]?></option>
   ...
</select>
```

Una vez mostrada esta página, el usuario puede seleccionar una provincia.

2) Selección de una provincia

El usuario ha seleccionado una provincia. La propiedad `onclic` del `<select>` hace que, al seleccionar una provincia, se llame a la función `requestCities()` para iniciar la comunicación con el servidor.

```
<select id="selectProvinces" size="5" onclick="requestCities()">
```

3) Iniciar la comunicación con el servidor

La función `requestCities()` modifica la propiedad `location` del marco oculto.

```
function requestCities(){
   var index = document.getElementById("selectProvinces").selectedIndex;
   var id = document.getElementById("selectProvinces").options[index].value;
   top.frames["hiddenFrame"].location = "getCities.php?id=" + id;
}
```

Las dos primeras líneas obtienen el identificador de provincia a partir del índice del `<option>` seleccionado de la lista de provincias. La última establece, en la propiedad `location` del marco oculto, el URL compuesto por la pági-

na `getCities.php` y el parámetro `"id"` con el identificador de provincia. Al modificar esta propiedad, automáticamente se genera una petición GET al URL establecido.

4) Generación de página y respuesta del servidor

El servidor recibe la petición y procesa la página `getCities.php`. Inicialmente, recupera el parámetro `"id"` del URL para generar la cadena SQL que permitirá obtener todos los municipios que pertenecen a la provincia indicada. Para cada municipio obtenido, se crea un `<option>` con el nombre del municipio.

```
...
<div id="divisionCities">
   <select size="15">
      <?php
      $db_nombre = 'ajax';
      $idProvincia = $_GET["id"];
      $sqlQuery = "Select municipio from municipios Where provincia=\""
      . $idProvincia."\"";
      $link = mysql_connect('localhost','ajax','J&J007')
         or die('Could not connect ' . mysql_errno());
      mysql_select_db($db_nombre ,$link)
         or die("Error seleccionando la base de datos.");
      if($result = mysql_query($sqlQuery))
      {
         while($row = mysql_fetch_assoc($result))
         {?>
            <option><?php echo $row["municipio"]?></option>
         <?php }
      }
      mysql_close($link);
      ?>
   </select>
</div>
...
```

Toda la lista está incluida dentro de una división. Ésta servirá para manipular dicha lista una vez que esté contenida dentro del marco oculto.

Como en el ejemplo anterior, será necesario mover parte de la información del marco oculto a la parte visible.

```
window.onload =
   function()
   {
      var divCities = document.getElementById('divisionCities');
      top.frames['visibleFrame'].updateNewContent(divCities.innerHTML);
```

```
};
```

Al cargarse la página en el marco oculto, esta función llamará a la función `updateNewContent()` situada en el marco visible. Como parámetro, se pasará el contenido de la división `divisionCities`, con la lista de municipios.

5) Actualización del contenido

La función `updateNewContent()` actualiza la división `divUpdatabl` con la nueva lista de municipios pasada por parámetro.

```
function updateCities(sDataText){
   var divisionCities = document.getElementById("divUpdatableCities");
   divisionCities.innerHTML = sDataText;
}
```

Una vez hecha la llamada a esta función, el navegador muestra la nueva lista de ciudades pertenecientes a la provincia seleccionada.

1.2.2. Peticiones POST

En algunos casos, es necesario realizar peticiones POST en lugar de peticiones GET. Un ejemplo de ello sería el envío de múltiples parámetros o de parámetros de tamaño considerable. La longitud del URL en las peticiones está limitada y depende de cada navegador, por lo que en estos casos sería inviable el uso de GET. En cambio, POST permite el envío de una gran cantidad de información dentro del cuerpo de la petición.

Ved también

Ved los métodos HTTP GET y POST en el módulo "Introducción a AJAX" de esta asignatura.

Los marcos HTML sólo permiten enviar peticiones GET, por lo que únicamente serán de utilidad para recibir el contenido. En su lugar, se utilizan los formularios HTML. Mediante formularios HTML es posible especificar el método de la petición.

Formularios HTML

Los formularios pueden contener HTML normal y elementos especiales llamados *controles*. Al enviar un formulario, los *controles* se adjuntan en la petición de forma automática.

Las propiedades más importantes que deben definirse en el formulario son las siguientes:

- `method`. Es el método de la petición. Se debe establecer "post".
- `action`. Contendrá el URL con el recurso solicitado.
- `target`. Con esta propiedad estableceremos el marco de destino de la respuesta.

Marco de destino

Si no se establece el marco oculto como destino de una petición, la respuesta se recibirá en la misma página en la que resida el formulario, haciendo que se recargue la página completa.

La comunicación se inicia al llamar al método `submit()` del formulario. Todos los campos en su interior serán enviados de forma automática dentro del cuerpo de la petición. Si el método establecido en el formulario es GET, los campos del formulario se enviarán dentro del URL.

La recepción y la manipulación de la respuesta funciona de la misma forma que en las peticiones mediante el método GET: el recurso enviado al cliente deberá disponer de una función en el evento `onload` para notificar al motor AJAX que los datos solicitados se encuentran en el marco oculto.

Ejemplo de petición POST

Este ejemplo (ejemplo 5) es una adaptación del anterior, con la diferencia de que ahora el método de las peticiones será mediante POST. Cada provincia seleccionada se enviará dentro del cuerpo de la petición como parámetro, y no dentro del URL como en el ejemplo anterior.

Este ejemplo cuenta con los mismos ficheros que el anterior. Los cambios se hallan en los ficheros `provincesCities.php` y `getCities.php`, y se han marcado con el comentario *nuevo*.

- *provincesCities.php*:

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<script type="text/javascript">
/*
 * Realiza el envío del formulario al servidor mediante POST
 */
      function requestCities(){                          //nuevo
         document.myForm.submit();                       //nuevo
      }                                                  //nuevo

/* Función a la que se llamará desde el marco oculto una vez que se disponga
 * de una nueva lista de ciudades. Permite establecer el código HTML
 * pasado por parámetro a una división en el marco visible. */
function updateNewContent(sDataText){
   var divisionCities = document.getElementById('divUpdatableCities');
   divisionCities.innerHTML = sDataText;
}

</script>
<title>Marcos ocultos POST</title>
</head>
<body>
<h3>Selecciona una provincia y una localidad:</h3>
```

```
<table>
   <tr><td>Provincias</td><td>Municipios</td></tr>
   <tr><td>
      <form name="myForm" method="post" action="getCities.php"
         target="hiddenFrame">                              //nuevo
         <select name="selectProvinces" size="5" onclick=" requestCities()">
            <?php
            //Se obtiene la lista de provincias
            $db_nombre = 'ajax';
            $link = mysql_connect('localhost','ajax','J&J007')
            or die('Could not connect ' . mysql_errno());
            mysql_select_db($db_nombre ,$link) or die("Error selecting database.");
            $sqlQuery = 'Select id, provincia from provincias';
            //Si hay resultados...
            if($result = mysql_query($sqlQuery))
            {
            //Cada provincia obtenida se añade a la lista HTML
            while($row = mysql_fetch_assoc($result))
               {?>
               <option value="<?php echo $row["id"]?>">
                  <?php echo $row["provincia"]?>
               </option>
               <?php }
            }
            mysql_close($link);
            ?>
         </select>
      </form>                                               //nuevo
   </td>
   <td><div id="divUpdatableCities"></div></td>
   </tr>
</table>
</body>
</html>
```

El formulario engloba la lista de provincias. Esto hará que la provincia seleccionada se envíe automáticamente dentro del cuerpo de la petición en forma de parámetro. También se ha establecido como método de solicitud `"post"` y como destino de la navegación, el marco oculto. El URL se indica en el atributo `action`, que solicita la página `getCities.php`.

Al hacer clic sobre una provincia, se llama a la función `requestCities()`. Ahora esta función se limita simplemente a llamar al método `submit()` para iniciar la comunicación.

```
function requestCities(){
   document.myForm.submit();
```

```
}
```

La recepción y la gestión del nuevo contenido se lleva a cabo de la misma forma: la función `updateCities()` obtiene y muestra la nueva lista. Ésta será llamada por la nueva página una vez que se haya cargado por completo en el marco oculto.

- *getCities.php*:

```
<!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">
<script type="text/javascript">
   /* La siguiente función se ejecutará al cargarse esta página en el marco
    * oculto. Enviará parte del contenido de la página al motor AJAX
    * para que lo renderice en el marco visible. */
   window.onload =
      function()
      {
         var divCities = document.getElementById('divisionCities');
         parent.frames['visibleFrame'].updateNewContent(divCities.innerHTML);
      };
</script>
</head>
<body>
<div id="divisionCities">
   <select size="5">
      <?php
         //Se obtiene la lista de ciudades de la provincia a partir de su identificador
         $db_nombre = 'ajax';
         //El identificador de provincia se obtiene a partir del vector _POST
         $idProvincia = $_POST["selectProvinces"];
         $sqlQuery = "Select municipio from municipios Where provincia=\"" .
         $idProvincia."\"";
         $link = mysql_connect('localhost','ajax','J&J007') or
         die('Could not connect ' . mysql_errno());
         mysql_select_db($db_nombre ,$link) or die("Error seleccionando la base de datos.");
         //Si ha habido resultados...
         if($result = mysql_query($sqlQuery))
      {
         //Para cada ciudad, se añade un <option> a la salida
        while($row = mysql_fetch_assoc($result))
         {?>
            <option><?php echo $row["municipio"]?></option>
         <?php }
```

```
        }
        mysql_close($link);
        ?>
    </select>
</div>
</body>
</html>
```

Los únicos cambios realizados sobre este fichero son el tratamiento de parámetros. En el ejemplo anterior, se obtenían a partir del URL de la petición. En este caso, se hallan en el cuerpo de la misma.

```
<?php
    ...
    $idProvincia = $_POST["selectProvincias"];
    ...
?>
```

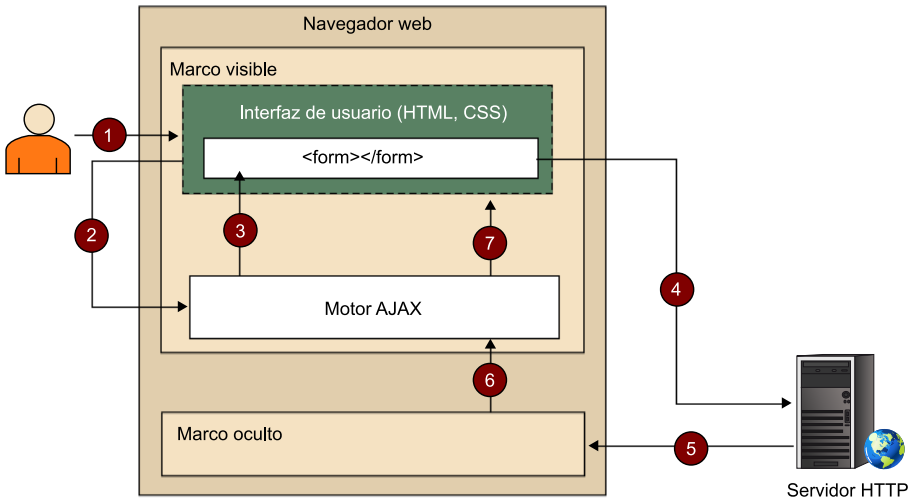
Vector `$_POST`

El vector `$_POST` contiene todas las variables pasadas por el método HTTP POST.

Al igual que en la técnica anterior, es necesario que la página devuelta al cliente contenga una función Java-Script en el evento `window.onload` de la página para forzar la actualización de la parte visible.

La figura 5 muestra paso a paso cada una de las acciones realizadas en este ejemplo.

Figura 5. Comunicación POST con un formulario



1) El usuario hace clic sobre una provincia.

2) El evento asociado a la lista llama a la función `requestCities()` del motor AJAX.

3) El motor AJAX llama al método `submit()` del formulario para iniciar la comunicación.

4) El formulario envía una petición POST al servidor enviando en el cuerpo el elemento seleccionado de la lista.

5) El servidor obtiene el identificador de provincia seleccionado a partir del parámetro de la petición y genera una página que incluye la lista de municipios.

6) El marco oculto recibe la nueva página. Al renderizarla, llama a la función asignada al evento `window.onload`, que llama a la función `updateCities()` pasándole por parámetro la lista de municipios.

7) La función `updateCities()` actualiza la interfaz de usuario con la nueva lista.

1.3. Marcos flotantes

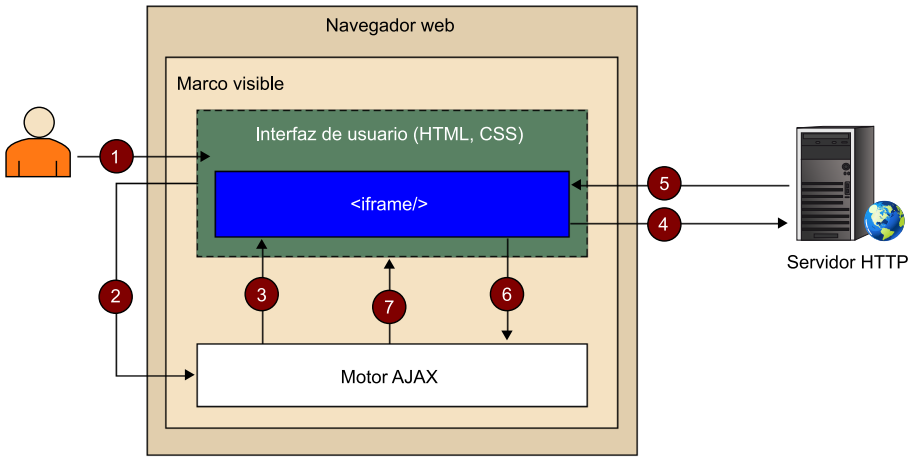
Los marcos flotantes son elementos HTML que, al igual que los marcos tradicionales, pueden presentar una página en su interior. Sin embargo, los marcos flotantes son más versátiles y ofrecen algunas ventajas frente a los marcos estáticos.

Marcos flotantes

Los marcos flotantes se introdujeron en la versión 4.0 de HTML y corresponden a la etiqueta `<iframe>`. Pueden situarse en cualquier posición dentro del área de cliente del navegador y definir esta posición y su dimensión dinámicamente.

En la técnica con marcos ocultos, se precisa de una página dividida en marcos para disponer de uno oculto que permita la comunicación asíncrona. En cambio, los marcos flotantes pueden declararse en la misma página que requiera el uso de AJAX. La arquitectura de una aplicación AJAX que hace uso de marcos flotantes se muestra en la figura 6.

Figura 6. Comunicación con marcos flotantes



Este nuevo modelo de aplicación no exige una división previa de marcos, sino que éstos se declaran en el mismo documento que requiera el uso de AJAX. Los marcos flotantes también pueden declararse de forma dinámica mediante el DOM sin que se precise una definición previa.

Los pasos realizados desde que el usuario lleva a cabo una acción hasta que se actualiza el documento son los siguientes:

1) El usuario interactúa con la interfaz realizando acciones que requieren nuevos datos.

2) Se genera un evento invocando a una función Java-Script del motor AJAX.

3) El motor AJAX modifica la propiedad `location` del marco flotante para iniciar la comunicación.

4) El marco flotante envía una petición GET al servidor.

5) El servidor procesa la petición y envía una respuesta al marco flotante.

6) Una vez recibida la página dentro del marco flotante, éste notifica este suceso al motor AJAX enviando el nuevo contenido.

7) El motor AJAX procesa este contenido y actualiza la interfaz de usuario.

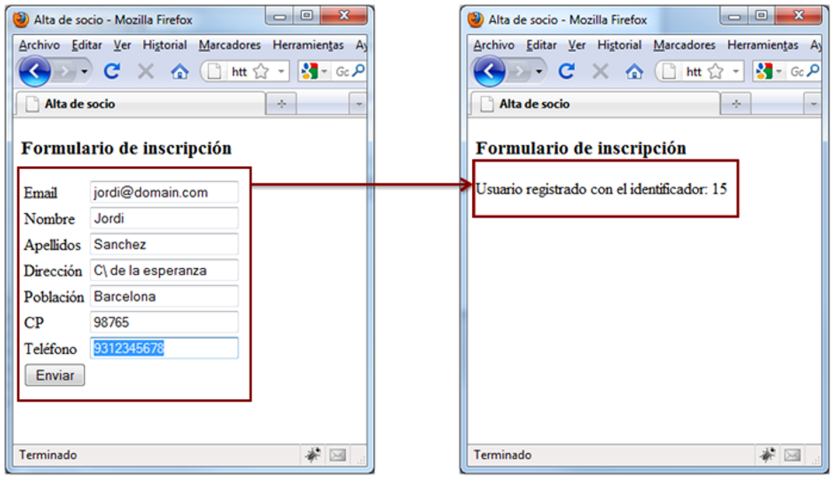
1.3.1. Ejemplo de marco flotante

Este ejemplo (ejemplo 6) consiste en un formulario de alta de un usuario con un conjunto de campos y un botón para enviarlos. Una vez que el usuario lo cumplimenta y hace clic sobre el botón "Enviar", se envían los datos al servidor. El servidor introduce los datos en la tabla Members de la base de datos y devuelve un mensaje con el resultado de la operación.

Ejemplo 6

La figura 7 muestra el resultado del envío del contenido del formulario.

Figura 7. Captura del ejemplo 5



El método HTTP utilizado en la petición es POST, y permitirá enviar de forma automática todos los campos del formulario en forma de parámetros.

