DWC

(Desarrollo Web en entorno cliente)

Índice

1.- Formularios

La programación de aplicaciones que contienen formularios web siempre ha sido una de las tareas fundamentales de JavaScript. De hecho, una de las principales razones por las que se inventó el lenguaje de programación JavaScript fue la necesidad de validar los datos de los formularios directamente en el navegador del usuario. De esta forma, se evitaba recargar la página cuando el usuario cometía errores al rellenar los formularios.

No obstante, la aparición de las aplicaciones AJAX ha relevado al tratamiento de formularios como la principal actividad de JavaScript. Ahora, el principal uso de JavaScript es el de las comunicaciones asíncronas con los servidores y el de la manipulación dinámica de las aplicaciones. De todas formas, el manejo de los formularios sigue siendo un requerimiento imprescindible para cualquier programador de JavaScript.
Los formularios y los elementos o controles que contienen son utilizados para diseñar los interfaces de comunicación con los usuarios, así como para mandar información al servidor.

2.- Acceso a los formularios y sus elementos

JavaScript dispone de numerosas propiedades y funciones que facilitan la programación de aplicaciones que manejan formularios.
Lo primero de todo para poder trabajar en un script con un formulario o con un elemento de formulario es seleccionarlo.

Hay diferentes formas de acceso

2.1.- Acceso mediante el array forms

En primer lugar, cuando se carga una página web, el navegador crea automáticamente un **array llamado forms** y que contiene la referencia a todos los formularios de la página.

Para acceder al array forms, se utiliza el objeto document, por lo que **document.forms** es el array que contiene todos los formularios de la página.

Como se trata de un array, el acceso a cada formulario se realiza con la misma sintaxis de los arrays. La siguiente instrucción accede al primer formulario de la página:

```
document.forms[0];
```

Además del array de formularios, el navegador crea automáticamente un array llamado elements por cada uno de los formularios de la página. Cada array elements contiene la referencia a todos los elementos (cuadros de texto, botones, listas desplegables, etc.) de ese formulario. Utilizando la sintaxis de los arrays, la siguiente instrucción obtiene el primer elemento del primer formulario de la página:

```
document.forms[0].elements[0];
```

La sintaxis de los arrays no siempre es tan concisa. El siguiente ejemplo muestra cómo obtener directamente el último elemento del primer formulario de la página:

```
document.forms[0].elements[document.forms[0].elements.length-1];
```

Aunque esta forma de acceder a los formularios es rápida y sencilla, tiene un inconveniente muy grave. ¿Qué sucede si cambia el diseño de la página y en el código HTML se cambia el orden de los formularios originales o se añaden nuevos formularios? El problema es que "el primer formulario de la página" ahora podría ser otro formulario diferente al que espera la aplicación.

En un entorno tan cambiante como el diseño web, es muy difícil confiar en que el orden de los formularios se mantenga estable en una página web. ***Por este motivo, siempre debería evitarse el acceso a los formularios de una página mediante el array document.forms.***

2.2.- Acceso por nombre

Una forma de evitar los problemas del método anterior consiste **en acceder a los formularios de una página a través de su nombre (atributo name) o a través de su atributo id**.

El objeto document permite acceder directamente a cualquier formulario mediante su atributo name:

```
var formularioPrincipal = document.formulario;
var formularioSecundario = document.otro_formulario;

<form name="formulario" >
</form>

<form name="otro_formulario" >
</form>
```

Accediendo de esta forma a los formularios de la página, el script funciona correctamente, aunque se reordenen los formularios o se añadan nuevos formularios a la página.

Los elementos de los formularios también se pueden acceder directamente mediante su atributo name:

```
var formularioPrincipal = document.formulario;
var primerElemento = document.formulario.elemento;

<form name="formulario">
  <input type="text" name="elemento" />
</form>
```

2.3.-Acceso mediante el DOM

Obviamente, también se puede acceder a los formularios y a sus elementos utilizando las funciones DOM de acceso directo a los nodos. El siguiente ejemplo utiliza la habitual función document.getElementById() para acceder de forma directa a un formulario y a uno de sus elementos:

```
var formularioPrincipal = document.getElementById("formulario");
var primerElemento = document.getElementById("elemento");

<form name="formulario" id="formulario" >
  <input type="text" name="elemento" id="elemento" />
</form>
```

