INTERACCIÓN CON EL SERVIDOR

Envío de parámetros 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.

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:

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-w
ww-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_htt
p.responseText;

    }

  }

}
```

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

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

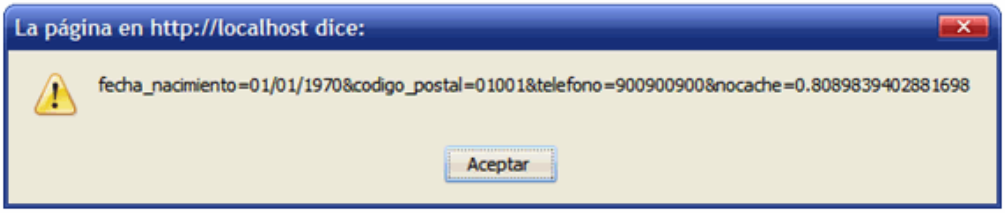
Por otra parte, el método `send()` es el que se encarga de enviar los parámetros al servidor.

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()`:

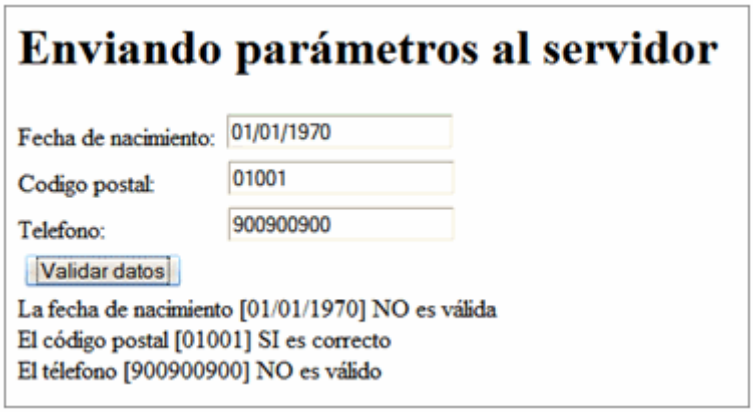
La función anterior obtiene el valor de todos los campos del formulario y los concatena junto con el nombre de cada parámetro para formar la cadena de texto que se envía al servidor. El uso de la función `encodeURIComponent()` es imprescindible para evitar problemas con algunos caracteres especiales.

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.

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:



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



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.

EJERCICIO 1

Un ejemplo de validación compleja es la que consiste en comprobar si un nombre de usuario escogido está libre o ya lo utiliza otro usuario. Como es una validación que requiere el uso de una base de datos muy grande, no se puede realizar en el navegador del cliente. Utilizando las técnicas mostradas anteriormente y la página web que se proporciona:

1. Crear un script que compruebe con AJAX y la ayuda del servidor si el nombre escogido por el usuario está libre o no.
2. El script del servidor se llama `compruebaDisponibilidad.php` y el parámetro que contiene el nombre se llama `login`.
3. La respuesta del servidor es `"si"` o `"no"`, en función de si el nombre de usuario está libre y se puede utilizar o ya ha sido ocupado por otro usuario.
4. A partir de la respuesta del servidor, mostrar un mensaje al usuario indicando el resultado de la comprobación.