Los diferentes ficheros que forman el ejemplo son los siguientes:

- *index.html*:

```
<html>
<head>
<title>Alta de socio</title>
<script type="text/javascript">
   function newUser(){
      document.myForm.submit();
   }

   function showResult(sResult){
      var div = document.getElementById('divForm');
      div.innerHTML = sResult;
   }

</script>
</head>
<body>
<h3>Formulario de inscripción</h3>
<iframe  name="hiddenFrame" width="0" height="0" frameborder="0"></iframe>
<div id="divForm">
<form name="myForm" method="post" action="saveData.php" target="hiddenFrame">
   <table>
      <tr><td>Email</td><td><input name="email" width="20" /></td></tr>
      <tr><td>Nombre</td><td><input name="name"/></td></tr>
      <tr><td>Apellidos</td><td><input name="surname"/></td></tr>
      <tr><td>Dirección</td><td><input name="address"/></td></tr>
      <tr><td>Población</td><td><input name="city"/></td></tr>
      <tr><td>CP</td><td><input name="postCode"/></td></tr>
      <tr><td>Teléfono</td><td><input name="phone"/></td></tr>
      <tr><td rowspan="3"><button onclick="newUser()">Enviar</button>
      </td></tr>
   </table>
</form>
</div>
</body>
</html>
```

En este documento reside el motor AJAX con dos funciones Java-Script: `newUser()`, para realizar la petición, y `showResult(sResult)`, para la visualización del nuevo contenido. La división divForm que engloba el formulario se utilizará para mostrar el resultado de la operación sustituyendo el formulario por la respuesta del servidor. También se declara el marco flotante con el nombre `"hiddenFrame"`, encargado de la comunicación asíncrona.

- *saveData.php*:

```
<html>
<head>
<script type="text/javascript">
   window.onload =
      function()
      {
         var div = document.getElementById("divResponse");
         parent.showResult(div.innerHTML);
      };
</script>
</head>
<body>
   <div id="divResponse">
      <?php
         $db_nombre = 'ajax';
         $sEmail = $_POST["email"];
         $sName = $_POST["name"];
         $sSurname = $_POST["surname"];
         $sAddress = $_POST["address"];
         $sCity = $_POST["city"];
         $sPC = $_POST["postCode"];
         $sPhone = $_POST["phone"];
         $sqlQuery = "insert into members"
         . " (Email,Name,Surnames,Address,City,PC,Phone)  Values ('$sEmail',"
         . " '$sName','$sSurname','$sAddress', '$sCity','$sPC','$sPhone')";
         $link = mysql_connect('localhost','ajax','J&J007')
         or die('Could not connect ' . mysql_errno());
         mysql_select_db($db_nombre ,$link)
         or die("Error seleccionando la base de datos.");
         if($oResult = mysql_query($sqlQuery))
         {
            $sResult = "Usuario registrado con el identificador: " . mysql_insert_id();
         }
         else
         {
            $sResult = "No se ha podido registrar el usuario";
         }
         echo $sResult;
         mysql_close($link);
      ?>
   </div>
</body>
</html>
```

Este documento contiene el *script* PHP para llevar a cabo el alta del usuario en la base de datos. Todos los campos que se registran se obtienen de los parámetros del cuerpo de la petición, a la que se accede desde el vector `$_POST`. El documento también dispone de una función Java-Script asignada al evento `window.onload` para notificar al motor AJAX la recepción de este contenido en el marco flotante oculto.

Los cambios más significativos en relación con la técnica de los marcos ocultos son dos. El primero es la definición del marco oculto. El marco oculto flotante puede declararse en el mismo documento en el que residen la interfaz y el motor AJAX.

El segundo es el acceso al documento que contiene la función de actualización del contenido desde el marco oculto. En un marco tradicional oculto, se debía acceder al nivel superior del objeto del documento y, a partir de ahí, buscar el marco visible tal como sigue:

```
parent.frames["visibleFrame"].javaScriptFunction();
```

En este caso, el marco flotante se encuentra en el mismo documento en el que reside la función `showResult()`. Basta con acceder al documento con el objeto `parent` y acceder así a la función:

```
parent.showResult(div.innerHTML);
```

1.4. XMLHttpRequest

Aunque el término AJAX no fue introducido hasta el año 2005, anteriormente ya se hacía uso de las técnicas de marcos ocultos para obtener información del servidor de forma asíncrona. Paralelamente, los navegadores empezaron a incorporar un objeto llamado XMLHttpRequest que permitía realizar las mismas tareas de una manera más eficiente.

XMLHttpRequest

XMLHttpRequest fue ideado inicialmente por Microsoft y se incluyó en su navegador Internet Explorer 5.

XMLHttpRequest es una interfaz a la que se accede desde Java-Script que está implementada por la mayoría de los navegadores, y que permite realizar peticiones HTTP. Dispone de un conjunto de mecanismos para el envío y la recepción de peticiones síncronas y asíncronas. El objeto XMLHttpRequest fue ideado para intercambiar información en formato XML[4], pero actualmente soporta diferentes formatos, como texto ASCII, HTML y JSON[5].

(4)XML es la sigla de *extensible markup language* ('lenguaje de marcado extensible').

(5)JSON es la sigla de *Java-Script object notation.*

Navegadores y XML

La mayoría de los navegadores actuales ofrecen una implementación nativa de este objeto. Es el caso de Mozilla Firefox, Opera, Google Chrome, Konkeror, Safari y Microsoft Internet Explorer a partir de su versión 7.

La interfaz XMLHttpRequest es regulada por el W3C con especificaciones definidas en dos niveles:

1) XMLHttpRequest Level 1. Su primer borrador fue presentado en abril del 2006 con la especificación estándar.

2) XMLHttpRequest Level 2. La primera versión del borrador de especificación fue presentada en febrero del 2008 e incluye nuevas funciones: eventos de progreso de peticiones, peticiones entre diferentes dominios y manejo de flujo de *bytes* para el envío y la recepción de datos.

Al no existir una definición definitiva, los desarrolladores deben tener en cuenta las diferentes implementaciones de esta interfaz en cada navegador. Estas diferencias pueden ser minimizadas utilizando entornos de desarrollo[6] que ofrezcan una capa de abstracción de las diferentes implementaciones.

[6]En inglés, *frameworks*.

Entornos de desarrollo

Existen entornos de desarrollo (*frameworks*), usualmente ficheros Java-Script, que se ejecutan en el lado del cliente y que facilitan el desarrollo web ampliando el DOM, ofreciendo funcionalidades AJAX, etc.

1.4.1. Propiedades

La siguiente tabla muestra las propiedades de XMLHttpRequest:

Propiedades	Descripción
readyState	Indica el estado actual del objeto.
responseText	Devuelve la respuesta del servidor en una cadena de texto.
responseXML	Devuelve la respuesta obtenida en un objeto DOM XML.
responseBody	Devuelve la respuesta del servidor en un vector de *bytes*.
status	Proporciona el código de estado de la respuesta.
statusText	Devuelve el estado de la respuesta con una explicación textual.

Propiedades de XMLHttpRequest

La propiedad *responseBody* fue añadida en la especificación XMLHttpRequest Level 2. El resto de propiedades se definieron en la especificación de nivel 1.

Una vez enumeradas las propiedades de XMLHttpRequest, pasamos a explicar en detalle cada una.

- `readyState`:

Un objeto XMLHttpRequest pasa por diferentes estados durante la creación y el intercambio de datos con el servidor HTTP. Esta propiedad proporciona el estado del objeto en cada momento. Los posibles estados son los siguientes:

Valor	Estado	Descripción
0	UNSENT	Indica que el objeto XMLHttpRequest ha sido creado pero no inicializado.
1	OPENED	Indica que el objeto está inicializado y preparado para enviar una petición.
2	HEADERS_RECEIVED	Todas las cabeceras de la respuesta ya han sido recibidas y algunos miembros del objeto ya están disponibles.
3	LOADING	Se está recibiendo el cuerpo de la respuesta.
4	DONE	La transacción se ha completado.

Las funciones Java-Script que controlan el envío y la recepción de datos pueden comprobar este valor para conocer el estado de una petición.

- *responseText*:

Devuelve el contenido de la respuesta HTTP en formato de texto siempre y cuando el valor de la propiedad `readyState` sea 4 `(DONE)` o 3 `(LOADING)`. En el primer caso se obtendrá toda la respuesta y en el segundo, el contenido recibido hasta el momento.

- *responseXML*:

Proporciona un objeto DOM del documento XML obtenido. A diferencia de lo que pasa con la propiedad `responseText`, con esta propiedad sólo se podrá obtener el resultado hasta recibir la respuesta completa: cuando la propiedad `readyState` valga 4 `(DONE)`.

- *responseBody*:

Proporciona la respuesta en una matriz[7] de *bytes* sin signo. Puede ser útil para obtener datos binarios en una respuesta.

(7)En inglés, *array*.

- *status*:

Obtiene el código de estado de la respuesta HTTP. Si la operación es correcta, devolverá el código 200; si la página no existe devolverá 404, etc.

Ved también

Ved los códigos de respuestas HTTP más comunes en el módulo "Introducción a AJAX" de esta asignatura.

- *statusText*:

Devuelve el código de respuesta en formato textual. Por ejemplo, si el código de respuesta es 200, el texto será “OK”.

1.4.2. Métodos

Los diferentes métodos del objeto XMLHttpRequest se muestran en la tabla siguiente:

Método	Descripción
open	Prepara el objeto para una petición HTTP.
send	Envía la petición al servidor.
abort	Cancela la petición en curso.
setRequestHeader	Permite añadir una cabecera a la petición HTTP.
getAllResponseHeaders	Obtiene todas las cabeceras HTTP de la última petición en una cadena de caracteres.
getResponseHeader	Obtiene la cabecera HTTP que se específica por parámetro.

A continuación, explicamos cada uno de los métodos enumerados.

- *open (método, url, async, usuario, password)*:

Inicializa el objeto XMLHttpRequest y lo prepara para enviar una petición HTTP. Los parámetros son los siguientes:

- *método*. Requerido. Es la cadena (*string*) que indica el método de la petición HTTP (GET, POST, etc.).

- *url*. Requerido. Es la cadena que indica el URL al que se dirige la petición.

- *async*. Opcional. Es un valor booleano que indica si la petición es asíncrona o no. Por defecto, la petición se realiza de forma asíncrona (*true*).

- *usuario* y *password*. Opcionales. Son cadenas que indican el usuario y la contraseña en el caso de que el servidor requiera autenticación.

Si la llamada tiene éxito, el estado del objeto pasa a ser 1 `(OPENED)`.

- *send([body])*:

Envía la petición previamente preparada por el método `open()`. El estado del objeto antes de llamar a `send()` debe ser 1 `(OPENED)`. En el caso contrario, lanzará una excepción.

Si en el método `open()` se estableció el parámetro `async` como verdadero, esta llamada devolverá el control de inmediato. En caso contrario, se bloqueará el proceso hasta que se obtenga una respuesta.

El parámetro `body` permite añadir información en el cuerpo de la petición. Si la petición no requiere ningún cuerpo, se especificará `null` o se omitirá el parámetro.

- *abort()*:

Cancela una petición en curso estableciendo el estado del objeto a 0 `(UNSENT)`.

- *setRequestHeader(header, value)*:

Toda petición HTTP incorpora un conjunto de líneas opcionales que aportan información adicional al destinatario, sea el cliente o el servidor. Estas líneas se denominan cabeceras[8] y dotan al protocolo HTTP de una gran flexibilidad pa-

(8) En inglés, *headers*.

ra intercambiar datos sobre la transacción: información sobre el tipo de contenido, hora y fecha de la petición, información sobre el navegador del cliente, etc. Las cabeceras están compuestas por un nombre, seguido de dos puntos (:) y el valor.

Ved también

Para más información sobre el protocolo HTTP, ved el apartado 2.1 del módulo "Introducción a AJAX" de esta asignatura.

La línea siguiente define una cabecera de respuesta de un servidor indicando el tipo de contenido devuelto al cliente.

```
Content-Type: text/html; charset=ISO-8859-1
```

Esta cabecera indica el tipo de contenido, que en este caso es una página HTML, y su correspondiente codificación de caracteres.

El método `setRequestHeader(header, value)` permite establecer en un objeto XMLHttpRequest una cabecera y su valor. Esta cabecera se incorporará en la siguiente petición al llamar a `send()`. Para añadir cabeceras, el estado del objeto debe ser 1 `(OPENED)`.

- *`getAllResponseHeaders()`*:

Obtiene todas las cabeceras de la respuesta HTTP en una cadena. Si el objeto no está en estado 3 `(LOADING)` o 4 `(DONE)`, se devuelve `null`.

- *`getResponseHeader(header)`*:

Devuelve una cadena con el contenido de la cabecera pasado por parámetro. Si el objeto no está en estado 3 `(LOADING)` o 4 `(DONE)`, se devuelve `null`.

1.4.3. Eventos

La tabla siguiente muestra los eventos que permiten notificar los cambios de estado y los sucesos que ocurren en una transacción HTTP:

Evento	Evento lanzado cuando...
onreadystatechange	... la propiedad *readyState* cambia de valor.
onloadstart	... la petición se inicia.
onprogress	... mientras se cargan y se envían datos en el transcurso de un diálogo.
onabort	... se cancela anormalmente la petición.
onerror	... la petición ha fallado.
onload	... la petición se ha completado satisfactoriamente.

Eventos

Todos los eventos fueron introducidos en la especificación **XMLHttpRequest Level 2**, salvo `onreadystatechange`, que se definió en la primera especificación.

Evento `onreadystatechange`

El evento `onreadystatechange` permite capturar todos los cambios de estado del objeto XMLHttpRequest. El resto de eventos tienen un carácter más específico.

Antes de realizar el envío de una petición, se deberán asignar las diferentes funciones Java-Script encargadas de gestionar los sucesos a los eventos* que se quieran capturar.

1.4.4. Uso del objeto XMLHttpRequest

Para realizar una petición asíncrona con el servidor utilizando la clase XMLHttpRequest, se deben llevar a cabo los pasos siguientes:

1) Creación de una instancia de XMLHttpRequest.

2) Inicialización.

3) Asignación de una función para gestionar los cambios de estado de la instancia de XMLHttpRequest.

4) Configuración de la cabecera de la petición.

5) Envío de la petición.

6) Recepción y gestión del nuevo contenido.

A continuación, describimos en detalle cada uno de estos pasos.

Creación de una instancia de XMLHttpRequest

El primer paso es crear una instancia de XMLHttpRequest. Según el navegador, la creación puede variar: Internet Explorer proporciona el objeto XMLHttpRequest como un control ActiveX hasta la versión 6 del navegador; en cambio, Firefox, Chrome y Safary ofrecen soporte nativo, y Microsoft® Internet Explorer ofrece de forma nativa el objeto XMLHttpRequest a partir de la versión 7.

Ejemplo 7

La siguiente función (ejemplo 7) crea y devuelve una instancia de XMLHttpRequest válida para múltiples navegadores.

```
function createXMLHttpRequestObject() {
   /*Esta condición determina si el navegador tiene soporte
    nativo para XMLHttpRequest*/
   if(window.XMLHttpRequest){
      return new XMLHttpRequest();
   }
   /*No hay soporte nativo, se intenta obtener como un objeto ActiveX
    para versiones anteriores de Internet Explorer 7*/
   else if(window.ActiveXObject){
      try{
         return new ActiveXObject('Msxml2.XMLHTTP');
      }
      catch(e){
         try{
            return new ActiveXObject('Microsoft.XMLHTTP');
         }
         catch(E){}
      }
   }
   alert('No se ha podido crear una instancia de XMLHttpRequest');
}
```

La primera condición evalúa si existe una implementación nativa de XMLHttpRequest. Si es así, crea una instancia. Si no existe dicha implementación, se comprueba que exista la clase ActiveXObject para crear una instancia de XMLHttpRequest para versiones de Internet Explorer anteriores a la 7.

En los ejemplos siguientes, todas las instancias de XMLHttpRequest partirán de esta función.

Inicialización

Una vez creada una instancia de XMLHttpRequest, es necesario inicializarla. Para ello se llamará al método `open` indicando el método HTTP que se empleará (GET o POST) y el URL que señala el recurso que se debe obtener.

```
var url = "urlToResource";
oXMLHttpObject = createXMLHttpRequestObject();
oXMLHttpRequest.open('GET', url);
```

Opcionalmente, puede especificarse un tercer parámetro booleano que indique si se desea una comunicación asíncrona o síncrona. El cuarto y el quinto parámetros, también opcionales, permiten establecer el nombre de usuario y la contraseña en caso de que el servidor HTTP requiera autenticación.

Tercer parámetro

Si se establece *false* en el tercer parámetro, el método `send` bloquea la ejecución hasta que finaliza la transacción.

Si la operación se ha llevado a cabo de forma correcta, el objeto está inicializado y pasa a estar en el estado 1 `(OPENED)`.

Asignación de una función para gestionar los cambios de estado

Para gestionar la transacción HTTP, se debe asignar una función Java-Script a los diferentes eventos. La propiedad `onreadystatechange` permite capturar todos los cambios de estado.

Una vez que se ha completado con éxito, la siguiente función permite gestionar una respuesta.

```
oXMLHttpObject.onreadystatechange = function(){
   if(oXMLHttpObject.readyState == 4){
      if(oXMLHttpObject.status == 200){
      //gestión de la respuesta
      }
      else{
         //error
      }
   }
}
```

Para saber si los datos han sido recibidos, la función comprueba el valor de la propiedad `readyState`. Si su valor es 4 `(DONE)`, ello implicará que la transacción ha finalizado. Para saber si ha tenido éxito, la función consulta en una segunda condición si el código del estado de la respuesta es 200 `(OK)`. En ese caso se realizará la gestión del contenido.

También se pueden asignar funciones a eventos para manejar estados y sucesos más concretos.

```
oXMLHttpObject.onload = function(){
   if(oXMLHttpObject.status == 200){
      //gestión de la respuesta
   }
};

oXMLHttpObject.onerror = function(){
   alert("XMLHttpRequest: Error en la petición HTTP.");
};
```

El ejemplo anterior asigna sendas funciones a los eventos `onload` y `onerror`. La primera será invocada si la respuesta se recibe correctamente; la segunda, en el caso contrario.