La mayoría de veces se utiliza esta alternativa por ser la más sencilla y legible en el código de nuestros scripts

3.- Propiedades generales

Independientemente del método utilizado para obtener la referencia a un elemento de formulario, cada elemento dispone de las siguientes propiedades útiles para el desarrollo de las aplicaciones:

- **type:** indica el tipo de elemento que se trata. Para los elementos de tipo <input> (text, button, checkbox, etc.) coincide con el valor de su atributo type. Para las listas desplegables normales (elemento <select>) su valor es select-one, lo que permite diferenciarlas de las listas que permiten seleccionar varios elementos a la vez y cuyo tipo es select-multiple. Por último, en los elementos de tipo <textarea>, el valor de type es textarea.

- **form:** es una referencia directa al formulario al que pertenece el elemento. Así, para acceder al formulario de un elemento, se puede utilizar document.getElementById("id_del_elemento").form

- **name**: obtiene el valor del atributo name de XHTML. Solamente se puede leer su valor, por lo que no se puede modificar.

- **value:** permite leer y modificar el valor del atributo value de XHTML. Para los campos de texto (<input type="text"> y <textarea>) obtiene el texto que ha escrito el usuario. Para los botones obtiene el texto que se muestra en el botón. Para los elementos checkbox y radiobutton no es muy útil, como se verá más adelante

4.- Eventos más utilizados

Por último, los eventos más utilizados en el manejo de los formularios son los siguientes:

- **onclick:** evento que se produce cuando se pincha con el ratón sobre un elemento. Normalmente se utiliza con cualquiera de los tipos de botones que permite definir XHTML (<input type="button">, <input type="submit">, <input type="image">).

- **onchange:** evento que se produce cuando el usuario cambia el valor de un elemento de texto (<input type="text"> o <textarea>). Sin embargo, el evento sólo se produce si después de realizar el cambio, el usuario pasa al siguiente campo del formulario, lo que técnicamente se conoce como que "el otro campo de formulario ha perdido el foco".
  También se produce cuando el usuario selecciona una opción en una lista desplegable (<select>)

- **onfocus:** evento que se produce cuando el usuario selecciona un elemento del formulario.

- **onblur:** evento complementario de onfocus, ya que se produce cuando el usuario ha deseleccionado un elemento por haber seleccionado otro elemento del formulario. Técnicamente, se dice que el elemento anterior "ha perdido el foco".

5.- Operaciones básicas sobre los elementos

La mayoría de técnicas JavaScript relacionadas con los formularios requieren leer y/o modificar el valor de los campos del formulario. Por tanto, a continuación, se muestra cómo obtener el valor de los campos de formulario más utilizados.

5.1.- Cuadro de texto y textarea

El valor del texto mostrado por estos elementos se obtiene y se establece directamente mediante **la propiedad value**.

```
<input type="text" id="texto" />
var valor = document.getElementById("texto").value;

<textarea id="parrafo"></textarea>
var valor = document.getElementById("parrafo").value;
```

Hay veces en las que queremos que un cuadro de texto no se vea para ello deberemos usar el type **hidden**

```
<input type="hidden" id="custId" name="custId" value="3487">
```

En otros casos nos puede interesar que sea sólo de lectura.

```
<input type="text" id="country" name="country" value="Norway" readonly>
```

5.2.- Radiobutton

Cuando se dispone de un grupo de radiobuttons, generalmente no se quiere obtener el valor del atributo value de alguno de ellos, sino que lo importante es conocer cuál de todos los radiobuttons se ha seleccionado. La propiedad **checked devuelve true para el radiobutton seleccionado y false en cualquier otro caso**.

Si por ejemplo se dispone del siguiente grupo de radiobuttons:

```
<input type="radio" value="si" name="pregunta" id="pregunta_si"/> SI
<input type="radio" value="no" name="pregunta" id="pregunta_no"/> NO
<input type="radio" value="nsnc" name="pregunta" id="pregunta_nsnc"/> NS/NC
```

El siguiente código permite determinar si cada radiobutton ha sido seleccionado.

```
var elementos = document.getElementsByName("pregunta");
for(var i=0; i<elementos.length; i++) {
 alert("Elemento: " + elementos[i].value + "\n Seleccionado: " + elementos[i].checked);
}
```

Generalmente lo que tenemos que hacer es recorrer la colección de los radiobutton bajo el mismo nombre y comprobar cuál de ellos esta seleccionado mediante la propiedad **checked==true,** una vez que sabemos cuál es podemos acceder a su valor mediante la propiedad value de ese radiobutton.

