Interacción con el servidor

1. Procesando respuestas

XML

Además del envío de parámetros en formato XML, el objeto `XMLHttpRequest` también permite la recepción de respuestas de servidor en formato XML. Una vez obtenida la respuesta del servidor mediante la propiedad `petición_http.responseXML`, es posible procesarla empleando los métodos DOM de manejo de documentos XML/HTML.

En este caso, se modifica la respuesta del servidor para que no sea un texto sencillo, sino que la respuesta esté definida mediante un documento XML:

```
<respuesta>
 <mensaje>...</mensaje>
 <parametros>
  <telefono>...</telefono>
  <codigo_postal>...</codigo_postal>
  <fecha_nacimiento>...</fecha_nacimiento>
 </parametros>
</respuesta>
```

La respuesta del servidor incluye un mensaje sobre el éxito o fracaso de la operación de validación de los parámetros y además incluye la lista completa de parámetros enviados al servidor.

La función encargada de procesar la respuesta del servidor se debe modificar por completo para tratar el nuevo tipo de respuesta recibida:

```
function procesaRespuesta() {
 if(peticion_http.readyState == READY_STATE_COMPLETE) {
  if(peticion_http.status == 200) {
   var documento_xml = peticion_http.responseXML;
   var root = documento_xml.getElementsByTagName("respuesta")[0];

   var mensajes = root.getElementsByTagName("mensaje")[0];
   var mensaje = mensajes.firstChild.nodeValue;

   var parametros = root.getElementsByTagName("parametros")[0];

   var telefono = parametros.getElementsByTagName("telefono")[0].firstChild.nodeValue;
   var fecha_nacimiento =
parametros.getElementsByTagName("fecha_nacimiento")[0].firstChild.nodeValue;
   var codigo_postal =
parametros.getElementsByTagName("codigo_postal")[0].firstChild.nodeValue;
```

```
    document.getElementById("respuesta").innerHTML = mensaje + "<br/>" + "Fecha
nacimiento = " + fecha_nacimiento + "<br/>" + "Codigo postal = " + codigo_postal + "<br/>" +
"Telefono = " + telefono;
   }
  }
}
```

El primer cambio importante es el de obtener el contenido de la respuesta del servidor. Hasta ahora, siempre se utilizaba la propiedad `responseText`, que devuelve el texto simple que incluye la respuesta del servidor. Cuando se procesan respuestas en formato XML, se debe utilizar la propiedad `responseXML`.

El valor devuelto por `responseXML` es un documento XML que contiene la respuesta del servidor. Como se trata de un documento XML, es posible utilizar con sus contenidos todas las funciones DOM que se vieron en el capítulo correspondiente a DOM.

Aunque el manejo de repuestas XML es mucho más pesado y requiere el uso de numerosas funciones DOM, su utilización se hace imprescindible para procesar respuestas muy complejas o respuestas recibidas por otros sistemas que exportan sus respuestas internas a un formato estándar XML.

El mecanismo para obtener los datos varía mucho según cada documento XML, pero en general, se trata de obtener el valor almacenado en algunos elementos XML que a su vez pueden ser descendientes de otros elementos. Para obtener el primer elemento que se corresponde con una etiqueta XML, se utiliza la siguiente instrucción:

```
var elemento = root.getElementsByTagName("nombre_etiqueta")[0];
```

En este caso, se busca la primera etiqueta `<nombre_etiqueta>` que se encuentra dentro del elemento `root` (en este caso se trata de la raíz del documento XML). Para ello, se buscan todas las etiquetas `<nombre_etiqueta>` del documento y se obtiene la primera mediante `[0]`, que corresponde al primer elemento del array de elementos.

Una vez obtenido el elemento, para obtener su valor se debe acceder a su primer nodo hijo (que es el nodo de tipo texto que almacena el valor) y obtener la propiedad `nodeValue`, que es la propiedad que guarda el texto correspondiente al valor de la etiqueta:

```
var valor = elemento.firstChild.nodeValue;
```

Normalmente, las dos instrucciones anteriores se unen en una sola instrucción:

```
var tfno = parametros.getElementsByTagName("telefono")[0].firstChild.nodeValue;
```

JSON

Aunque el formato XML está soportado por casi todos los lenguajes de programación, por muchas aplicaciones y es una tecnología madura y probada, en algunas ocasiones es más útil intercambiar información con el servidor en formato JSON.