Configuración de la cabecera de la petición

Antes del envío de la petición, se deben establecer las cabeceras adicionales si fuera necesario. Por ejemplo, las líneas siguientes modifican la cabecera User-Agent, que proporciona al servidor información sobre el cliente.

```
oXMLHttp.setRequestHeader('User-Agent', 'AJAX');
```

Envío de la petición

El objeto XMLHttpRequest está preparado para realizar la comunicación, y para iniciarla se llama al método `send`.

```
oXMLHttpObject.send(null);
```

Recepción y gestión del nuevo contenido

Los manejadores de eventos gestionarán la respuesta una vez enviada la petición. En el caso de que la recepción se haya completado con éxito, se puede acceder a las propiedades `responseText` o `responseXML` para acceder así a los datos recibidos.

Manejadores de eventos

Los manejadores de eventos son las funciones Java-Script asignadas a los diferentes eventos. Éstos se establecieron en el paso 3.

Ejemplo mediante peticiones GET

El ejemplo 8 permite consultar los usuarios registrados en el ejemplo anterior de marcos flotantes.

Consta de una página que permite consultar la información de un socio a partir de su número de abonado. Al introducir el número de abonado en una caja de texto y hacer clic sobre un botón, se solicitará la información del abonado mediante una petición asíncrona y se actualizará la página mostrando una tabla con los datos. Los archivos de este ejemplo son los siguientes:

- *index.html*:

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Search member form</title>
<script type="text/javascript" src="../script/Ajax.js"></script>
<script type="text/javascript">

   function searchMember(){
      //Se obtiene el objeto XMLHttpRequest
      var oXMLHttpRequest = createXMLHttpRequestObject();
      //Se obtiene el identificador del usuario
      var id = document.getElementById("memberNumber").value;
      var div = document.getElementById("divRequest");
```

```
      //Se construye el URL con un parámetro indicando el identificador
         var url = "getMember.php?id=" + id;
         //Se establece el método GET y el URL anterior
         oXMLHttpRequest.open('GET', url);
         //Se asigna la función que gestionará los cambios de estado
         oXMLHttpRequest.onreadystatechange = function(){
            if(oXMLHttpRequest.readyState == 4){
               if(oXMLHttpRequest.status == 200){
                  div.innerHTML = oXMLHttpRequest.responseText;
               }
               else{
                  alert("There was a problem with the request.");
               }
            }
         }
         //Se envía la petición
         oXMLHttpRequest.send(null);
      }
</script>
</head>
<body>
   N° Socio <input id="memberNumber" width="20" />
   <button onclick="searchMember()">Search</button><br><br>
   <div id="divRequest"></div>
</body>
</html>
```

Esta es la página principal del ejemplo con la interfaz de usuario y el motor AJAX. La interfaz consta de una caja de texto y de un botón para consultar el número de un socio.

El motor AJAX contiene la función `searchMember()`, que crea y prepara una instancia de XMLHttpRequest.

- *`getMember.php`*:

```
<?php
   header('Content-Type: text/html; charset=ISO-8859-1');
   $db_nombre = 'ajax';
   //Se obtiene el identificador del usuario del parámetro en el URL
   $idMember = $_GET["id"];
   $sqlQuery = "Select * from members Where nMember=\"" .
   $idMember."\"";
   $link = mysql_connect('localhost','ajax','J&J007') or
   die('Could not connect ' . mysql_errno());
   mysql_select_db($db_nombre ,$link) or die("Error selecting db.");
   //Si hay resultados en la consulta SQL...
```

```
    if($result = mysql_query($sqlQuery))
    {
       $total_rows = mysql_num_rows($result);
       //se devuelve la información del usuario en una tabla
       if($total_rows >= 1)
       {
          $row = mysql_fetch_assoc($result);?>
          <table  BORDER="1">
             <tr><td>Email: </td>
             <td><?php echo $row["Email"]?></td></tr>
             <tr><td>Name: </td>
             <td><?php echo $row["Name"]?></td></tr>
             <tr><td>Surname: </td>
             <td><?php echo $row["Surnames"]?></td></tr>
             <tr><td>Address: </td>
             <td><?php echo $row["Address"]?></td></tr>
             <tr><td>City: </td>
             <td><?php echo $row["City"]?></td></tr>
             <tr><td>CP: </td>
             <td><?php echo $row["PC"]?></td></tr>
             <tr><td>Phone: </td>
             <td><?php echo $row["Phone"]?></td></tr>
          </table><?php
       }
       else
       {
          echo "Member not found";
       }
    }
    mysql_close($link);
 ?>
```

Esta página consulta el socio de la base de datos a partir del identificador pasado por el URL. Una vez obtenido, construye una tabla que será devuelta al cliente.

A continuación, detallamos paso a paso el proceso de obtención de información de un socio.

1) Carga de la página inicial *index.html*

La página muestra la caja de texto en la que el cliente podrá indicar el número de socio y el botón sobre el que tendrá que hacer clic para enviar la consulta. El atributo `onclick` se establece en la función `searchMember()`, que iniciará la petición. Para representar los datos obtenidos, se declara una división con el identificador `divRequest`.

```
<body>
   N° Socio
   <input id="memberNumber" width="20" />
   <button onclick="searchMember()">Search</button><br><br>
   <div id="divRequest"></div>
</body>
```

2) Envío de la petición GET

El usuario ha introducido un número de socio y ha hecho clic sobre el botón para que se llame a la función `searchMember()`. Esta función crea el objeto XMLHttpRequest y prepara el URL formado por la página `getMember.php` y el parámetro `id` que se obtiene de la caja de texto. A continuación, llama al método `open()` indicando que la comunicación se realizará con el método GET y asigna el URL. Antes de enviar la petición, se establece en la propiedad `onreadystatechange` una función Java-Script para gestionar los cambios de estado del objeto.

Finalmente, se inicia la comunicación llamando al método `send()`.

```
var id = document.getElementById("memberNumber").value;
var div = document.getElementById("divRequest");
var url = "getMember.php?id=" + id;
oXMLHttpRequest.open('GET', url);
oXMLHttpRequest.onreadystatechange = function(){
   ...
}
oXMLHttpRequest.send(null);
```

3) Generación de página y respuesta del servidor

El servidor recibe la petición y construye la página `getMember.php`. Inicialmente, se establece el tipo de contenido del documento que hay que enviar, que será de tipo texto (`text/html`) mediante la función *php* `header()`. A continuación, se obtiene el identificador del socio del URL y se realiza la consulta a la base de datos. Si el socio existe, se devuelve una tabla con su información; en el caso contrario, se devuelve un mensaje de error.

4) Recepción y actualización de página

Cada vez que el objeto XMLHttpRequest cambie de estado, se ejecutará la función asignada a la propiedad `onreadystatechange`. Cuando se obtenga la respuesta del servidor, esta función actualizará la división siempre y cuando el objeto esté en el estado 4 (`loaded`) y el estado de la respuesta sea 200 (`OK`). En ese caso se asignará a la división el contenido de la respuesta obtenido a partir de la propiedad `responseText`, mostrando así el resultado de la consulta.

```
oXMLHttpRequest.onreadystatechange = function(){
   if(oXMLHttpRequest.readyState == 4){
      if(oXMLHttpRequest.status == 200){
         div.innerHTML = oXMLHttpRequest.responseText;
      }
      else{
         alert("There was a problem with the request.");
      }
   }
}
```

Ejemplo mediante el método POST

Este ejemplo (ejemplo 9) realiza un alta de usuario al igual que el ejemplo anterior de marcos ocultos flotantes. El intercambio de información se realiza con POST, usando XMLHttpRequest.

Las peticiones POST que contienen campos dentro de un formulario deben tratarse de forma especial. Al llamar al método `submit` del formulario, éste lleva a cabo la petición automáticamente. En este caso, el objeto XMLHttpRequest es el que ha de realizar la petición. Para ello, se tendrá que enviar la petición añadiendo los campos del formulario dentro del cuerpo de la misma.

Básicamente, el ejemplo consiste en una página HTML que mostrará un formulario para dar de alta a un nuevo socio. El formulario está compuesto por diferentes cajas de texto y un botón para iniciar el envío de información.

Los ficheros son los siguientes:

- *index.html*:

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Add member form</title>
<script type="text/javascript" src="../script/AJAX.js"></script>
<script type="text/javascript">

function newUser(){
   var oXMLHttpObject = createXMLHttpRequestObject();
   var form = document.forms[0];
   var div = document.getElementById("divRequest");
   oXMLHttpObject.open('post', form.action);
   oXMLHttpObject.setRequestHeader("Content-Type",
   "application/x-www-form-urlencoded");
   oXMLHttpObject.onreadystatechange = function(){
      if(oXMLHttpObject.readyState == 4){
```

```
            if(oXMLHttpObject.status == 200){
               div.innerHTML = oXMLHttpObject.responseText;
            }
            else{
               alert("There was a problem with the request.");
            }
         }
      }
      oXMLHttpObject.send(getFormParams(form));
   }

   function getFormParams(form){
      var params = new Array();
      for(var i = 0; i < form.elements.length; i++){
         var param = encodeURIComponent(form.elements[i].name);
         param += "=";
         param += encodeURIComponent(form.elements[i].value);
         params.push(param);
      }
      return params.join("&");
   }
   </script>
   </head>
   <body>
      <form method="post" action="saveData.php" onsubmit="newUser(); return false">
      <table>
         <tr><td>Email</td><td><input name="email" width="20" /></td></tr>
         <tr><td>Nombre</td><td><input name="name"/></td></tr>
         <tr><td>Apellidos</td><td><input name="surname"/></td></tr>
         <tr><td>Dirección</td><td><input name="address"/></td></tr>
         <tr><td>Población</td><td><input name="city"/></td></tr>
         <tr><td>CP</td><td><input name="postCode"/></td></tr>
         <tr><td>Teléfono</td><td><input name="phone"/></td></tr>
      </table>
      <input type="submit" value="Save member" /><br>
      </form>

      <div id="divRequest"></div>
   </body>
   </html>
```

Es la página principal, que consta de un formulario y un conjunto de campos para cumplimentar.

- *saveData.php*:

```
<?php
```

```
    header('Content-Type: text/html; charset=ISO-8859-1');
    $db_nombre = 'ajax';
    $sEmail = $_POST["email"];
    $sName = $_POST["name"];
    $sSurname = $_POST["surname"];
    $sAddress = $_POST["address"];
    $sCity = $_POST["city"];
    $sPC = $_POST["postCode"];
    $sPhone = $_POST["phone"];
    $sqlQuery = "Insert into members"
    . " (Email,Name,Surnames,Address,City,PC,Phone) Values('$sEmail',"
    . " '$sName','$sSurname','$sAddress', '$sCity','$sPC','$sPhone')";
    $link = mysql_connect('localhost','ajax','J&J007')
    or die('Could not connect ' . mysql_errno());
    mysql_select_db($db_nombre ,$link) or die("Error selecting db.");
    if($oResult = mysql_query($sqlQuery))
    {
       $sResult = "Usuario registrado con el identificador: " . mysql_insert_id();
    }
    else
    {
       $sResult = "No se ha podido registrar el usuario";
    }
    echo $sResult;
    mysql_close($link);
?>
```

Obtiene los diferentes parámetros del formulario e intenta crear un nuevo socio devolviendo el resultado de la operación al cliente.

Una vez presentados los ficheros que forman el ejemplo, explicaremos cada uno de los pasos involucrados en el alta de un usuario.

1) Carga inicial de la página *index.html*

El usuario carga la página *index.html*, en la que se muestra un formulario para introducir toda la información de un nuevo socio.

```
<form method="post" action="saveData.php" onsubmit="newUser(); return false">
<table>
   <tr><td>Email</td><td><input name="email" width="20" /></td></tr>
   <tr><td>Nombre</td><td><input name="name"/></td></tr>
   <tr><td>Apellidos</td><td><input name="surname"/></td></tr>
   <tr><td>Dirección</td><td><input name="address"/></td></tr>
   <tr><td>Población</td><td><input name="city"/></td></tr>
   <tr><td>CP</td><td><input name="postCode"/></td></tr>
   <tr><td>Teléfono</td><td><input name="phone"/></td></tr>
```

```
</table>
<input type="submit" value="Save member" /><br>
</form>
<div id="divRequest"></div>
```

En el formulario se establece la propiedad `action` hacia la página `saveData.php`. El evento `onsubmit` llamará a la función `newUser()` para iniciar la comunicación y, acto seguido, devolverá `false` para evitar que el formulario envíe la petición, puesto que es el objeto XMLHttpRequest el que lo debe hacer.

Para representar la respuesta del servidor, se declara una división con el identificador `divRequest` después del formulario.

2) Envío de la petición POST

El usuario ha introducido los datos en el formulario y ha hecho clic sobre "Save member" provocando una llamada a la función `newUser()`. La segunda línea obtiene el objeto del formulario en la variable `form`.

```
var form = document.forms[0];
```

Posteriormente, se prepara el objeto `oXMLHttpRequest` para realizar la petición llamando al método `open()`.

```
oXMLHttpObject.open('post', form.action);
```

Como método, se establece `post`, y el URL se obtiene del atributo `action` del formulario, que en este caso es `saveData.php`.

En la línea siguiente se indica el tipo de contenido usado para enviar el formulario al servidor como `application/x-www-form-urlencoded`.

```
oXMLHttpObject.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
```

Luego se asigna una función en la propiedad `onreadystatechange` para gestionar la recepción de la respuesta.

Al realizar el envío de la petición, se deben enviar, en el cuerpo, todos los campos del formulario. La función `getFormParams(form)` devuelve una cadena con los campos y los valores del formulario pasado por parámetro.

```
function getFormParams(form){
   var params = new Array();
   for(var i = 0; i < form.elements.length; i++){
      var param = encodeURIComponent(form.elements[i].name);
      param += "=";
```

```
        param += encodeURIComponent(form.elements[i].value);
        params.push(param);
    }
    return params.join("&");
}
```

Esta función recorre todos los campos del formulario y añade en una matriz el nombre del campo seguido de su valor con el formato siguiente:

```
NombreCampo=ValorCampo.
```

Finalmente, se utiliza la función *join*, que devuelve una cadena con todos los elementos de la matriz concatenados utilizando el carácter pasado por parámetro. De esta forma, el conjunto de parámetros quedaría de la manera siguiente:

```
NombreCampo1=Valor1&amp;NombreCampo2=Valor2&amp;...
```

Finalmente, se llama al método `send()` pasando por parámetro la cadena de caracteres que representa el conjunto de campos del formulario que deben enviarse en el cuerpo de la petición.

```
oXMLHttpObject.send(getFormParams(form));
```

3) Generación de página y respuesta del servidor

Al recibir la petición, el servidor HTTP ejecuta la página `addMember.php`. Ésta obtiene los diferentes parámetros, los inserta en la base de datos y devuelve al cliente el resultado de la operación.

4) Recepción y actualización de página

Al igual que en el ejemplo anterior, la función asignada a la propiedad `onreadystatechange` comprueba que se hayan recibido correctamente los datos y los traslada a la división reservada para ello.

1.4.5. La política del mismo origen

La política del mismo origen es una medida de seguridad sobre Java-Script de la que disponen los navegadores y que determina los URL válidos a los que puede acceder una aplicación web. Esta política previene que una aplicación cargada desde un dominio pueda obtener datos situados en otros dominios.

Política del mismo origen

Todos los navegadores, incluidos Firefox, Chrome e Internet Explorer, disponen de esta medida de seguridad.

Esto evita que una página web con código malicioso pueda llevar a cabo ataques o comprometer la confidencialidad de otra página web.

Zonas de seguridad

Internet Explorer utiliza otra medida, llamada **zonas de seguridad**, por la que un conjunto de sitios que cumplen ciertas normas están exentos de la política del mismo origen.

Los navegadores basados en Mozilla consideran que dos páginas tienen un mismo origen cuando el protocolo, el puerto y el dominio son iguales. La tabla siguiente muestra una serie de URL y si se puede acceder a ellos desde el origen http://multimedia.uoc.edu/AJAX/index.html:

URL	Resultado	Razón
https://multimedia.uoc.edu/index.html	Fallo	Protocolo diferente
http://multimedia.uoc.edu:8080/index.html	Fallo	Puerto diferente
http://www.otrodominio.com	Fallo	Dominio diferente
http://multimedia.uoc.edu/index.html	Correcto	
http://multimedia.uoc.edu/info/index.html	Correcto	

La política del mismo origen también afecta a las peticiones realizadas desde objetos XMLHttpRequest, en los que los URL se limitan al mismo dominio del que proviene la aplicación. Esta restricción es un gran inconveniente para muchos desarrolladores que necesitan obtener datos desde dominios cruzados.

Una posible solución es la de utilizar *proxies* en el lado del servidor, que redirigen las peticiones a servidores en otros dominios y envían de vuelta la respuesta a la aplicación.

El W3C publicó una especificación llamada CORS que consiste en el intercambio de cabeceras específicas para informar al navegador de los diferentes dominios y métodos HTTP a los que puede acceder.

CORS

CORS es la sigla de Cross-Origin Resource Sharing. Firefox 3.5 y Safary 4 la implementan. Internet Explorer 8 implementa una parte. La especificación CORS se puede consultar por Internet en:

- Cross-Origin Resource Sharing

1.5. Ventajas e inconvenientes de las diferentes técnicas

Los navegadores guardan en su historial de navegación las diferentes peticiones que se van realizando durante la navegación. De esta manera, el usuario puede retroceder a URL ya visitados, y también avanzar por éstos. Los URL

generados desde marcos ocultos para comunicaciones asíncronas también se almacenan en el historial, de forma que el usuario puede volver atrás después de que se realice una recarga parcial.

Gmail

Gmail utiliza la técnica de los marcos ocultos, de manera que puede navegar hacia atrás y hacia delante por las diferentes acciones realizadas.

Un inconveniente del uso de XMLHttpRequest es que las peticiones generadas desde una instancia no se almacenarán en el historial, por lo que no es posible navegar entre las diferentes peticiones realizadas. Si un requisito importante es la navegación entre peticiones, habrá que optar por el uso de marcos ocultos.

Otra desventaja son las diferencias entre las implementaciones de esta interfaz en los diferentes navegadores. Crear una aplicación web compatible con los navegadores más populares requiere más lógica, un inconveniente que puede ser contrarrestado con el uso de entornos de desarrollo.