La complicación reside en que al ser un control con varias opciones mutuamente excluyentes, los radiobutton deben tener el mismo nombre

5.3.- Checkbox

Los elementos de tipo checkbox son muy similares a los radiobutton, salvo que en este caso se debe comprobar cada checkbox de forma independiente del resto. El motivo es que los grupos de radiobutton son mutuamente excluyentes y sólo se puede seleccionar uno de ellos cada vez. Por su parte, los checkbox se pueden seleccionar de forma independiente respecto de los demás.

Si se dispone de los siguientes checkbox:

```
<input type="checkbox" value="condiciones" name="condiciones" id="condiciones"/> He leído y acepto las condiciones
<input type="checkbox" value="privacidad" name="privacidad" id="privacidad"/> He leído la política de privacidad
```

Utilizando la propiedad checked, es posible comprobar si cada checkbox ha sido seleccionado:

```
var elemento = document.getElementById("condiciones");
alert(" Elemento: " + elemento.value + "\n Seleccionado: " + elemento.checked);

elemento = document.getElementById("privacidad");
alert(" Elemento: " + elemento.value + "\n Seleccionado: " + elemento.checked);
```

5.4.- Select

Las listas desplegables (<select>) son los elementos en los que más propiedades tenemos. Un select o lista, podemos verlo como un array de elementos (versión gráfica) en los que cada elemento (option) ocupa una posición y tiene dos propiedades (**text**, lo que se en la lista y **value**, el valor que tiene esa opción). Cuando seleccionamos una opcion nueva de la lista se ejecuta el evento **onChange** y el valor del select es la propiedad **value**.

Si se dispone de una lista desplegable como la siguiente:

```
<select id="lista" name="opciones">
  <option value="1">Primer valor</option>
  <option value="2">Segundo valor</option>
  <option value="3">Tercer valor</option>
  <option value="4">Cuarto valor</option>
</select>
```

Si el usuario selecciona una opción de la lista el valor del select se puede obtener de la siguiente forma:

```
document.getElementById(“lista”).value
```

Si queremos ejecutar algo cada vez que se seleccione un elemento distinto del select deberemos incluir las acciones en el evento onChange del select

```
<select id="lista" name="opciones">
  <option value="1">Primer valor</option>
  <option value="2">Segundo valor</option>
  <option value="3">Tercer valor</option>
  <option value="4">Cuarto valor</option>
</select>

<script>
  let milista=document.getElementById("lista")
  milista.addEventListener("change",()=>alert(milista.value));
</script>
```

5.4.1.- Acceso a cualquier elemento del Select

En general, lo que se requiere es obtener el valor del atributo value de la opción (<option>) seleccionada por el usuario y generalmente cuando se cambie el option seleccionado, es decir en el evento onChange como hemos comentado anteriormente.

En algunas ocasiones igual necesitamos trabajar con una lista y realizar alguna de estas operaciones:

- Saber cuál es el elemento seleccionado y la posición del elemento en la lista.
- Acceder a un elemento de la lista en una posición determinada.
- Modificar desde código el elemento seleccionado de la lista

Estas operaciones ya no son tan sencillas como obtener el valor que devuelve una lista, ya que deben utilizarse las siguientes propiedades:

- **options,** es un array creado automáticamente por el navegador para cada lista desplegable y que contiene la referencia a todas las opciones de esa lista. De esta forma, la primera opción de una lista se puede obtener mediante document.getElementById("id_de_la_lista").options[0].