JSON es un formato mucho más compacto y ligero que XML. Además, es mucho más fácil de procesar en el navegador del usuario. Afortunadamente, cada vez existen más utilidades para procesar y generar el formato JSON en los diferentes lenguajes de programación del servidor (PHP, Java, C#, etc.)

El ejemplo mostrado anteriormente para procesar las respuestas XML del servidor se puede reescribir utilizando respuestas JSON. En este caso, la respuesta que genera el servidor es mucho más concisa:

```
{
mensaje: "...",
parametros: {telefono: "...", codigo_postal: "...", fecha_nacimiento: "..." }
}
```

Considerando el nuevo formato de la respuesta, es necesario modificar la función que se encarga de procesar la respuesta del servidor:

```
function procesaRespuesta() {
  if(http_request.readyState == READY_STATE_COMPLETE) {
  if(http_request.status == 200) {
    var respuesta_json = http_request.responseText;
    var objeto_json = eval("("+respuesta_json+")");

    var mensaje = objeto_json.mensaje;

    var telefono = objeto_json.parametros.telefono;
    var fecha_nacimiento = objeto_json.parametros.fecha_nacimiento;
    var codigo_postal = objeto_json.parametros.codigo_postal;

    document.getElementById("respuesta").innerHTML = mensaje + "<br>" + "Fecha nacimiento
= " + fecha_nacimiento + "<br>" + "Codigo postal = " + codigo_postal + "<br>" + "Telefono = " +
telefono;
    }
  }
}
```

La respuesta JSON del servidor se obtiene mediante la propiedad `responseText`:

```
var respuesta_json = http_request.responseText;
```

Sin embargo, esta propiedad solamente devuelve la respuesta del servidor en forma de cadena de texto. Para trabajar con el código JSON devuelto, se debe transformar esa cadena de texto en un objeto JSON. La forma más sencilla de realizar esa conversión es mediante la función `eval()`, en la que deben añadirse paréntesis al principio y al final para realizar la evaluación de forma correcta:

```
var objeto_json = eval("("+respuesta_json+")");
```

Una vez realizada la transformación, el objeto JSON ya permite acceder a sus métodos y propiedades mediante la notación de puntos tradicional. Comparado con las respuestas XML, este procedimiento permite acceder a la información devuelta por el servidor de forma mucho más simple:

```
// Con JSON
var fecha_nacimiento = objeto_json.parametros.fecha_nacimiento;

// Con XML
var parametros = root.getElementsByTagName("parametros")[0];
var fecha_nacimiento =
parametros.getElementsByTagName("fecha_nacimiento")[0].firstChild.nodeValue;
```

También es posible el envío de los parámetros en formato JSON. Sin embargo, no es una tarea tan sencilla como la creación de un documento XML. Así, se han diseñado utilidades específicas para la transformación de objetos JavaScript a cadenas de texto que representan el objeto en formato JSON. Esta librería se puede descargar desde el sitio web www.json.org.

Para emplearla, se añade la referencia en el código de la página:

```
<script type="text/javascript" src="json.js"></script>
```

Una vez referenciada la librería, se emplea el método `stringify` para realizar la transformación:

```
var objeto_json = JSON.stringify(objeto);
```

Además de las librerías para JavaScript, están disponibles otras librerías para muchos otros lenguajes de programación habituales. Empleando la librería desarrollada para Java, es posible procesar la petición JSON realizada por un cliente:

```
import org.json.JSONObject;
...
String cadena_json = "{propiedad: valor, codigo_postal: otro_valor}";
JSONObject objeto_json = new JSONObject(cadena_json);
String codigo_postal = objeto_json.getString("codigo_postal");
```

2. Envío de parámetros con la petición HTTP

Hasta ahora, el objeto `XMLHttpRequest` se ha empleado para realizar peticiones HTTP sencillas. Sin embargo, las posibilidades que ofrece el objeto `XMLHttpRequest` son muy superiores, ya que también permite el envío de parámetros junto con la petición HTTP.