Entornos de desarrollo

Un entorno de desarrollo (en inglés, *framework*) es un conjunto de componentes de software utilizados para facilitar el desarrollo de ciertas funciones. En este caso, los entornos de desarrollo de AJAX se pueden utilizar en un proyecto web para añadir funcionalidades AJAX.

Por otro lado, XMLHttpRequest permite gestionar todos los aspectos de la comunicación con un único objeto: cambio de cabeceras y método de la petición, seguimiento de los cambios de estado de la comunicación, código de respuesta del servidor, etc. XMLHttpRequest está pensado para esta labor y dispone de un amplio conjunto de propiedades, métodos y eventos que no ofrecen los marcos ocultos.

WAI

WAI es la sigla de Web Accessibility Initiative ('Iniciativa para la Accesibilidad Web'). Puede consultarse por Internet en:

- Web Accessibility Initiative (WAI)
- XML Path Language (XPath)

1.6. AJAX accesible

La accesibilidad permite el acceso a la Web y a sus contenidos a personas con discapacidad visual, motriz, auditiva, cognitiva o neuronal. La WAI es una iniciativa del W3C que se encarga de establecer las especificaciones y las tecnologías necesarias para una correcta accesibilidad de la Web para personas con discapacidad y gente de edad avanzada. Esta iniciativa provee de estrategias, guías y recursos para la creación de aplicaciones web accesibles.

Tecnologías de apoyo

La expresión *tecnologías de apoyo* (*assistive technologies*) se refiere a dispositivos, instrumentos, tecnologías o software que se utilizan para incrementar las capacidades funcionales de personas con discapacidad.

La mayoría de sitios web no se ajustan a las medidas de esta iniciativa, lo que dificulta el acceso a la Web a la mayoría de gente con discapacidades. A medida que la Web crece rápidamente, se va haciendo cada vez más esencial que ésta sea accesible, de forma que haya una igualdad en el acceso y las oportunidades para todos. Por ello, la accesibilidad no sólo depende de las tecnologías de apoyo, como programas de lectura de pantalla, sintetizadores de voz, líneas braille, etc. La mayor responsabilidad reside en los desarrolladores web y en el software utilizado para producir y evaluar la accesibilidad.

Uno de los objetivos de la WAI es el desarrollo de guías y técnicas que describan soluciones de accesibilidad para aplicaciones web. Estas guías son consideradas un estándar internacional para la accesibilidad web.

Enlaces de interés

Este enlace muestra los pasos básicos para llevar a cabo proyectos web con accesibilidad:

Implementation Plan for Web Accessibility

Este otro enlace muestra información detallada para desarrolladores:

Web Content Accessibility Guidelines (WCAG) Overview

Para valorar la accesibilidad y detectar problemas relacionados con ella, existen herramientas que ayudan en la evaluación de una aplicación web. Aun así, la evaluación humana es imprescindible.

WAI-ARIA define la manera de hacer más accesibles las aplicaciones web enriquecidas que disponen de contenidos dinámicos y una interfaz de usuario avanzada con controles desarrollados con AJAX y tecnologías relacionadas. Estos controles deben interactuar con las tecnologías de apoyo que utilizan las personas con discapacidades. Por ejemplo, la actualización dinámica de contenido en una página puede no ser detectada por un lector de pantalla que utilice una persona invidente.

WAI-ARIA

WAI-ARIA es la sigla de Web Accessibility Initiative - *Access rich Internet applications* ('Iniciativa para la Accesibilidad Web - Acceso a aplicaciones de Internet enriquecidas'). Se encarga de la iniciativa para la accesibilidad de aplicaciones RIA.

1.6.1. Ejemplo de AJAX accesible

WAI-ARIA define el término regiones activas[9] AJAX, que permiten que las tecnologías de apoyo se informen de actualizaciones de contenidos mediante atributos HTML.

(9)En inglés, *live regions*.

Algunas de estas propiedades son `aria-live` y `aria-atomic`.

1) La propiedad `aria-live`

Esta propiedad indica la prioridad con la que una tecnología de apoyo debe tratar las actualizaciones en regiones activas. Los valores posibles son los siguientes:

- `off`. Es el valor por defecto e indica que la región no es activa.
- `polite`. Indica a la tecnología de apoyo que se trata de una actualización normal y que no es necesaria su lectura hasta que el usuario complete su actividad actual.
- `assertive`. Este valor indica una prioridad alta, pero no es necesario interrumpir la actividad actual del usuario.
- `rude`. Éste es el valor más alto, e indica que la actividad del usuario ha de ser interrumpida cuando se actualice el contenido.

Valor `rude`

El valor `rude` no es recomendable si no es extremadamente necesario, ya que puede desorientar al usuario si está llevando a cabo otra tarea.

El siguiente fragmento de documento muestra dos divisiones con comportamientos diferentes al actualizarse.

```
<div id="NoticiaUltimaHora" aria-live="rude">
</div>
<div id="chat" aria-live="polite">
</div>
```

La primera división interrumpirá la actividad del usuario al actualizarse. La segunda división representa un *chat*, y todos los elementos que se vayan añadiendo, por ejemplo párrafos, se comunicarán de forma normal cuando el usuario acabe su actividad actual.

2) La propiedad `aria-atomic`

Esta propiedad indica a las tecnologías de apoyo si al actualizarse parte de una región activa se debe presentar todo el contenido en su interior o sólo aquel que se haya actualizado. Si el valor de la propiedad es *true*, se presentará toda el área por completo. Si, por el contrario, es *false*, se presentarán sólo los elementos que se hayan actualizado.

El siguiente código dispone de dos divisiones anidadas que establecen el atributo `aria-atomic`.

```
<div id="ListadoNoticias" aria-atomic="false">
<p>Listado de noticias</p>
   <div id="destacadas" aria-atomic="true">
<p>Noticia 1</p>
<p>Noticia 2</p>
   </div>
</div>
```

La división `"destacadas"` define el atributo `aria-atomic` como `true`. De esta forma, si se modificara alguno de los dos párrafos que contiene, el dispositivo mostraría todo su contenido leyendo, por ejemplo, todos los títulos. En cambio, no se mostraría el párrafo que se encuentra fuera, puesto que está contenido por otra división que declara este atributo como `true`.

2. Intercambio y gestión de documentos XML

El formato predilecto para el intercambio de datos, y para el que los navegadores ofrecen un amplio soporte, es XML. En este punto estudiaremos cómo se pueden obtener, manipular y mostrar datos en formato XML. Inicialmente, veremos la estructura lógica de un documento XML mediante el DOM. Más tarde, mostraremos cómo obtener documentos XML de forma asíncrona en diferentes navegadores, así como mediante el objeto XMLHttpRequest.

Una vez obtenido un documento, aprenderemos a recorrer y a seleccionar partes de su estructura mediante el DOM y XPath. También podremos mostrar documentos o fragmentos XML por medio de transformaciones al formato HTML con plantillas.

2.1. Estructura lógica de objetos DOM, XML y XHTML

El DOM proporciona una estructura lógica de los documentos como un conjunto de nodos relacionados jerárquicamente entre sí en forma de árbol. Cada nodo es representado por un objeto con métodos y propiedades que permiten la manipulación y el recorrido del documento.

La tabla siguiente enumera los tipos de nodos más relevantes:

Tipo de nodo	Descripción
Element	Representa un elemento (por ejemplo, `<html>`, `<p>`, etc.).
Attribute	Representa un atributo de un elemento.
Text	Es un contenido textual dentro de un elemento o atributo.
CDATASection	Representa una sección CDATA: delimita secuencias de escape en bloques de texto con el fin de que no se interprete como lenguaje de marcado.
Comment	Representa un comentario.
Document	Es un nodo contenedor de todo el documento. Es el elemento raíz del árbol de nodos.
DocumentType	Representa un nodo `<!DOCTYPE...>`
DocumentFragment	Representa un objeto de documento ligero que puede contener fragmentos XML.

La constante y el valor numérico de cada uno de los tipos de nodos expuestos en la tabla anterior son los siguientes:

Valor numérico	Nombre de la constante
1	NODE_ELEMENT
2	NODE_ATTRIBUTE
3	NODE_TEXT
4	NODE_CDATA_SECTION
8	NODE_COMMENT
9	NODE_DOCUMENT
10	NODE_DOCUMENT_TYPE
11	NODE_DOCUMENT_FRAGMENT

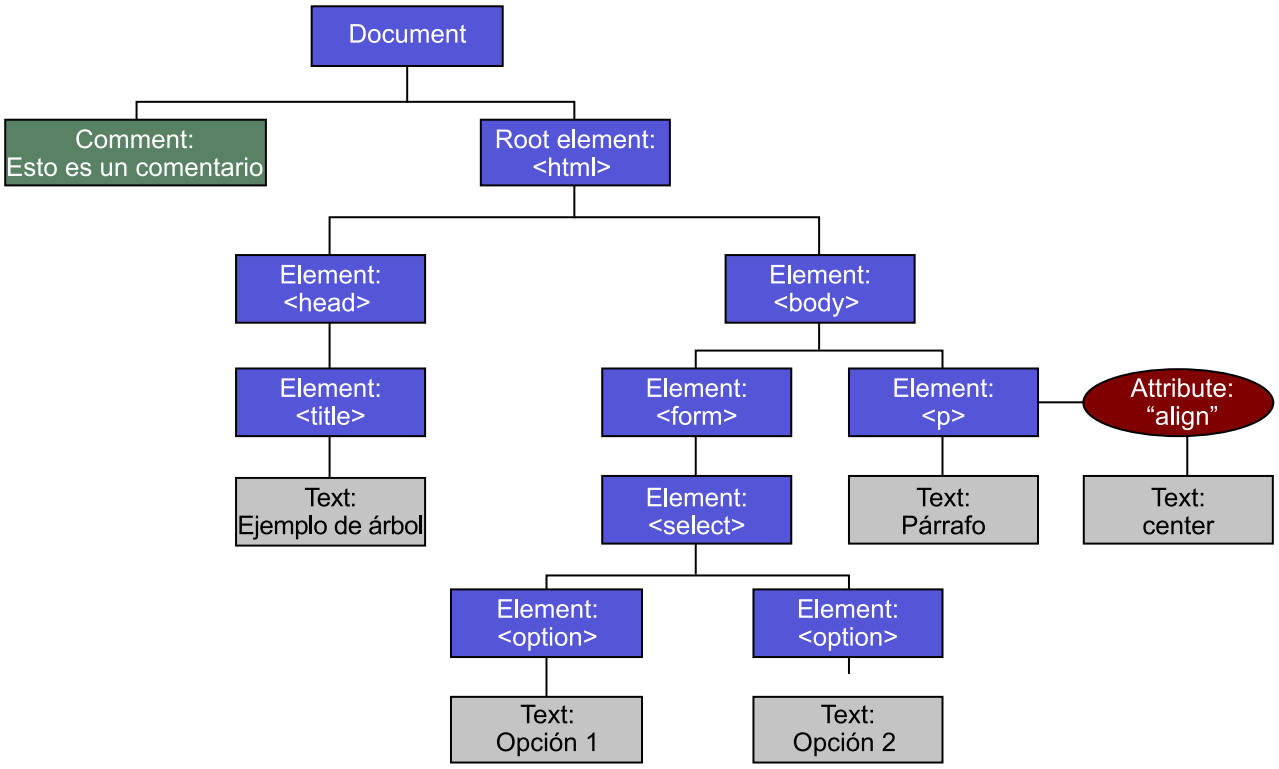
2.1.1. Ejemplo de estructura lógica de un documento

A partir del siguiente documento HTML, mostraremos su árbol lógico.

```
<!-Esto es un comentario -->
<html>
<head>
   <title>Ejemplo de árbol</title>
</head>
<body>
   <form name="userForm">
      <select>
         <option>Option1</option>
         <option>Option2</option>
      </select>
   </form>
   <p align="center">Párrafo</p>
</body>
</html>
```

El árbol resultante de un objeto DOM del documento HTML anterior es el que se muestra en la figura siguiente.

Figura 8. Ejemplo de árbol HTML



El primer nodo de la jerarquía del documento es de tipo `Document`. Todo objeto de documento contiene este tipo de nodo, y es el que se proporciona al crear un DOM. A partir de este nodo, se tiene acceso a todo el documento.

Objeto de documento

Un objeto de documento es una instancia del DOM del documento.

Los diferentes tipos de nodos que aparecen son `Document`, `Element`, `Attribute`, `Text` y `Comment`. Los nodos de tipo `Node_Element` corresponen a todas las etiquetas HTML, `<html>`, `<head>`, `<title>`, `<body>`, `<form>`, `<p>`, `<select>` y `<option>`. La elipse que depende del nodo `<p>` representa un nodo de tipo `Attribute`. Los textos del título del documento, los del atributo `"Align"` y los de las opciones `<option>` son nodos de tipo `Text`.

2.2. Carga de documentos XML

El primer paso para tratar un documento es cargarlo con el fin de disponer de una instancia del DOM. Existen diferentes maneras de obtener un documento XML en función del navegador. El objeto DOM obtenido al cargar un documento es de tipo `Document`.

En este subapartado, estudiaremos cómo se puede cargar un documento XML con los navegadores Internet Explorer y Mozilla Firefox. Más adelante, explicaremos cómo obtener un objeto de documento haciendo uso de XMLHttpRequest para múltiples navegadores.

2.2.1. Carga de un documento XML en Internet Explorer

MSXML[10] es un conjunto de bibliotecas COM de Microsoft para el procesamiento de XML. MSXML ofrece una implementación del DOM a partir de la que se pueden crear y manipular documentos.

(10)MSXML es la sigla de Microsoft XML Core Services.

Para crear un objeto DOM y cargar un documento XML desde Internet Explorer, se debe instanciar MSXML como un objeto ActiveX.

```
var oXmlDom = new ActiveXObject("Microsoft.XmlDom");
```

Puesto que existen diferentes versiones de MSXML, siempre es preferible crear una instancia de la versión más reciente posible.

```
function createDOMXML(){
   var aVersions = ["MSXML2.DOMDocument.6.0",
                    "MSXML2.DOMDocument.5.0",
                    "MSXML2.DOMDocument.4.0",
                    "MSXML2.DOMDocument.3.0",
                    "MSXML2.DOMDocument",
                    "Microsoft.XmlDom"];
   for(var i = 0; i < aVersions.length; i++){
      try{
         var oXmlDom = new ActiveXObject(aVersions[i]);
         return oXmlDom;
      } catch(oError){

      }
   }
   throw new Error("Couldn't create a MSXML instance.");
   }
oXML = createDOMXML();
```

El vector `aVersions` contiene un conjunto de cadenas de versiones diferentes de DOM MSXML. El bucle recorre desde la versión más actual hasta la primera intentando instanciar el objeto. Si se produce una excepción, en la siguiente iteración se intentará hacer lo mismo con una versión anterior, hasta que se obtenga una instancia.

Una vez obtenido el objeto, se puede emplear para conseguir objetos de documentos XML a partir de dos métodos, `load(sURL)` y `loadXML(sXML)`, que se explican a continuación:

- *`load(sURL)`*:

Este método inicializa el objeto a partir de un documento situado en el URL pasado por parámetro.

```
oXML.load("/XML/xmlFile.xml");
```

Al igual que en el objeto XMLHttpRequest, aquí es posible indicar si la comunicación debe realizarse en modo asíncrono o síncrono con la propiedad `async`, estableciéndola como `true` o `false` respectivamente. Por defecto, la comunicación se realiza en modo asíncrono.

Para controlar el estado de la comunicación, MSXML también dispone de las propiedades `readyState` y `onreadystatechange`.

```
oXML = createDOMXML();
oXML.onreadystatechange = function(){
   if(oXML.readyState == 4)
      alert("Document loaded.");
   };
oXML.load("/XML/xmlFile.xml");
```

En el código anterior, se establece en la propiedad `onreadystatechange` una función Java-Script que, en el caso de que el estado del documento sea 4 (`loaded`), hará que se muestre un mensaje indicando que los datos se han cargado correctamente.

- *`loadXML(sXMLString)`*:

Un objeto DOM de MSXML también proporciona el método `loadXML(XMLString)`, que inicializa el objeto a partir de una cadena que contiene el documento XML. La llamada es síncrona, puesto que la lectura es inmediata.

```
var sXML = " <message>
   <from>abc@uoc.edu</from>
   <to>jsirvent123@uoc.edu</to>
</message>
";
oXML.loadXML(sXML);
```

La variable `oXML` contiene el objeto DOM construido a partir de la cadena de caracteres.

2.2.2. Carga de un documento XML en Mozilla Firefox

Firefox ofrece una interfaz para obtener documentos DOM vacíos, es decir, que no representan ningún documento existente. Se trata del método `createDocument()` de la interfaz `DOMImplementation`.

```
function getNewDocument(){
   if(document.implementation &&
   document.implementation.createDocument){
         var domXML = document.implementation.createDocument("", "", null);
      return domXML;
   }
}
```

La función anterior comprueba inicialmente que el navegador disponga del objeto `document.implementation`, que devuelve una instancia de la interfaz `DOMImplementation`. Si es así, llama al método `createDocument()` para obtener un documento vacío pasando tres argumentos: el primero especifica un espacio de nombres para el documento; el segundo acepta una cadena que especificará el nombre del nodo raíz del documento, y el tercer parámetro especifica el tipo de documento, que para mayor compatibilidad es recomendable pasar `null`.

Una vez creado el objeto, se puede solicitar un documento XML a partir de su URL con el método `load`.

```
var oDOM = getNewDocument ();
oDOM.onload = function(){
   alert("XML Document loaded");
}
oDOM.load("lista.xml");
```

En el ejemplo anterior, se asigna una función al evento `onload` que será invocada cuando el documento esté cargado.

Carga del documento

Por defecto, la carga se realiza de forma asíncrona. Esto se puede modificar mediante la propiedad `async` del documento, estableciendo `true` o `false`.