- **selectedIndex**, cuando el usuario selecciona una opción, el navegador actualiza automáticamente el valor de esta propiedad, que guarda el índice de la opción seleccionada. El índice hace referencia al array options creado automáticamente por el navegador para cada lista. Si no hay ningún elemento seleccionado esta propiedad devuelve -1

```
// Obtener la referencia a la lista
var milista = document.getElementById("lista");

// Obtener el índice de la opción que se ha seleccionado
var indiceSeleccionado = milista.selectedIndex;
// Con el índice y el array "options", obtener la opción seleccionada
var opcionSeleccionada = milista.options[indiceSeleccionado];
```

```
// Obtener el valor y el texto de la opción seleccionada
var textoSeleccionado = opcionSeleccionada.text;
var valorSeleccionado = opcionSeleccionada.value;

alert("Opción seleccionada: " + textoSeleccionado + "\n Valor de la opción:
" + valorSeleccionado);
```

Como se ha visto, para obtener el valor del atributo value correspondiente a la opción seleccionada por el usuario, es necesario realizar varios pasos. No obstante, normalmente se abrevian todos los pasos necesarios en una única instrucción:

```
// Acceder a lista
var milista = document.getElementById("lista");

// Obtener el valor de la opción seleccionada
var valorSeleccionado = milista.options[milista.selectedIndex].value;

// Obtener el texto que muestra la opción seleccionada
var valorSeleccionado = milista.options[milista.selectedIndex].text;
```

Lo más importante es no confundir el valor de la propiedad selectedIndex con el valor correspondiente a la propiedad value de la opción seleccionada. En el ejemplo anterior, la primera opción tiene un value igual a 1. Sin embargo, si se selecciona esta opción, el valor de selectedIndex será 0, ya que es la primera opción del array options (y los arrays empiezan a contar los elementos en el número 0).

5.4.2.-Añadir elementos a un Select

Trabajando con listas es muy habitual crear una lista de forma dinámica o tener una lista ya creada y poder añadir elementos de la lista.

Esto lo podemos hacer de una forma sencilla utilizando los métodos vistos en el DOM.

La idea es crear un elemento option por cada uno de los elementos a añadir en la lista y para cada uno de ellos definir su propiedad value y text.

Una vez realizado esto deberemos añadir el option como hijo del select donde queramos que se cree la nueva opción de la lista.

```
<select id="lista" name="opciones">
</select>
<button id="boton">Añadir elemento</button>
<script>

 var milista = document.getElementById("lista");
 var miboton = document.getElementById("boton");

 miboton.addEventListener("click",addLista);

 function addLista(){
  let texto=prompt("Dime Text");
  let valor=prompt("Dime Value");
  mioption=document.createElement("option");
  mioption.value=valor;
  mioption.text=texto;

  milista.appendChild(mioption);
 }
</script>
```

5.4.3 Métodos propios del Select

El element select dispone de métodos propios para insertar y eliminar. Son adicionales a los métodos generales del DOM.

La ventaja que ofrecen frente al DOM es que permiten indicar directamente la posición para añadir o eliminar un elemento option en el select

5.4.3.1 Añadir elementos

Se utiliza el método add. La sintaxis es la siguiente:

```
selectObject.add(option, index)
```

- **option:** el elemento option a añadir al select
- **index:** es opcional e indica la posición en la que se añade el elemento option, si no se indica se añade al final del select.

Este ejemplo añade un elemento nuevo a la lista en la posición 2 (tercer elemento pués comienza por 0)

```
<select id="lista" name="opciones">
  <option value="1">Primer valor</option>
  <option value="2">Segundo valor</option>
  <option value="3">Tercer valor</option>
  <option value="4">Cuarto valor</option>
</select>
<button id="boton">Añadir elemento</button>
<script>

  var milista = document.getElementById("lista");
  var miboton = document.getElementById("boton");

  miboton.addEventListener("click",addLista);

  function addLista(){
   let texto=prompt("Dime Text");
   let valor=prompt("Dime Value");
   mioption=document.createElement("option");
   mioption.value=valor;
   mioption.text=texto;

   milista.add(mioption,2);
  }
</script>
```

5.4.3.2.- Eliminar elementos.

Se utiliza el método remove. La sintaxis es la siguiente:

```
selectObject.remove(index)
```

- **index:** La posición del option a eliminar del select.

Elimina el elemento en el índice 2 de la lista, elimina el elemento que aparece en la posición 3 del desplegable

```
let milista= document.getElementById("lista");
milista.remove(2);
```

Elimina el último elemento de la lista.

```
let milista= document.getElementById("lista");
if (milista.length > 0) {
  milista.remove(x.length-1);
}
```