El objeto `XMLHttpRequest` puede enviar parámetros tanto con el método GET como con el método POST de HTTP. En ambos casos, los parámetros se envían como una serie de pares clave/valor concatenados por símbolos `&`. El siguiente ejemplo muestra una URL que envía parámetros al servidor mediante el método GET:

http://localhost/aplicacion?parametro1=valor1&parametro2=valor2&parametro3=valor3

La principal diferencia entre ambos métodos es que mediante el método POST los parámetros se envían en el cuerpo de la petición y mediante el método GET los parámetros se concatenan a la URL accedida. El método GET se utiliza cuando se accede a un recurso que depende de la información proporcionada por el usuario. El método POST se utiliza en operaciones que crean, borran o actualizan información.

Técnicamente, el método GET tiene un límite en la cantidad de datos que se pueden enviar. Si se intentan enviar más de 512 bytes mediante el método GET, el servidor devuelve un error con código `414` y mensaje `Request-URI Too Long` "*La URI de la petición es demasiado larga*").

Cuando se utiliza un elemento `<form>` de HTML, al pulsar sobre el botón de envío del formulario, se crea automáticamente la cadena de texto que contiene todos los parámetros que se envían al servidor. Sin embargo, el objeto `XMLHttpRequest` no dispone de esa posibilidad y la cadena que contiene los parámetros se debe construir manualmente.

A continuación se incluye un ejemplo del funcionamiento del envío de parámetros al servidor. Se trata de un formulario con tres campos de texto que se validan en el servidor mediante AJAX. El código HTML también incluye un elemento `<div>` vacío que se utiliza para mostrar la respuesta del servidor:

```
<form>
 <label for="fecha_nacimiento">Fecha de nacimiento:</label>
```

```
 <input type="text" id="fecha_nacimiento" name="fecha_nacimiento"
/><br/>

 <label for="codigo_postal">Codigo postal:</label>
 <input type="text" id="codigo_postal" name="codigo_postal" /><br/>

 <label for="telefono">Telefono:</label>
 <input type="text" id="telefono" name="telefono" /><br/>

 <input type="button" value="Validar datos" />
</form>

<div id="respuesta"></div>
```

El código anterior produce la siguiente página:

Formulario de ejemplo

El código JavaScript necesario para realizar la validación de los datos en el servidor se muestra a continuación:

```
var READY_STATE_COMPLETE=4;
var peticion_http = null;

function inicializa_xhr() {
 if(window.XMLHttpRequest) {
  return new XMLHttpRequest();
 }
 else if(window.ActiveXObject) {
  return new ActiveXObject("Microsoft.XMLHTTP");
 }
}

function crea_query_string() {
 var fecha = document.getElementById("fecha_nacimiento");
 var cp = document.getElementById("codigo_postal");
 var telefono = document.getElementById("telefono");

 return "fecha_nacimiento=" + encodeURIComponent(fecha.value) +
    "&codigo_postal=" + encodeURIComponent(cp.value) +
    "&telefono=" + encodeURIComponent(telefono.value) +
    "&nocache=" + Math.random();
}
```

```
function valida() {
 peticion_http = inicializa_xhr();
 if(peticion_http) {
  peticion_http.onreadystatechange = procesaRespuesta;
  peticion_http.open("POST", "http://localhost/validaDatos.php", true);

  peticion_http.setRequestHeader("Content-Type", "application/x-www-form-
urlencoded");
  var query_string = crea_query_string();
  peticion_http.send(query_string);
 }
}

function procesaRespuesta() {
 if(peticion_http.readyState == READY_STATE_COMPLETE) {
  if(peticion_http.status == 200) {
   document.getElementById("respuesta").innerHTML = peticion_http.responseText;
  }
 }
}
```

La clave del ejemplo anterior se encuentra en estas dos líneas de código:

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

En primer lugar, si no se establece la cabecera `Content-Type` correcta, el servidor descarta todos los datos enviados mediante el método POST. De esta forma, al programa que se ejecuta en el servidor no le llega ningún parámetro. Así, para enviar parámetros mediante el método POST, es obligatorio incluir la cabecera `Content-Type` mediante la siguiente instrucción:

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

Por otra parte, el método `send()` es el que se encarga de enviar los parámetros al servidor. En todos los ejemplos anteriores se utilizaba la instrucción `send(null)` para indicar que no se envían parámetros al servidor. Sin embargo, en este caso la petición si que va a enviar los parámetros.