A diferencia de MSXML, el objeto obtenido mediante `createDocument()` no dispone de las propiedades `readyState` y del manejador `onreadystatechange`. Para obtener un documento a partir de una cadena, se puede hacer uso del método `parseFromString` de la clase `DOMParser`. Éste devuelve un objeto DOM de XML a partir de una cadena de caracteres pasada por parámetro.

```
var sXML =
   "<message>
   <from>abc@uoc.edu</from>
   <to>asanchez123@uoc.edu</to>
</message>";
```

```
var oParser =  new DOMParser();
var oDOM = oParser.parseFromString(sXML,"text/xml");
```

Como primer parámetro, se pasa la cadena que contiene el documento. El segundo parámetro permite indicar el tipo de contenido (en este caso XML).

2.2.3. Cargar un documento XML desde un URL para múltiples navegadores

Tanto Internet Explorer como Firefox ofrecen soporte para XML. Ambas implementaciones son diferentes e incompatibles, por lo que hacer uso de una de ellas descartaría a los clientes del resto de navegadores.

La interfaz XMLHttpRequest permite obtener documentos XML y proporcionar objetos de documentos XML mediante la propiedad `responseXML`.

El ejemplo siguiente recupera un documento XML desde el servidor y devuelve un objeto DOM. Se tendrá en cuenta que el objeto XMLHttpRequest está creado y representado mediante la variable `oXMLHttpObject`.

```
function getXMLDOMObject(xmlFileName){
   oXMLHttpObject.open('GET', xmlFileName);
   oXMLHttpObject.onreadystatechange = function(){
      if(oXMLHttpObject.readyState == 4){
         if(XMLHttpObject.status == 200){
            oXMLDOMObject = oXMLHttpObject.responseXML;
         }
         else{
            alert("There was a problem with the request.");
         }
      }
   XMLHttpObject.send(null);
   }
}
```

La función `getXMLDOMObject(xmlFileName)` utiliza XMLHttpRequest para solicitar al servidor el fichero pasado por parámetro. Una vez recibido correctamente el documento, se obtiene el objeto DOM con la propiedad `responseXML`. Este objeto creado dependerá de la implementación del navegador, y esto se debe tener en cuenta a la hora de usar sus propiedades y sus métodos.

2.3. Recorrido de un documento mediante el DOM

Interfaz Node

La interfaz Node está definida en la especificación Document Object Model Level 1 por el W3C.

La interfaz Node, implementada por los diferentes tipos de nodos, ofrece un conjunto de propiedades y métodos para el acceso al documento y su recorrido por él. Las propiedades más importantes se enumeran en la tabla siguiente:

Propiedad	Descripción
attributes	Proporciona un vector con los atributos del nodo actual. Es sólo aplicable a nodos de tipo `Element`.
childNodes	Devuelve un vector con los nodos hijos.
documentElement	Devuelve el nodo que representa el hijo directo del nodo raíz `Document`.
firstChild	Obtiene el primer nodo hijo. Si no existe, devolverá `null`.
lastChild	Obtiene el último nodo hijo.
nextSibling	Obtiene el nodo siguiente dentro de la lista de nodos hijos a la que pertenece un nodo.
nodeName	Obtiene el nombre del nodo.
nodeType	Devuelve el tipo de nodo.
nodeValue	Devuelve el texto contenido en la etiqueta.
parentNode	Hace referencia al padre de un nodo.
previousSibling	Obtiene el nodo anterior dentro de la lista de nodos hijos a la que pertenece un nodo.
tagName	Devuelve el nombre de la etiqueta.

Los métodos siguientes pertenecen a los tipos de nodo `Element` y `Document`:

Nombre	Retorno
appendChild(appendedNode)	Añade un nuevo nodo hijo al final de la lista de nodos.
attributes	Propiedad. Proporciona un vector con los atributos del nodo actual. Sólo es aplicable a nodos de tipo `Element`.
cloneNode(deep)	Devuelve una copia del nodo. Si el parámetro booleano `deep` es `true`, también serán clonados los nodos hijos del nodo actual.
getAttribute(sName)	Devuelve el valor del atributo con el nombre pasado por parámetro.
getAttributeNode(sName)	Devuelve el nodo que representa el atributo con el nombre pasado por parámetro.
getElementsByTagName(sName)	Devuelve un conjunto de nodos `Element`, con el nombre pasado por parámetro, que descienden del nodo actual.
hasAttributes()	Devuelve un parámetro booleano indicando si el nodo contiene atributos.
hasChildNodes()	Devuelve un parámetro booleano indicando si el nodo tiene nodos hijos.

Nombre	Retorno
insertBefore(insNode, node)	Inserta un nodo hijo antes que el segundo nodo.
removeChild(node)	Elimina un nodo hijo.
replaceChild(insNode, replNode)	Reemplaza un nodo hijo por otro pasado por parámetro.

2.3.1. Manipular los espacios en blanco en Mozilla

Los navegadores basados en Mozilla tratan los espacios en blanco y los saltos de línea de un documento XML como nodos de tipo texto. Al cargar un DOM XML, se obtiene un documento cuyo árbol contiene más nodos de lo esperado. Esto afecta al recorrido y a la interpretación del árbol, ya que se recorren nodos innecesarios.

La siguiente función Java-Script elimina nodos de texto vacíos de un objeto DOM pasado por parámetro.

```
function removeWhiteSpace(xmlDOM){
   var notWhitespace = /\s/;
   var i;
   for (i = 0; i < xmlDOM.childNodes.length; i++){
      var currentNode = xmlDOM.childNodes[i];
      if(currentNode.nodeType == 1){
         removeWhiteSpace(currentNode);
      }
      if((notWhitespace.test(currentNode.nodeValue))
         &&(currentNode.nodeType == 3)){
         xmlDOM.removeChild(xmlDOM.childNodes[i--]);
      }
   }
   return xmlDOM;
}
```

Esta función recorre todos los nodos del árbol de forma recursiva. Si un nodo es de tipo texto y contiene un espacio en blanco, se elimina del árbol resultante.

Ved también

Los ejemplos numerados que se presentan a lo largo de todo este subapartado están también disponibles en línea.

Ejemplo 10

El ejemplo 10 pone en práctica algunas propiedades para el recorrido de nodos. A partir de un objeto XMLHttpRequest, se obtiene el DOM de un documento XML y se recorre su estructura por los diferentes nodos. Los ficheros son los siguientes:

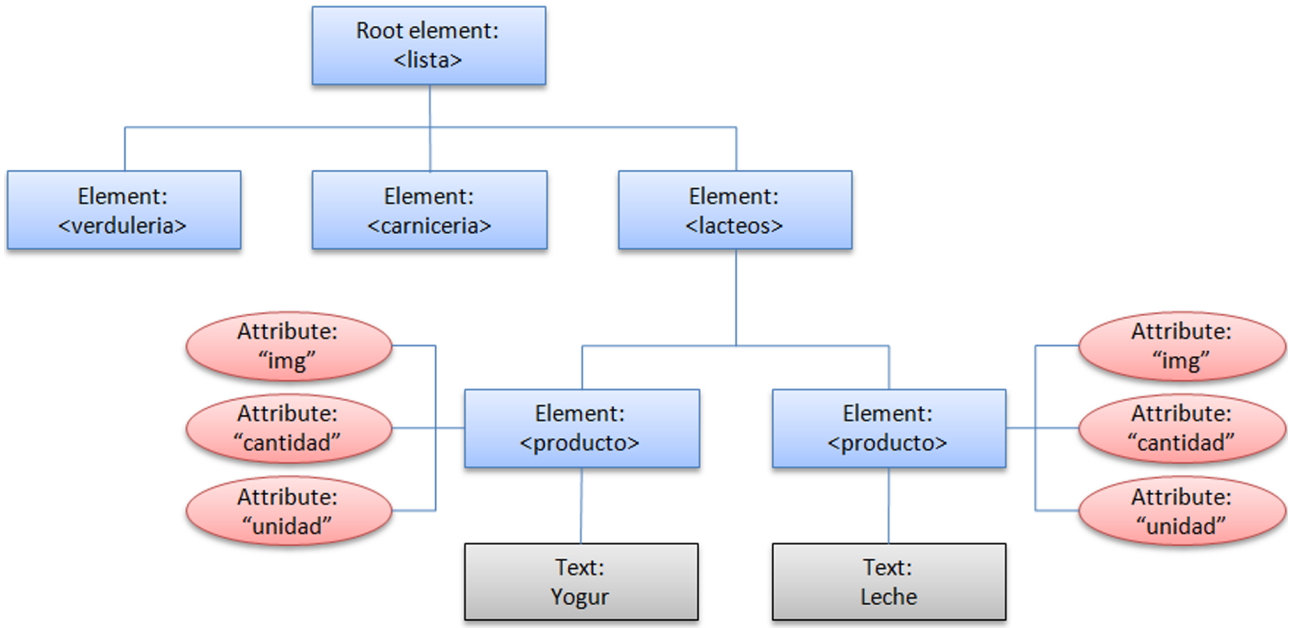
- *lista.xml*:

```
<?xml version="1.0" encoding="UTF-8"?>
<lista>
   <verduleria>
```

```
      <producto imagen="patatas.jpg" cantidad="2" unidad="Kg">
      Patatas</producto>
      <producto imagen="zana.jpg" cantidad="1" unidad="Kg">
      Zanahorias</producto>
      <producto imagen="huevos.jpg" cantidad="1" unidad="Docena">
      Huevos</producto>
   </verduleria>
   <carniceria>
      <producto cantidad="500" unidad="Gramos">
      Jamón</producto>
      <producto cantidad="250" unidad="Gramos">
      Pollo</producto>
   </carniceria>
   <lacteos>
      <producto img="yogur.jpg" cantidad="6">
      Yogur</producto>
      <producto img="leche.jpg" cantidad="2" unidad="litro">
      Leche</producto>
   </lacteos>
</lista>
```

La vista lógica en forma de árbol del documento anterior podría representarse como se muestra en la figura 9.

Figura 9. Árbol de nodos del documento *lista.xml*



Para simplificar la figura, no se muestran los nodos ni los atributos descendientes de los elementos <verduleria> y <carniceria>

- *recorrido.xml*:

```
<html>
<head>
<title>Recorrido de un documento XML</title>
<script type="text/javascript" src="../script/AJAX.js"></script>
<script type="text/javascript">
function mostrarLista(){
   var oXMLHttpRequest = createXMLHttpRequestObject();
   oXMLHttpRequest.open('GET', 'lista.xml',false);
   oXMLHttpRequest.send(null);
   if((oXMLHttpRequest.readyState == 4)&&(oXMLHttpRequest.status == 200)){
      var oDomXML = removeWhiteSpace(oXMLHttpRequest.responseXML);
      var oLista = oDomXML.documentElement;
```

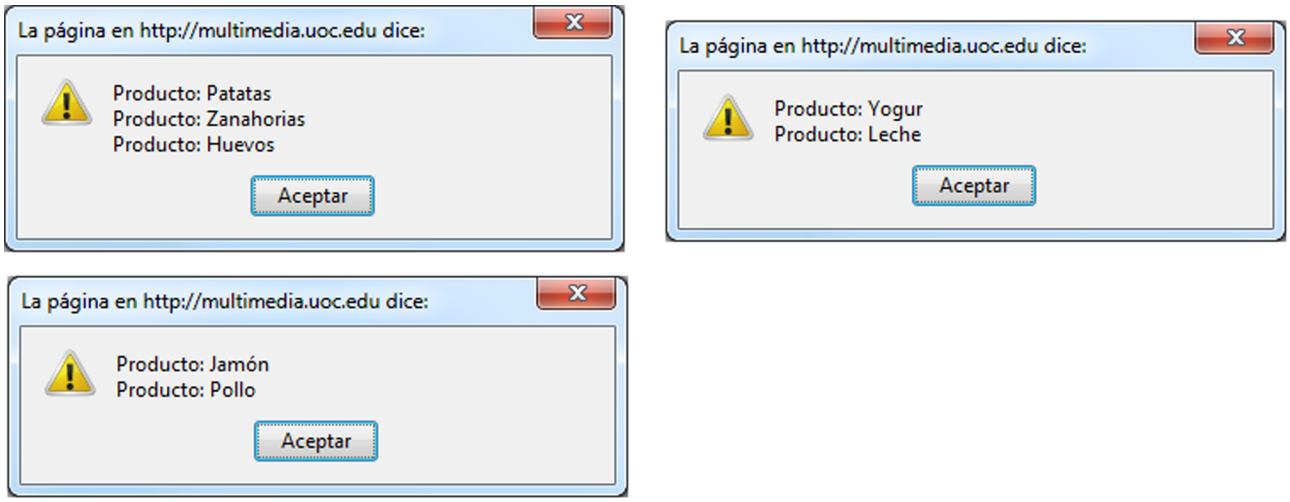
```
        var oVerduleria = oLista.firstChild;
        var oLacteos = oLista.lastChild;
        var oCarniceria = oVerduleria.nextSibling;
        oCarniceria = oLacteos.previousSibling;
        mostrarProductos(oVerduleria);
        mostrarProductos(oLacteos);
        mostrarProductos(oCarniceria);
    }
    else{
        alert('XMLHttpRequest: error en la petición.');
    }
}

function mostrarProductos(oNode){
    var aProducts = oNode.childNodes;
    var sText = '';
    for(var i = 0; i < aProducts.length; i++){
        if(aProducts[i].nodeType == 1){
            if(aProducts[i].firstChild.nodeType == 3){
                sText += 'Producto: ' + aProducts[i].firstChild.nodeValue + '\n';
            }
        }
    }
    alert(sText);
}

</script>
</head>
<body>
<button onclick="mostrarLista()">Mostrar lista</button>
</body>
</html>
```

La página `recorrido.html` muestra todos los productos de la lista. Concretamente, muestra el texto contenido dentro de la etiqueta `<producto>` de cada una de las secciones, `<verduleria>`, `<carniceria>` y `<lacteos>`. El resultado del ejemplo se muestra en la figura 10.

Figura 10. Resultado del ejemplo



La función `mostrarLista()` obtiene el DOM del documento `lista.xml` y elimina los nodos de texto vacíos con la función `removeWhiteSpace()`. A partir del objeto obtenido, se recorre el árbol para obtener los nodos `<verduleria>`, `<carniceria>` y `<lacteos>`.

```
var oDomXML = removeWhiteSpace(oXMLHttpRequest.responseXML);
var oLista = oDomXML.documentElement;
var oVerduleria = oLista.firstChild;
var oLacteos = oLista.lastChild;
var oCarniceria = oVerduleria.nextSibling;
```

La propiedad `documentElement` devuelve el elemento situado en la raíz, que corresponde al nodo `<lista>`. A partir de éste, se pueden obtener el primer y el último nodos hijos con las propiedades `firstChild` y `lastChild`, que en este caso serán `<verduleria>` y `<lacteos>` respectivamente. El nodo `<carniceria>` es hermano de `<verduleria>` y `<lacteos>`. Para obtenerlo desde `oVerduleria`, se puede utilizar la propiedad `nextSibling`, y la propiedad `previousSibling` para obtenerlo desde `oLacteos`. Es decir, las dos instrucciones siguientes devolverían el mismo objeto.

```
var oCarniceria = oVerduleria.nextSibling;
oCarniceria = oLacteos.previousSibling;
```

Luego se llama tres veces a la función `mostrarProductos()` pasando por parámetro cada uno de los nodos obtenidos. La función `mostrarProductos()` recorre los nodos hijos del nodo pasado por parámetro y muestra el texto contenido en la etiqueta.

```
function mostrarProductos(oNode){
   var aProducts = oNode.childNodes;
   var sText = '';
   for(var i = 0; i < aProducts.length; i++){
      if(aProducts[i].nodeType == 1){
```

```
            if(aProducts[i].firstChild.nodeType == 3){
               sText += 'Producto: ' + aProducts[i].firstChild.nodeValue + '\n';
            }
         }
      }
      alert(sText);
   }
```

La primera línea obtiene una matriz con todos los nodos hijos mediante la propiedad `childNodes`. Después recorre todos los nodos de la matriz y, para cada uno, comprueba que el nodo sea de tipo `Element`. Si es así, se accederá al primer nodo hijo, y comprueba que sea de tipo texto, concatenando su contenido con la variable `sText`, que acumula el texto de los productos. Finalmente, se muestra la cadena construida.

2.3.2. Acceso a elementos XML a partir del nombre

En el ejemplo anterior se ha mostrado el texto de todos los nodos de tipo `<producto>`. El código depende del documento, y cualquier cambio que afectara a la estructura del árbol podría hacer que no funcionara de forma correcta.

Los nodos de tipo `Document` ofrecen el método `getElementsByTagName()`, que devuelve un vector de nodos correspondientes a todas las etiquetas con el nombre pasado por parámetro que dependen del nodo que realiza la llamada. Si este método se aplicara sobre el nodo raíz, el resultado serían todos los elementos `<producto>` que se encuentren en el documento.

Nodos de tipo `Document` y `Element`

Los nodos de tipo `Document` y `Element` disponen del método `getElementsByTagName()`.

El ejemplo 11 muestra cómo se pueden obtener todos los productos, representados por la etiqueta `<product>`, sobre el documento XML correspondiente al fichero `lista.xml` del ejemplo anterior.

Ejemplo 11

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
   charset=ISO-8859-1">
<title>Recorrido de un documento XML</title>
<script type="text/javascript" src="../script/Ajax.js"></script>
<script type="text/javascript">
function mostrarLista(){
   var oXMLHttpRequest = createXMLHttpRequestObject();
   oXMLHttpRequest.open('GET', 'lista.xml',false);
   oXMLHttpRequest.send(null);
   //Si se ha obtenido la lista correctamente...
   if((oXMLHttpRequest.readyState == 4)&&
      (oXMLHttpRequest.status == 200)){
      //Se obtiene el objeto de documento
      var oDomXML = oXMLHttpRequest.responseXML;
      //La variable aProducts obtiene un vector
      //con los nodos de tipo producto
      var aProductos = oDomXML.getElementsByTagName('producto');
      var sText = '';
      //Se acumulan los productos en una cadena de caracteres
      for(var i = 0; i < aProductos.length; i++){
         sText += 'Producto: ' + aProductos[i].firstChild.nodeValue
         + '\n';
      }
      //Se muestra la lista de productos
      alert(sText);
   }
   else{
      alert('XMLHttpRequest: error en la petición.');
   }
}
</script>
</head>
<body>
   <button onclick="mostrarLista()">Mostrar lista</button>
</body>
</html>
```

Una vez obtenido el objeto del documento `lista.xml`, se llama al método `getElementsByTagName()` pasando por parámetro 'producto'. El objeto devuelto es una matriz con todos los elementos `<producto>` del documento. La instrucción `for` recorre todos los elementos y muestra el texto que contienen.

2.3.3. Acceso a los atributos de un elemento y a sus valores

Los nodos de tipo `Element` pueden contener atributos. Para acceder a ellos, es posible utilizar la propiedad `attributes`, que devuelve una matriz con todos los nodos de tipo `Attribute`. Si se conoce el nombre de los atributos a los que se quiere acceder, pueden usarse los métodos `getAttribute(attributeName)` y `getAttributeNode(attributeName)`: el primero devuelve el valor del atributo pasado por parámetro, y el segundo devuelve el nodo.

El ejemplo 12 muestra la lista de la compra añadiendo información contenida en los atributos que presenta la etiqueta `<producto>`.

Ejemplo 12

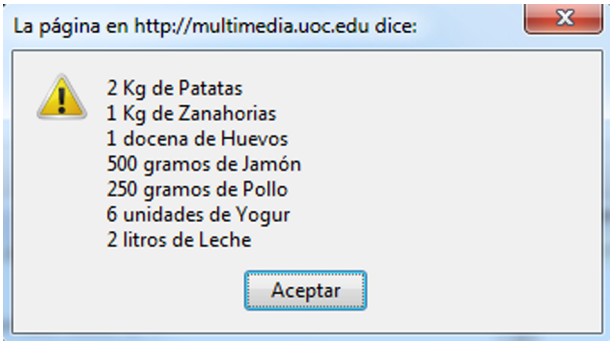
```
<html>
```

```
<head>
<meta http-equiv="Content-Type" content="text/html;
    charset=ISO-8859-1">
<title>Recorrido de un documento XML</title>
<script type="text/javascript" src="../script/Ajax.js"></script>
<script type="text/javascript">
function mostrarLista(){
   var oXMLHttpRequest = createXMLHttpRequestObject();
   oXMLHttpRequest.open('GET', 'lista.xml',false);
   oXMLHttpRequest.send(null);
   //Si se ha recibido la lista correctamente...
   if((oXMLHttpRequest.readyState == 4)&&
          (oXMLHttpRequest.status == 200)){
      //Se obtiene el objeto del documento lista.xml
      var oDomXML = oXMLHttpRequest.responseXML;
      //Se obtiene un vector con todos los nodos <producto>
      var aProductos = oDomXML.getElementsByTagName('producto');
      //Se crea una matriz y en cada posición se añade un producto
      //con su descripción
      var aListaCompra = new Array();
      for(var i = 0; i < aProductos.length; i++){
         var oNodo = aProductos[i];
         var sCantidad = oNodo.getAttribute('cantidad');
         var sUnidad = oNodo.getAttribute('unidad');
         var sProducto = oNodo.firstChild.nodeValue;
         aListaCompra.push(sCantidad + ' ' + sUnidad + ' de ' +
            sProducto);
      }
      //Se concatenan todas las descripciones de productos
      //añadiendo un salto de línea entre ellos
      alert(aListaCompra.join('\n'));
   }
   else{
      alert('XMLHttpRequest: error en la petición.');
   }
}
</script>
</head>
<body>
<button onclick="mostrarLista()">Mostrar lista</button>
</body>
</html>
```

Una vez obtenido el DOM del documento `lista.xml`, se obtienen todos los nodos `<producto>` mediante el método `getElementsByTagName()`. Para cada uno, se crea una cadena que contenga los atributos de cantidad, unidad y texto del elemento, en diferentes variables. La concatenación de las variables se añade en una matriz.

Finalmente, se muestra la matriz llamando a la función `join()` y pasando por parámetro el carácter `\n`, que representa un salto de línea. El resultado es una cadena con la concatenación de todas las cadenas incluidas en la matriz y con el carácter de salto de línea entre cada uno de los elementos.

Figura 11. Listado de productos del fichero *lista.xml*



2.4. Gestión de errores

Al cargar un documento XML se pueden producir errores. Por ejemplo, el documento puede no encontrarse en el URL especificado, el documento puede no estar bien formado, etc. La gestión de errores permite detectar si se ha producido un error al obtener un documento XML y conseguir información acerca de la incidencia.

2.4.1. Gestión de errores en Internet Explorer

Un objeto de documento XML creado con MSXML proporciona el objeto `parseError`. Una vez cargado un documento XML, podemos comprobar el resultado de la operación mediante la propiedad `errorCode` de `parseError`. Siempre que `errorCode` sea diferente de 0, ello indica que se ha producido un error.

A continuación, mostramos las diferentes propiedades del objeto `parseError`:

Propiedad	Descripción
errorCode	Es el código de error.
filePos	Especifica la posición del documento en el que se ha producido el error.
line	Indica el número de línea del documento en la que se ha producido el error.
linePos	Indica la columna, dentro de la línea de la propiedad anterior, en la que se ha producido el error.
reason	Es un texto descriptivo sobre el error.
srcText	Es un texto que contiene la línea que produjo el error.
url	Es el URL del documento.

Ejemplo de error en la carga de un documento

El siguiente ejemplo carga un documento XML a partir de una cadena. La cadena está mal formada, lo que provoca un error:

```
function exception(){
   var oDOMXML = createDOMXML();
   var sXML =
      "<message>abc@uoc.edu</from>
         <to>asanchez123@uoc.edu</to>
      </message>";
   oDOMXML.loadXML(sXML);
   if(oDOMXML.parseError.errorCode != 0){
      alert("Couldn't load XML Document: " +
        oDOMXML.parseError.reason);
   }else{
      alert("XML document loaded successfully");
   }
}
```

Después de la etiqueta `<message>`, faltaría la apertura de `<from>`. Al cargar el documento, se comprueba el estado de la propiedad `errorCode` y, si se ha producido error, se muestra al usuario la descripción de éste.

2.4.2. Gestión de errores en Firefox

Cuando falla el proceso de carga de un documento, en lugar de lanzar una excepción, se devuelve otro documento XML con la descripción del error.

La siguiente función trata de cargar un objeto DOM de XML a partir de una cadena de caracteres. El documento está mal formado, por lo que se generará un documento de error.

```
function exception(){
   var sXML = "<message>abc@uoc.edu</from><to>asanchez123@uoc.edu</to></message>";
   var oParser =  new DOMParser();
   var oDOM = oParser.parseFromString(sXML,"text/xml");
   var oXMLSerializer = new XMLSerializer();
   var sXMLError = oXMLSerializer.serializeToString(oDOM);
   if(oDOM.documentElement.tagName == "parsererror"){
      alert("Couldn't load XML Document: " + sXMLError);
   }else{
      alert("XML document loaded successfully");
   }
}
```

El código anterior comprueba el valor del nodo raíz comparándolo con la cadena `"parsererror"` para verificar si se ha producido algún error al cargar el documento. En este caso, el contenido de la variable `sXMLError` es el siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<parsererror xmlns="http://www.mozilla.org/newlayout/xml/parsererror.xml">
   Error de lectura XML: etiqueta sin pareja. Se esperaba: &lt;/message&gt;.
   Ubicación: file:///C:/xampp/htdocs/DOMXMLIE7/ExceptionFF.html
```

```
   Número de línea 1, columna 23:
   <sourcetext>
&lt;message&gt;abc@uoc.edu&lt;/from&gt;&lt;to&gt;
asanchez123@uoc.edu&lt;/to&gt;&lt;/message&gt;
      ----------------------^
   </sourcetext>
</parsererror>
```

La descripción del error contiene el texto descriptivo del mismo y la fracción del código en la que se originó.

2.5. Selección de nodos con XPath

XPath es un lenguaje que permite direccionar y seleccionar partes de un documento XML a partir de rutas de localización y expresiones. En este punto, presentaremos una breve introducción a este lenguaje.

XPath

XPath es el acrónimo de XML Path Language ('Lenguaje de Rutas XML'). Es un lenguaje utilizado en navegadores y diferentes plataformas de desarrollo. Es una recomendación del W3C y se puede consultar por Internet en:

- Web Accessibility Initiative (WAI)
- XML Path Language (XPath)

Las rutas de localización direccionan un nodo dentro de un documento XML de forma similar a como se realiza la localización de archivos y directorios dentro de un sistema de ficheros. Por ejemplo, si en el sistema UNIX se tuvieran un directorio desde la raíz llamado `DirectorioPadre` y un subdirectorio `DirectorioHijo` que contuviera un fichero llamado `doc.txt`, se accedería al fichero a partir de la ruta siguiente:

```
/DirectorioPadre/DirectorioHijo/doc.txt
```

En un sistema Windows la ruta podría ser la siguiente:

```
C:\DirectorioPadre\DirectorioHijo\doc.txt
```

De forma parecida, la ruta de localización `/Directory/Subdirectory/file` permitiría seleccionar el nodo `<file>` del siguiente documento XML:

```
<Directory>
   <Subdirectory>
      <file name="doc.txt"/>
   </Subdirectory>
</Directory>
```

En XPath las expresiones se evalúan sobre un nodo que se conoce como *nodo de contexto* y, al ser procesadas, se obtiene un objeto. El tipo de objeto devuelto a partir de una expresión puede ser uno de los siguientes:

- *node-set*, una colección desordenada de nodos sin duplicados;
- valor booleano;
- valor numérico, y

- cadena de caracteres.

En este módulo sólo trataremos las expresiones que devuelven una colección de nodos de tipo *node-set*.

2.5.1. Selección de nodos

Los símbolos y los operadores de la tabla siguiente pueden combinarse para construir expresiones que devuelvan un conjunto de nodos en un *node-set*:

Símbolo	Descripción
nodename	Selecciona todos los nodos hijos con el nombre especificado.
/	Identifica el nodo raíz, y también permite especificar rutas separando niveles de nodos.
//	Selecciona los nodos por todo el árbol.
.	Identifica el nodo actual en relación con el nodo de contexto.
..	Identifica el nodo padre del nodo actual.
|	Realiza la unión entre dos expresiones que devuelven un *node-set*.
@	Selecciona atributos.

Ejemplo de selección de nodos

Los ejemplos de expresiones que se exponen a continuación parten del siguiente documento XML:

```
<menu>
   <breakfast time="7:00 am">
      <food price="1">toasts</food>
      <food price="1">eggs</food>
      <food price="3">beans</food>
      <drink price="1">Orange juice</drink>
      <tea price="1">Tea with milk</tea>
   </breakfast>
   <lunch time="12:00 pm">
      <food price="3">beans</food>
      <food price="1">bread</food>
      <drink price="2">coke</drink>
   </lunch>
   <dinner time="8:00 pm">
      <food price="3">salad</food>
      <drink price="1">water</drink>
   </dinner>
</menu>
```

Las expresiones pueden declararse de forma absoluta o relativa al nodo de contexto. Si el nodo de contexto es `<menu>` y se desean obtener todos los elementos `<food>` dentro de `<dinner>`, se aplicará la siguiente expresión:

```
dinner/food
```

La expresión anterior es relativa al nodo de contexto. Concretamente, devolvería el nodo o el conjunto de nodos `<food>` que dependen del nodo `<dinner>` a partir del nodo de contexto. Si el nodo de contexto no fuera en este caso `<menu>`, no devolvería ningún resultado. La expresión absoluta equivalente es la siguiente:

```
/menu/dinner/food
```

Las expresiones absolutas devuelven el mismo resultado sea cual sea el nodo de contexto dentro de un mismo árbol.

La tabla siguiente muestra una serie de ejemplos de expresiones XPath:

Expresión	Descripción
/	Devuelve el nodo `Document`.
/menu/breakfast	Devuelve los nodos `<breakfast>`.
//drink	Devuelve todos los nodos de tipo `<drink>` situados en cualquier parte del documento.
drink	Devuelve todos los nodos hijos `<drink>` que sean hijos directos del nodo de contexto.
../dinner/food	Desciende un nivel sobre el nodo de contexto y localiza el nodo `<dinner>`. A partir de ahí devuelve todos los nodos `<food>`.
drink | food	Selecciona todos los elementos `<drink>` y `<food>` hijos directos del nodo de contexto.
/menu/breakfast@time	Devuelve el nodo correspondiente al atributo `time` del elemento `<breakfast>`.

2.5.2. Uso de XPath en Internet Explorer

Un objeto DOM de XML en Internet Explorer dispone de dos métodos para la selección de nodos mediante expresiones XPath:

- `selectSingleNode(sExpression)`. Devuelve el primer nodo encontrado tras la selección. Como parámetro, solicita una cadena con la expresión XPath.
- `selectNodes(sExpression)`. Devuelve una colección de nodos a partir de la expresión pasada por parámetro.

Ejemplo de uso de XPath en Internet Explorer

A partir del documento `Menu.xml`, se obtendrá el primer nodo `<food>` dentro de `<breakfast>`. Se considera que la variable `oDOM` representa el objeto del documento `menu.xml`.

```
function getFirstFoodBreakfast(oDOM){
return oDOM.documentElement.selectSingleNode("/menu/breakfast/food");
}
```

La función anterior ejecuta la expresión `/menu/breakfast/food`, que devuelve todos los nodos `<food>` dentro de la etiqueta `<breakfast>`. El nodo de contexto es el nodo `Element`, y la expresión especifica una ruta absoluta. Aunque dentro del nodo `<breakfast>` hay tres nodos `<food>`, la función `selectSingleNode()` únicamente devolverá el primero.

Es posible realizar la consulta a partir de cualquier nodo. La siguiente función realiza la misma acción que la anterior.

```
function getFirstFoodBreakfast(oDOM){
var oBreakfast = oDOM.documentElement.firstChild;
var oneFood = oBreakFast.selectSingleNode("food");
return oneFood;
}
```

La variable `oBreakfast` representa el nodo `<breakfast>` del documento. La expresión se ejecuta a partir de este nodo, por lo que no es necesario indicar la ruta absoluta.

La siguiente función devuelve todos los resultados de la misma expresión anterior en una matriz de nodos.

```
function getAllFoodBreakfast(oDOM){
var allFood = oDOM.documentElement.selectNodes("/menu/breakfast/food");
return allFood;
}
```

2.5.3. Uso de XPath en Firefox

El W3C define un conjunto de interfaces para el uso de XPath en DOM. Entre éstas se encuentran `XPathEvaluator` y `XPathResult`, a las que se accede desde Java-Script en navegadores Mozilla. Una instancia de XPathEvaluator permite realizar expresiones XPath sobre un objeto DOM de XML mediante el método `evaluate()`, indicando los parámetros siguientes:

Parámetro	Descripción
xpathExpression	Es una cadena que contiene la expresión XPath que debe ser evaluada.

Parámetro	Descripción
contextNode	Es el nodo del objeto de documento a partir del que se evaluará la expresión, incluyendo sus nodos hijos.
namespaceResolver	Es una función encargada de gestionar los espacios de nombres.
resultType	Es una constante que especifica el tipo de resultado que debe ser devuelto al evaluar la expresión. Estas constantes se encuentran dentro de la interfaz `XPathResult`.
result	Si se especifica un objeto existente, éste se usará para devolver los resultados. Especificando `null`, se devolverá un nuevo objeto `XPathResult`.

El resultado de `evaluate()` es siempre un objeto del tipo indicado por el parámetro `resultType`. Para especificar este parámetro, se puede utilizar una constante definida dentro de la interfaz `XPathResult`. Por ejemplo, para las expresiones XPath que devuelven un tipo simple, se pueden usar las siguientes constantes:

- `XPathResult.NUMBER_TYPE`. El tipo devuelto por la expresión es numérico.
- `XPathResult.STRING_TYPE`. El tipo devuelto por la expresión es una cadena de caracteres.
- `XPathResult.BOOLEAN_TYPE`. El tipo devuelto por la expresión es booleano.

Si, por el contrario, la expresión utilizada devuelve un conjunto de nodos, se puede utilizar la siguiente constante:

```
XPathResult.UNORDERED_NODE_ITERATOR_TYPE
```

El tipo devuelto es un objeto que permite recorrer los nodos del resultado mediante el método `iterateNext()`. Este método devuelve el siguiente nodo del resultado de la expresión. En el caso de que no hayan más, devolverá `null`.

Enlace de interés

Se pueden consultar todos los tipos de resultados de una expresión XPath mediante XPathEvaluator en este enlace:

• *Document Object Model XPath.*

Excepción

Si mientras se recorre el resultado mediante `iterateNext()` se modifica la estructura del árbol del objeto DOM, este método generará una excepción.

Ejemplo de uso de XPath en Firefox

La función `getFoodArray` devuelve una matriz con todos los nodos de tipo `<food>` dentro del árbol.

```
function getFoodArray(oDOM){
   var oEvaluator = new XPathEvaluator();
   var sXPath = "//food";
   var iterator = oEvaluator.evaluate( sXPath,
      oDOM.documentElement,
      null,
      XPathResult.UNORDERED_NODE_ITERATOR_TYPE,
      null);
   var aFood = new Array;
   try{
      var thisNode;
      while(thisNode = iterator.iterateNext())
      {
         aFood.push(thisNode);
      }
      return aFood;
   }
   catch(e){
      alert('Error: The tree was modified during iteration' + e);
   }
}
```

Para llevar a cabo la selección de nodos, se crea un objeto de la clase `XPathEvaluator`. El método `evaluate()` devuelve un objeto que representa el conjunto de nodos de la expresión al especificar la constante `UNORDERED_NODE_ITERATOR_TYPE`.