Como ya se ha comentado, los parámetros se envían en forma de cadena de texto con las variables y sus valores concatenados mediante el símbolo `&` (esta cadena normalmente se conoce como "*query string*").

La cadena con los parámetros se construye manualmente, para lo cual se utiliza la función `crea_query_string()`:

```
function crea_query_string() {
 var fecha = document.getElementById("fecha_nacimiento");
 var cp = document.getElementById("codigo_postal");
 var telefono = document.getElementById("telefono");

 return "fecha_nacimiento=" + encodeURIComponent(fecha.value) +
    "&codigo_postal=" + encodeURIComponent(cp.value) +
    "&telefono=" + encodeURIComponent(telefono.value) +
    "&nocache=" + Math.random();
}
```

La función anterior obtiene el valor de todos los campos del formulario y los concatena junto con el nombre de cada parámetro para formar la cadena de texto que se envía al servidor. El uso de la función `encodeURIComponent()` es imprescindible para evitar problemas con algunos caracteres especiales.

La función `encodeURIComponent()` reemplaza todos los caracteres que no se pueden utilizar de forma directa en las URL por su representación hexadecimal. Las letras, números y los caracteres `- _ . ! ~ * ' ( )` no se modifican, pero todos los demás caracteres se sustituyen por su equivalente hexadecimal.

Las sustituciones más conocidas son las de los espacios en blanco por `%20`, y la del símbolo `&` por `%26`. Sin embargo, como se muestra en el siguiente ejemplo, también se sustituyen todos los acentos y cualquier otro carácter que no se puede incluir directamente en una URL:

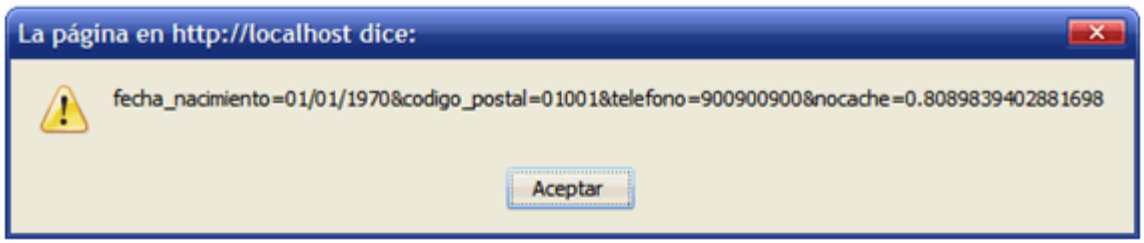
```
var cadena = "cadena de texto";
var cadena_segura = encodeURIComponent(cadena);
// cadena_segura = "cadena%20de%20texto";

var cadena = "otra cadena & caracteres problemáticos / : =";
var cadena_segura = encodeURIComponent(cadena);
// cadena_segura =
"otra%20cadena%20%26%20caracteres%20problem%C3%A1ticos%20%2F%20%3A%
20%3D";
```

JavaScript incluye una función contraria llamada `decodeURIComponent()` y que realiza la transformación inversa. Además, también existen las funciones `encodeURI()` y `decodeURI()` que codifican/decodifican una URL completa. La principal diferencia entre `encodeURIComponent()` y

`encodeURI()` es que esta última no codifica los caracteres `; / ? : @ & = + $ , #`:

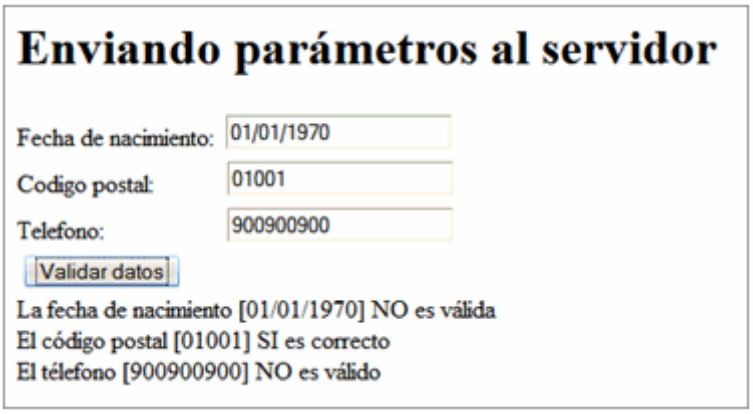
```
var cadena = "http://www.ejemplo.com/ruta1/index.php?parametro=valor con ñ y &";
var cadena_segura = encodeURIComponent(cadena);
// cadena_segura = "http%3A%2F%2Fwww.ejemplo.com%2Fruta1%2Findex.php%3Fparametro%3Dvalor%20con%20%C3%B1%20y%20%26";

var cadena_segura = encodeURI(cadena); // cadena_segura = "http://www.ejemplo.com/ruta1/index.php?parametro=valor%20con%20%C3%B1%20y%20";
```

Por último, la función `crea_query_string()` añade al final de la cadena un parámetro llamado `nocache` y que contiene un número aleatorio (creado mediante el método `Math.random()`). Añadir un parámetro aleatorio adicional a las peticiones GET y POST es una de las estrategias más utilizadas para evitar problemas con la caché de los navegadores. Como cada petición varía al menos en el valor de uno de los parámetros, el navegador está obligado siempre a realizar la petición directamente al servidor y no utilizar su cache. A continuación se muestra un ejemplo de la *query string* creada por la función definida:



Query String creada para el formulario de ejemplo

En este ejemplo sencillo, el servidor simplemente devuelve el resultado de una supuesta validación de los datos enviados mediante AJAX:



Mostrando el resultado devuelto por el servidor

En las aplicaciones reales, las validaciones de datos mediante AJAX sólo se utilizan en el caso de validaciones complejas que no se pueden realizar mediante el uso de código JavaScript básico. En general, las validaciones complejas requieren el uso de bases de datos: comprobar que un nombre de usuario no esté previamente registrado, comprobar que la localidad se corresponde con el código postal indicado, etc.

XML

La flexibilidad del objeto `XMLHttpRequest` permite el envío de los parámetros por otros medios alternativos a la tradicional *query string*. De esta forma, si la aplicación del servidor así lo requeire, es posible realizar una petición al servidor enviando los parámetros en formato XML.

A continuación se modifica el ejemplo anterior para enviar los datos del usuario en forma de documento XML. En primer lugar, se modifica la llamada a la función que construye la *query string*:

```
function valida() {
  peticion_http = inicializa_xhr();
  if(peticion_http) {
    peticion_http.onreadystatechange = procesaRespuesta;
    peticion_http.open("POST", "http://localhost/validaDatos.php", true);
    var parametros_xml = crea_xml();
    peticion_http.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
    peticion_http.send(parametros_xml);
  }
}
```

Seguidamente, se crea la función `crea_xml()` que se encarga de construir el documento XML que contiene los parámetros enviados al servidor:

```
function crea_xml() {
```

```
 var fecha = document.getElementById("fecha_nacimiento");
 var cp = document.getElementById("codigo_postal");
 var telefono = document.getElementById("telefono");

 var xml = "<parametros>";
 xml = xml + "<fecha_nacimiento>" + fecha.value + "<\/fecha_nacimiento>";
 xml = xml + "<codigo_postal>" + cp.value + "<\/codigo_postal>";
 xml = xml + "<telefono>" + telefono.value + "<\/telefono>";
 xml = xml + "<\/parametros>";
 return xml;
}
```

El código de la función anterior emplea el carácter `\` en el cierre de todas las etiquetas XML. El motivo es que las etiquetas de cierre XML y HTML (al contrario que las etiquetas de apertura) se interpretan en el mismo lugar en el que se encuentran, por lo que si no se incluyen esos caracteres `\` el código no validaría siguiendo el estándar XHTML de forma estricta.

El método `send()` del objeto `XMLHttpRequest` permite el envío de una cadena de texto y de un documento XML. Sin embargo, en el ejemplo anterior se ha optado por una solución intermedia: una cadena de texto que representa un documento XML. El motivo es que no existe a día de hoy un método robusto y que se pueda emplear en la mayoría de navegadores para la creación de documentos XML completos.