Mediante el método `iterateNext`, se recorren cada uno de los nodos obtenidos para añadirlos en la matriz. Todo esto está incluido dentro de un bloque *try* para capturar una excepción en el caso de que el árbol del documento sea modificado en el momento en que se accede a los resultados.

2.6. Ejemplo general

Hasta este punto hemos estudiado cómo obtener un documento XML, cómo recorrer su estructura mediante el DOM y cómo seleccionar diferentes partes con XPath. El ejemplo 13 pone en práctica cada uno de estos aspectos.

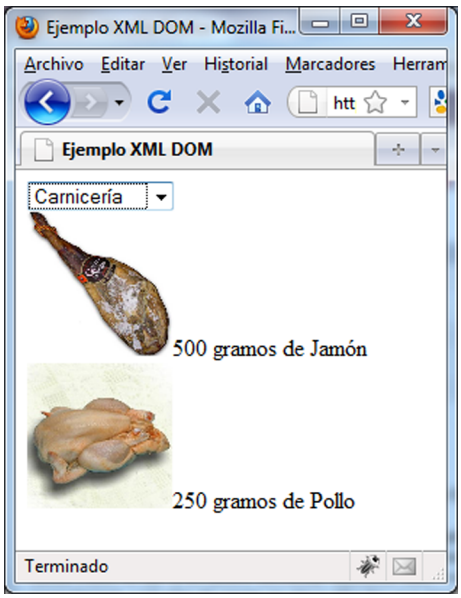
Ejemplo 13

Consta de una página en la que se muestran los diferentes productos del documento `lista.xml`. Un *combo* permite seleccionar una o todas las categorías de productos que se deben visualizar. Para cada producto se muestra su imagen y un texto a partir de sus atributos mediante HTML dinámico.

Ved también

Ved el documento `lista.xml` en el subapartado 2.3 de este módulo didáctico.

Figura 12. Captura del ejemplo



El ejemplo consta de la página siguiente:

- *listaCompra.html*:

```
<html>
<head>
   <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
   <title>Ejemplo XML DOM</title>
   <script type="text/javascript" src="../script/Ajax.js"></script>
   <script type="text/javascript">
   //Se obtiene una instancia de XMLHttpRequest
   oXMLHttpRequest = createXMLHttpRequestObject();
   //variable que representará el objeto del documento lista.xml
   oListaCompra = null;

   //Obtiene el fichero lista.xml de forma asíncrona
   function getListaCompra(){
   ...
   }

   //Añade en el documento una división con la lista
   //de productos a partir de un vector
   function mostrarLista(aProductos){
   ...
   }

   //Crea una matriz de nodos con el producto seleccionado del combo
   //y llama a la función MostrarLista() pasándolos por parámetro
   function cambiarCategoria(){
   ...
```

```
    }
    </script>
</head>

<body onload="getListaCompra()">
    <select id="seccionesCompra" disabled="disabled"
    onclick="cambiarCategoria()">
        <option>Toda la lista</option>
        <option>Verdulería</option>
        <option>Carnicería</option>
        <option>Lácteos</option>
    </select>
    <div id="divListaCompra"></div>
</body>
</html>
```

La página consta de un *combo*, que está inicialmente deshabilitado, y contiene cuatro opciones en la lista, tres para visualizar los productos de cada una de las categorías de alimentos, y la cuarta para mostrar toda la lista. Al seleccionar un elemento del *combo*, se llama a la función `cambiarCategoria()`, que se encargará de visualizar los productos correspondientes dentro de la división `divListaCompra`.

El motor AJAX dispone de las siguientes funciones Java-Script:

a) `createXMLHttpRequestObject()`

Crea un objeto XMLHttpRequest teniendo en cuenta el navegador. La función está dentro del fichero `AJAX.js` incluido en la página.

b) `getListaCompra()`

```
//Obtiene el fichero lista.xml de forma asíncrona
    function getListaCompra(){
        oXMLHttpRequest.open('GET', 'lista.xml');
        oXMLHttpRequest.onreadystatechange = function(){
            if(oXMLHttpRequest.readyState == 4){
                if(oXMLHttpRequest.status == 200){
                    oListaCompra = oXMLHttpRequest.responseXML;
                    if(oListaCompra != null){
                        var oSelect = document.getElementById('seccionesCompra');
                        oSelect.disabled = false;
                        cambiarCategoria();
                    }
                }
                else{
                    alert('Error de petición: ' + oXMLHttpRequest.statusText);
```

```
            }
         }
      }
      oXMLHttpRequest.send(null);
   }
```

Recupera el fichero `lista.xml` de forma asíncrona. Una vez obtenido, habilita el *combobox* de la interfaz para que el usuario pueda seleccionar las diferentes categorías. Posteriormente, llama al método `cambiarCategoria()` para visualizar la lista completa de productos. Esta función se describe a continuación.

c) `mostrarLista(aProductos)`

```
//Añade en el documento una división con la lista de productos a partir de un vector
function mostrarLista(aProductos){
   //En divLista se añadirá el detalle de cada producto
   var divLista = document.createElement('div');
   /*Se recorren todos los productos pasados por parámetro.
   Para cada uno crea una división y añade en ella un conjunto
   de elementos: texto, imagen del producto, etc. Finalmente
   lo añade a divLista*/
   for (var i = 0; i < aProductos.length; i++){
      var sImagen = aProductos[i].getAttribute('imagen');
      var iCantidad = aProductos[i].getAttribute('cantidad');
      var sUnidad = aProductos[i].getAttribute('unidad');
      var sNombre = aProductos[i].firstChild.nodeValue;
      var img = document.createElement('img');
      var div = document.createElement('div');
      img.src = 'images/' + sImagen;
      img.width = 100; img.height = 100;
      div.appendChild(img);
      div.appendChild(document.createTextNode(iCantidad + ' ' + sUnidad +
      ' de ' + sNombre));
      divLista.appendChild(div);
   }
   //Añade la división con los productos a la división divListaCompra
   var docDiv = document.getElementById('divListaCompra');
   if(docDiv.firstChild != null)
      docDiv.removeChild(docDiv.firstChild);
   docDiv.appendChild(divLista);
}
```

Esta función crea la división representada por la variable `divLista` y añade texto e imágenes HTML a partir de la matriz de nodos `<producto>` pasada por parámetro.

El bucle `for` recorre todos los nodos `<producto>` y, para cada uno, realiza lo siguiente:

1) Obtiene los valores de los atributos `imagen`, `cantidad` y `unidad` del producto en las variables correspondientes. También recupera el texto de la etiqueta (*tag*), que describe el nombre del producto mediante la propiedad `nodeValue`.

2) Crea una división en la que se insertarán la imagen del producto y el texto.

3) Crea una imagen dinámicamente y le asigna los atributos `img` con el URL y `width` y `height` para establecer el ancho y el alto de la misma. Posteriormente, añade la imagen dentro de la división.

4) Añade en la división un nodo de texto con la cantidad, el tipo de unidad y el nombre del producto.

5) Añade la división del producto dentro de la división representada por la variable `divLista`.

Después del bucle, la división `divLista` contendrá aún una división con una imagen y un texto para cada uno de los productos de la matriz. Las siguientes líneas después del bucle establecen la división `divLista` en la división declarada en el documento para su visualización; si ya se había establecido una división anteriormente, se elimina.

d) `cambiarCategoria()`

```
function cambiarCategoria(){
   if(oListaCompra == null) getListaCompra();
   var index = document.getElementById('seccionesCompra').selectedIndex;
   var sxPathString;
   switch(index){
      case 0:
         sxPathString = '//producto';
         break;
      case 1:
         sxPathString = '/lista/verduleria//producto';
         break;
      case 2:
         sxPathString = '/lista/carniceria//producto';
         break;
      case 3:
         sxPathString = '/lista/lacteos//producto';
         break;
   }
   var aProducts = null;
```

```
    try{
       var iterator = oListaCompra.evaluate(sxPathString, oListaCompra, null,
       XPathResult.ANY_TYPE, null);
       aProducts = new Array();
       while(thisNode = iterator.iterateNext()){
          aProducts.push(thisNode);
       }
    }catch(error){
       try{
          aProducts = oListaCompra.selectNodes(sxPathString);
       }catch(error){
          alert("Navegador no compatible con XPath");
       }
    }
    mostrarLista(aProducts);
 }
```

Esta función crea la matriz de nodos `<producto>`, que se pasará a la función `mostrarLista()`. Los nodos obtenidos corresponden a la categoría seleccionada en el *combo* y se seleccionan a partir de una expresión XPath que simplifica la tarea de obtener los nodos correspondientes a una categoría de producto.

La primera condición comprueba que se haya obtenido el DOM del documento `lista.xml` y, si no es así, lo obtiene llamando a la función `getListaCompra()`. La tarea siguiente es obtener el índice del *combo*. En un conmutador[11] se evalúa este índice y se asigna a una variable la expresión XPath que permitirá obtener los nodos `<producto>` correspondientes a la categoría seleccionada.

(11) En inglés, *switch*.

A continuación, hay dos `try... catch` anidados. En su interior se procesa la expresión XPath sobre el documento `lista.xml`. El primer `try... catch` evalúa la expresión para navegadores Firefox y otros que siguen el estándar definido por el W3C. Después de ejecutar el método `evaluate` sobre el DOM del documento XML, se añaden todos los nodos obtenidos sobre la matriz `aProducts`. Si el navegador utilizado fuera Internet Explorer, se generaría una excepción y se ejecutaría la instrucción del segundo `try... catch`. El objeto retornado por la sentencia `selectNodes` para Internet Explorer ya devuelve una matriz al que se establece a `aProducts`.

Finalmente, se llama a la función `mostrarLista(aProducts)` pasando por parámetro la matriz `aProducts`, que será representada en la página.

Figura 13. Captura del ejemplo

2.7. Transformaciones XSL

XSL es una familia de recomendaciones de W3C que definen transformaciones y presentaciones de documentos XML en diferentes formatos, como el XHTML. A partir de un documento XML, los diseñadores utilizan XSL para expresar los datos del documento en un medio de presentación mediante un proceso de transformación.

Dentro de esta familia, cabe destacar los lenguajes siguientes:

- XSLT[12]. Este lenguaje define cómo deben formatearse los documentos XML en otros también basados en XML. Puede consultarse la recomendación por Internet en "XLS transformations (XSLT)".

- XSL-FO[13]. Está orientado a la creación de documentos de diferentes formatos no basados en XML. Permite definir otros aspectos de presentación, como las características de una página, párrafos, tablas, vínculos, etc.

- XPath. Es el acrónimo de XML Path Language. Es un lenguaje que permite el direccionamiento y la selección de diferentes partes de un documento.

En esta familia hay que destacar XSLT y XPath, lenguajes que pueden usarse conjuntamente para transformar documentos XML en documentos XHTML a partir de una plantilla. El objetivo de este punto es estudiar cómo se puede obtener un documento XML, transformarlo en un documento en formato XHTML y presentarlo en un área de la interfaz del cliente.

(12)XSLT es la sigla de *extensible stylesheet language transformations* ('lenguaje de hojas de estilo de transformaciones').

(13)XSL-FO es la sigla de *extensible stylesheet language formatting objects* ('lenguaje de hojas de estilo para formatear objetos').

Formatos no basados en XML

Algunos de los formatos de salida son documentos PDF, SVG, RTF, etc.

Para transformar un documento XML, será necesario tener en cuenta el navegador del cliente. A continuación, explicamos cómo se puede realizar esta transformación en navegadores Internet Explorer y Firefox. La plantilla XSL aplicada para los dos casos es la siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:output method="html" omit-xml-declaration="yes" indent="yes" />
   <xsl:template match="/">
      <div>
      <xsl:for-each select="//producto">
         <div>
            <xsl:variable name="varImagen" select="@imagen" />
            <xsl:variable name="varCantidad" select="@cantidad" />
            <xsl:variable name="varUnidad" select="@unidad"/>
            <xsl:variable name="varProducto" select="."/>
            <img  src="./images/{$varImagen}" width="100" height="100" />
            <xsl:value-of select="$varCantidad"/> 
            <xsl:value-of select="$varUnidad"/> de
            <xsl:value-of select="$varProducto"/>
         </div>
      </xsl:for-each>
      </div>
   </xsl:template>
</xsl:stylesheet>
```

Esta plantilla puede aplicarse sobre el documento *lista.xml*. La transformación que lleva a cabo consta de un fragmento HTML con la lista de productos. Para cada uno de los productos, muestra su imagen y un texto explicativo. El código HTML obtenido a partir de la plantilla anterior es el siguiente:

```
<div>
   <div>
      <img src="./images/patatas.jpg" width="100" height="100">
      2 Kg de patatas
   </div>
   <div>
      <img src="./images/zana.jpg" width="100" height="100">
      1 Kg de zanahorias
   </div>
   <div>
      <img src="./images/huevos.jpg" width="100" height="100">
      1 docena de huevos
   </div>
   <div>
      <img src="./images/jamon.jpg" width="100" height="100">
```

```
      500 gramos de jamón
   </div>
   <div>
      <img src="./images/pollo.jpg" width="100" height="100">
      250 gramos de pollo
   </div>
   <div>
      <img src="./images/yogur.jpg" width="100" height="100">
      6 unidades de yogur
   </div>
   <div>
      <img src="./images/leche.jpg" width="100" height="100">
      2 litros de leche
   </div>
</div>
```

2.7.1. Transformaciones XSL en Internet Explorer

El DOM de Internet Explorer dispone del método `transformNode(oXSL)`. Este método devuelve el documento en formato de texto resultado de transformar el DOM que realiza la llamada con la plantilla XSL pasada por parámetro. Puede llamarse al método desde cualquier nodo, y el resultado será la transformación del árbol que desciende a partir de éste.

Inicialmente, será necesario obtener los DOM de los documentos XML y XSL. Para ello se puede utilizar la propiedad `responseXML` de XMLHttpRequest.

Plantilla de transformacion XSL

Esta plantilla *lista.xsl* también se puede obtener con la propiedad `responseXML` de XMLHttpRequest, ya que los documentos XSL son también XML.

El código siguiente obtiene los objetos de documento de *lista.xml* y *lista.xsl* mediante la función `getDocument(sDocURL)`.

```
function getDocument(sDocURL){
   var oXMLHttpRequest = createXMLHttpRequestObject();
   oXMLHttpRequest.open('GET', sDocURL,false);
   oXMLHttpRequest.send(null);
   if(oXMLHttpRequest.readyState == 4){
      if(oXMLHttpRequest.status == 200){
         return oXMLHttpRequest.responseXML;
      }
      else{
         alert('Error de petición: ' + oXMLHttpRequest.statusText);
      }
   }
}

var oDOMXML = getDocument('lista.xml');
var oDOMXSL = getDocument('lista.xsl');
```

Una vez obtenidos los DOM, se puede llevar a cabo la transformación a partir del objeto de documento XML con el método `transformNode(oXSL)`, pasando por parámetro el DOM del documento XSL.

```
oDOMXML.async = false;
oDOMXSL.async = false;
var sResult = oDOMXML.transformNode(oDOMXSL);
```

La variable `sResult` contendrá el texto de los datos transformados. Para mostrarlos, se puede establecer, en la propiedad `innerHTML`, una etiqueta `<div>`.

```
var div = document.getElementById('div');
div.innerHTML = sResult;
```

2.7.2. Transformaciones XSL en Firefox

Las transformaciones XSL en Firefox se realizan mediante la interfaz provista por la clase `XSLTProcessor`. Se debe crear una instancia e inicializar el objeto con una plantilla.

```
var oXSLTProcessor = new XSLTProcessor();
oXSLTProcessor.importStylesheet(oDOMXSL);
```

La variable oDOMXSL es un DOM de un documento XSL previamente cargado.

Para llevar a cabo la transformación, se puede hacer uso de uno de los dos métodos de `XSLTProcessor` que se enumeran a continuación:

1) `transformToDocument(oXMLDOM)`. Requiere un nodo como argumento. Devuelve un nuevo DOM de tipo *document* con el resultado de la transformación.

```
var oDocument = oXSLTProcessor.transformToDocument(oXMLDOM);
```

2) `transformToFragment(oXMLDOM, ownerDocument)`. Devuelve un fragmento de documento de tipo DocumentFragment y requiere dos parámetros: el primero es el nodo XML a partir del que se aplicará la transformación. El segundo es el objeto de documento que poseerá el fragmento devuelto.

```
var oFrag = oXSLTProcessor.transformToFragment(oListaXML, document);
var div = document.getElementById('division');
div.appendChild(oFrag);
```

Fragmentos de documento

Los fragmentos de documento son porciones ligeras de objetos de documento. Son útiles para manipular nodos sin tener que procesar el árbol completo. Una vez procesados estos nodos, se pueden añadir al documento original.

En el código anterior se añade el fragmento devuelto a una parte del mismo documento que se pasó por parámetro para llevar a cabo la transformación.

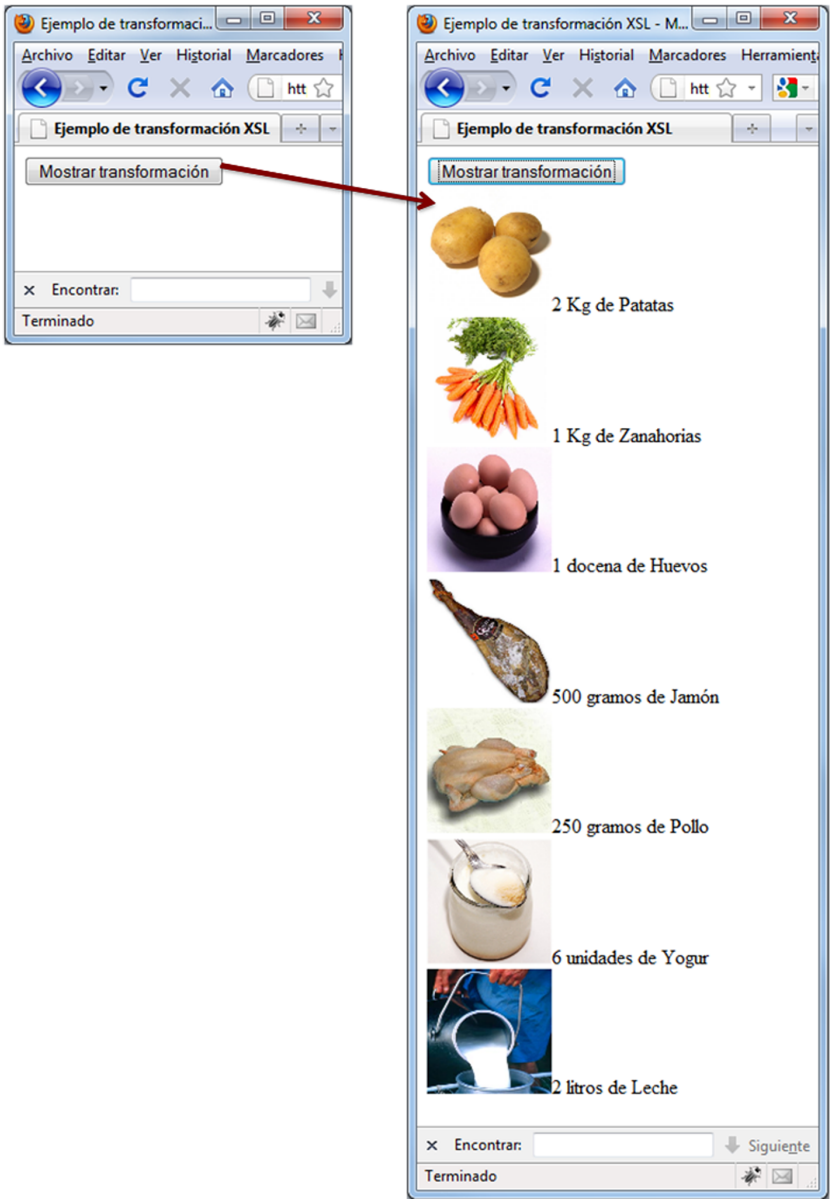
2.7.3. Ejemplo de transformación XSL para plataformas cruzadas

El ejemplo 14 es parecido al del apartado anterior, en el que se mostraban productos de un mismo tipo a partir de un *combo*. En este caso, todos los productos se muestran al hacer clic sobre un botón.

Ejemplo 14

Se utiliza la plantilla *lista.xsl* y se realiza una transformación a partir de la que se obtiene un fragmento HTML listo para ser añadido a cualquier parte del documento sin necesidad de crear HTML dinámico. Se tienen en cuenta las diferentes implementaciones de los navegadores Internet Explorer y Mozilla Firefox.

Figura 14. Resultado del ejemplo



El fichero del ejemplo es el siguiente:

- *listaCompraXSL.html*:

```
<html>
<head>
   <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
   <title>Ejemplo de transformación XSL</title>
   <script type="text/javascript" src="../script/Ajax.js"></script>
   <script type="text/javascript">
   oListaXML = null;
   oListaXSL = null;

   //Devuelve el objeto de documento a partir del URL
   //de un fichero XML
   function getDocument(sDocURL){
```

```
    ...
    }

    //Realiza la transformación XSL y muestra el resultado en divListaCompra
    function mostrarXSLT(){
    ...
    }

    //Obtiene los objetos de documento de lista.xml y lista.xsl
    function cargaDocumentos() {
    ...
    }

    </script>
</head>

<body onload="cargaDocumentos()">
    <button id="button" disabled="disabled" onclick="mostrarXSLT()">
       Mostrar transformación
    </button>
    <div id="divListaCompra"></div>
</body>
</html>
```

La página contiene, dentro de `<body>`, un botón y una división. El botón, deshabilitado inicialmente, llamará a la función encargada de realizar la transformación XSL y de mostrarla en la división. La función establecida en el evento `onload` de la etiqueta `<body>` carga los documentos *lista.xml* y *lista.xsl*.

```
//Obtiene los objetos de documento de lista.xml y lista.xsl
function cargaDocumentos() {
    oListaXML = getDocument('lista.xml');
    oListaXSL = getDocument('lista.xsl');
    //Si se han obtenido correctamente, se habilita el botón
    if((oListaXML != null) && (oListaXSL != null)){
       var oButton = document.getElementById('button');
       oButton.disabled = false;
    }
}
```

Las variables globales `oListaXML` y `oListaXSL` contendrán un DOM XML a partir de los documentos *lista.xml* y *lista.xsl* respectivamente. Para cargarlos y obtenerlos, se utiliza la función `getDocument()`, pasando como parámetro el URL del documento que se desea obtener. Si se han obtenido correctamente, se habilita el botón que permite al usuario hacer clic sobre él y mostrar la lista de la compra.

La función `getDocument` carga los documentos a partir de un objeto XMLHttpRequest.

```
//Devuelve el objeto de documento a partir del URL
   //de un fichero XML
   function getDocument(sDocURL){
      var oXMLHttpRequest = createXMLHttpRequestObject();
      oXMLHttpRequest.open('GET', sDocURL,false);
      oXMLHttpRequest.send(null);
      if(oXMLHttpRequest.readyState == 4){
         if(oXMLHttpRequest.status == 200){
            return oXMLHttpRequest.responseXML;
         }
         else{
            alert('Error de petición: ' + oXMLHttpRequest.statusText);
         }
      }
}
```

La petición se realiza de manera síncrona, porque se debe esperar a la respuesta para devolver el DOM XML obtenido.

Una vez obtenidos los documentos y habilitado el botón, el usuario puede hacer clic sobre él para lanzar la función `mostrarXSLT()`. Ésta transforma el documento `lista.xml` y lo visualiza en la división.

```
//Realiza la transformación XSL y muestra el resultado en divListaCompra
   function mostrarXSLT(){
      var div = document.getElementById('divListaCompra');
      try{//Firefox
         var oXSLTProcessor = new XSLTProcessor();
         oXSLTProcessor.importStylesheet(oListaXSL);
         var oFragment = oXSLTProcessor.transformToFragment(oListaXML, document);
         if(div.firstChild != null)
            div.removeChild(div.firstChild);
            div.appendChild(oFragment);
      }catch(error){
         try{//Internet Explorer
         oListaXML.async = false;
         oListaXSL.async = false;
         var sResult = oListaXML.transformNode(oListaXSL);
         div.innerHTML = sResult;
      }catch(error){
         alert("Navegador no compatible con transformaciones XSL");
      }
   }
```

```
    }
```

La transformación se realiza en dos bloques `try` anidados. El primer `try... catch` lleva a cabo la transformación para navegadores Firefox y otros. En el caso de que ocurra un error, ésta se intentará en el segundo `try` para navegadores Internet Explorer:

1) En el caso de **Firefox**, se crea una instancia de XSLTProcessor y se le establece la plantilla con el método `importStylesheet()` pasando por parámetro el DOM de `lista.xsl`. La variable `oFragment` contendrá un fragmento de objeto de documento con el resultado de la transformación del documento *lista.xml* después de llamar a `transformToFragment()`. El resultado es un nodo de un fragmento de documento que se debe añadir al nodo `<div>` de la página. Previamente, mediante su visualización, se comprueba que la división no tenga un nodo hijo; en el caso de que así fuera, se eliminaría. Por último, se llama al método `appendChild` de la división pasando por parámetro el nodo del fragmento obtenido para que se visualice.

2) Para **Internet Explorer**, el procedimiento es diferente. Inicialmente, se establece la propiedad `async` de `oListaXML` y `oListaXSL` como `false`. Esto evitará que el método `transformNode` bloquee la ejecución hasta realizar la transformación; en el caso contrario, se intentaría visualizar el resultado sin haber obtenido la transformación.

El método `transformNode` devuelve el texto HTML resultado de la transformación. Una vez obtenido éste, se establece en la división del documento la propiedad `innerHTML`.

3. Intercambio de datos con JSON

JSON[14] es un formato de datos ligero y abierto que permite el intercambio de información estructurada entre diferentes plataformas. Se puede usar como una alternativa al formato XML cuando el tamaño de los datos que se han de transmitir es de vital importancia.

(14)JSON es la sigla de *Java-Script object notation.*

Enlace de interés

Para más información, podéis consultar la página json.org.

El modelo de datos de JSON está basado en la sintaxis de Java-Script para la notación literal de objetos, que permite la definición textual de matrices y objetos. El acceso a estos datos desde Java-Script se realiza sin ningún tipo de manipulación, a diferencia de lo que ocurre en el DOM de XML.

3.1. Notación literal de objetos y matrices en Java-Script

La notación literal de objetos de Java-Script es un subconjunto del lenguaje que provee de mecanismos concisos para definir objetos a partir de valores literales a los que se accede por un nombre. Se construyen mediante llaves, dentro de las cuales se especifica un conjunto de parejas de nombre y valor separados por comas.

```
var oPersona = {
   "nombre"   : "Albert",
   "edad"     : 25,
   "email"    : albert@dominio.net
};
```

El ejemplo anterior crea un objeto en una variable `oPersona`. Se puede acceder a los valores de `oPersona` por las propiedades definidas por el nombre de cada pareja nombre-valor:

```
alert(oPersona.nombre);
```

También se puede definir una matriz con notación literal e instanciando un objeto de la clase *Array*. La notación literal de una matriz se especifica con corchetes, dentro de los cuales aparece un conjunto de valores literales separados por comas. Cada valor puede ser de un tipo diferente: numérico, booleano, cadena de caracteres, un objeto en su notación literal, etc.

```
var aListaLiteral = [5, "cadena de caracteres", false];
```

Se puede acceder a los diferentes valores contenidos en la matriz por un índice numérico.

```
for(var i = 0; i < aListaLiteral.length; i++){
   alert(aListaLiteral[i]);
}
```

Los objetos y las matrices declarados con la notación literal pueden combinarse para crear un objeto con propiedades que sean matrices o una matriz cuyos valores sean objetos.

En el caso de un objeto, cualquier propiedad puede contener un valor con una matriz literal.

```
var oPersona = {
   "nombre"    : "Albert",
   "edad"      : 25,
   "email"     : "albert@dominio.net",
   "carnets"   : ["A","B1"]
};
```

La propiedad `"carnets"` representa una matriz literal con dos valores. El acceso al segundo valor de carnets se lleva a cabo tal como sigue:

```
alert(oPersona.carnets[1]);
```

De la misma forma, se pueden definir matrices con valores que sean objetos.

```
var aPersonas = [
   {
      "nombre"   : "Albert",
      "edad"     : 25,
      "email"    : "albert@dominio.net"
   },
   {
      "nombre"   : "Jordi",
      "edad"     : 30,
      "email"    : "jordi@dominio.net"
   }
];
```

La matriz anterior define dos objetos con sus respectivas propiedades y valores. El acceso a una propiedad de un objeto se realizaría indicando el índice y la propiedad.

```
alert(aPersonas[1].email);
```

3.2. Intercambio de información en formato JSON

Analizador sintáctico

Un analizador sintáctico forma parte de un compilador o intérprete y permite la transformación de un texto de entrada en objetos y datos del propio lenguaje.

Los sistemas que intercambian datos en JSON deben disponer de un analizador sintáctico que permita codificar los objetos y las matrices representados textualmente en objetos y matrices del propio entorno y viceversa.

Java-Script dispone de la función `eval()` para analizar y ejecutar código Java-Script a partir de una cadena de caracteres. Si pasamos una cadena de caracteres que contenga un objeto o una matriz en notación literal, `eval()` devolverá una variable que permitirá el acceso a los datos.

El ejemplo siguiente muestra cómo obtener un objeto JSON del servidor y cómo crear un objeto Java-Script para acceder a sus propiedades.

- *persona.txt*:

```
{
   "nombre"   : "Albert",
   "edad"     : 25,
   "email"    : "albert@dominio.net"
}
```

- *eval.html*:

```
...
var oXMLHttpRequest = createXMLHttpRequestObject();
oXMLHttpRequest.open('GET','persona.txt',false);
oXMLHttpRequest.send(null);
var sCadena = oXMLHttpRequest.responseText;
var oPersona = eval('(' + sCadena + ')');
alert(oPersona.nombre);
...
```

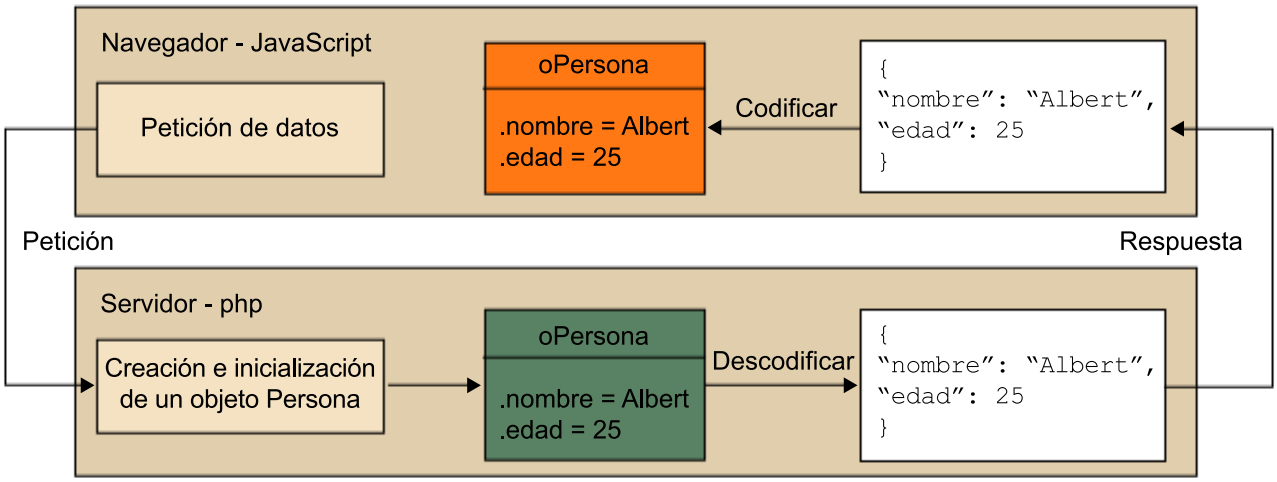
El objeto XMLHttpRequest del ejemplo recupera el texto del fichero *persona.txt* y se asigna en la variable `sCadena`. La función `eval()` transforma el texto pasado por parámetro en un objeto. Java-Script no dispone de ninguna función para transformar objetos a su notación literal, por lo que será necesario el uso de bibliotecas externas.

Por razones de seguridad, no se recomienda el uso de la función `eval()`, sino un analizador distinto que ofrezca mayor garantía. Ello es porque `eval()` puede evaluar cualquier código Java-Script, incluidas funciones. Si en lugar de datos en formato JSON se obtuviera código malicioso, éste se podría ejecutar, lo que pondría en peligro el correcto funcionamiento de la aplicación.

JSON en el lado del servidor

En el ejemplo anterior, se ha obtenido un objeto procedente de un fichero que contenía la información textual. Pero en la mayoría de casos, interesa tratar los datos desde el servidor y convertirlos en objetos JSON de forma dinámica, tal y como muestra la figura siguiente.

Figura 15. Codificación y descodificación de datos JSON



La figura representa una petición desde un navegador a un servidor. El servidor genera los datos en objetos propios PHP que, más adelante, descodifica para transformarlos en formato JSON. El navegador recibe los datos y los codifica para tratarlos posteriormente.

La mayoría de entornos de desarrollo del lado del servidor disponen de herramientas para descodificar objetos a este formato.

3.3. Ventajas e inconvenientes de JSON sobre XML

Los datos representados en formato JSON son más accesibles en Java-Script, puesto que utilizan un subconjunto del lenguaje. La manipulación de un documento XML puede resultar más compleja. Además, los datos en formato JSON son más compactos. Para ver la diferencia entre JSON y XML, estudiaremos un ejemplo con dos ficheros que contienen la misma información, pero representada en ambos formatos.

```
<application>
   <users>
      <user>
         <name>Joan Perelló</name>
         <id>34532</id>
         <email>jperello@domain.net</email>
      </user>
      <user>
         <name>Silvia Pons</name>
         <id>12598</id>
         <email>silpons@domain.net</email>
      </user>
   </users>
   <administrators>
      <admin>
```

```
            <name>Andrea Álvarez</name>
            <id>43229</id>
            <email>aalvarez@domain.net</email>
         </admin>
         <admin>
            <name>Enrique Fernández</name>
            <id>41321</id>
            <email>efernandez@domain.net</email>
         </admin>
      </administrators>
   </application>
```

El documento anterior representa una lista de usuarios y administradores que tienen acceso a una aplicación. Las etiquetas `<application>`, `<users>`, `<administrators>`, `<user>`, `<admin>`, `<name>`, `<id>` y `<email>` se duplican en cada aparición para identificar un solo elemento. El documento tiene un total de 430 caracteres si se eliminan los espacios.

Por otro lado, el fichero que contiene los datos en formato JSON es el siguiente:

```
{
   "application" :
      {
         "users" : [
            {
               "name" : "Joan Perelló",
               "id" : 34532,
               "email" : "jperello@domain.net"
            },
            {
               "name" : "Silvia Pons",
               "id" : 12598,
               "email" : "silpons@domain.net"
            }
         ],
         "administrators" : [
            {
               "name" : "Andrea Álvarez",
               "id" : 43229,
               "email" : "aalvarez@domain.net"
            },
            {
               "name" : "Enrique Fernández",
               "id" : 41321,
               "email" : "efernandez@domain.net"
            }
         ]
```

```
            }
        }
```

Este formato, aunque pueda parecer más extenso por el número de líneas, contiene menos caracteres, 317, y éstos representan la misma información. Esto significa que los mismos datos en formato JSON ocupan 113 caracteres menos.

Pese a su reducido tamaño, JSON puede resultar más inteligible para los humanos que el formato XML. Esto puede no ser un problema si los datos intercambiados no requieren la interpretación humana.

Aunque podamos encontrar herramientas JSON en la mayoría de entornos, XML puede presumir de mayor soporte, tanto en el lado del cliente como en el del servidor.