De: http://geneura.ugr.es/~victor/cursillos/javascript/js_document.html

Document

En este capítulo veremos como se puede usar JavaScript para acceder a cada uno de los elementos que se encuentran dentro de un documento. Esto permite, por ejemplo, mejorar la introducción de datos mediante formularios o personalizar las páginas mediante respuestas a los eventos que puede provocar el usuario.

1. Accediendo a los nodos con `getElementById`

Cada vez que un navegador lee un fichero html, genera interiormente lo que se conoce como el DOM (*Document Object Model*), que es básicamente un árbol en el que está representada toda la información que contiene el documento: textos, enlaces, imágenes, formularios, etc.
Para que el árbol sea generado correctamente, es conveniente que se haya utilizado correctamente la sintaxis de HTML y que la página que se está visualizando sea válida; esto es, que todas las etiquetas están cerradas y, sobretodo, correctamente anidadas. En caso de no haberlo hecho así, podemos encontrar que el árbol del DOM no es correcto y por tanto no podremos manipularlo fácilmente desde Javascript.

Podemos acceder a los distintos nodos del árbol DOM para observar sus propiedades o incluso para modificar algunas de ellas. Además, podremos añadir y cambiar o quitar eventos a dichos nodos, de forma que su comportamiento podría variar a lo largo de la visualización del documento, en función de las acciones del usuario.

La forma más rápida de acceder a un nodo es mediante su **identificador**, para lo cual ha tenido que serle asignado dicho identificador en el fichero HTML

Ejemplo 6.1

```
<html>
<head>
<title>Identificando elementos </title>
</head>
<body>
<h3 id="intro">Introducción

<script>
var nodo=document.getElementById( "intro" )
</script>
</body>
</html>
```

Como se observa en el ejemplo anterior, una vez identificado un elemento se puede acceder a él mediante la función `document.getElementById( identificador )`, que devuelve el nodo cuyo identificdor es el pasado como parámetro.

A partir de este momento, podemos utilizar el nodo del árbol para acceder y modificar sus propiedades, así como para acceder a otros nodos relacionados con él en el árbol (esto es: sus hijos, su padre, sus hermanos...)

En el siguiente ejemplo, ampliamos el código del ejemplo anterior para conseguir que el texto del nodo se vea de color rojo.

Ejemplo 6.2

```
<html>
<head>
```

```
<title>Identificando elementos (II) </title>
</head>
<body>
<h3 id="intro">Introducción

<script>
var nodo=document.getElementById( "intro" )
nodo.style.color='red'
</script>
</body>
</html>
```

Y en el siguiente, añadimos eventos al nodo para que responda cuando el ratón pasa por encima de él.

Ejemplo 6.3

```
<html>
<head>
<title>Identificando elementos (III) </title>
</head>
<body>
<h3 id="intro">Introducción

<script>
var nodo=document.getElementById( "intro" )
nodo.style.color='red'
nodo.onmouseover=function() {
  this.style.color='blue'
}
</script>
</body>
</html>
```

Por último, en este otro ejemplo cambiamos la propiedad "src" de una imagen para que muestre una imagen distinta de la original cuando pasa un segundo

```
<html>
<head>
<title>Identificando elementos (III-bis) </title>
</head>
<body>
<img src="1.gif" id="img1">
<script>
setTimeout( "document.getElementById('img1').src='2.gif'", 1000 )
</script>
</body>
</html>
```

De entre las propiedades más importantes de un nodo, destacamos:

- **id**: indica el identificador del nodo.
- **nodeType**: que indica el tipo de nodo que es; es decir, es un número indicando el tipo de nodo que representa. Los códigos de los distintos tipos de nodos son:
    1. **ELEMENT_NODE**
    2. ATTRIBUTE_NODE
    3. **TEXT_NODE**
    4. CDATA_SECTION_NODE
    5. ENTITY_REFERENCE_NODE

6. ENTITY_NODE
7. PROCESSING_INSTRUCTION_NODE
8. COMMENT_NODE
9. DOCUMENT_NODE
10. DOCUMENT_TYPE_NODE
11. DOCUMENT_FRAGMENT_NODE
12. NOTATION_NODE

Como se puede ver, hemos indicado en negrita los tipos 1 y 3 que corresponden a nodos elementos y nodos texto, respectivamente.

- **nodeName**: que devuelve el tipo de etiqueta HTML que representa al nodo. Dado que hay navegadores que lo devuelven en mayúsculas y otros en minúsculas, siempre que queramos comparar es bueno usar el método `toLowerCase` que posee toda cadena (ver ejemplo más abajo).
- **className**: que devuelve el nombre de la clase que haya sido asignado al nodo.
- **nodeValue**: que almacena el texto contenido en los nodos de tipo texto, es `null` para los nodo de tipo elemento.

El siguiente ejemplo muestra estas propiedades para el elemento llamado `intro`.

Ejemplo 6.4

```
<html>
<head>
<title>Identificando elementos (IV) </title>
</head>
<body>
<h3 id="intro" class="titulo">Introducción</h3>

<script>
var nodo=document.getElementById( "intro" )
alert( "PROPIEDADES del nodo intro:\n\n"+
        "Tipo: "+nodo.nodeType+"\n"+
        "Nombre: "+nodo.nodeName.toLowerCase()+"\n"+
        "Clase: "+nodo.className+"\n"+
        "Valor: "+nodo.nodeValue )
</script>
</body>
</html>
```

2. Accediendo a colecciones de nodos con `getElementsByTagName`

De la misma forma que hemos accedido a un solo nodo por su nombre, podemos acceder a una colección de nodos por el hecho de ser todos del mismo tipo. Para ello usaremos el método `getElementsByTagName`, y posteriormente podremos acceder a cada uno de los nodos que dicho método devuelve iterando a través de sus elementos como lo hacemos con cualquier matriz.

A diferencia de `getElementById` (que es un método de `document`), `getElementsByTagName` es un método de todo nodo, por lo que podemos seleccionar todos los nodos de un determinado tipo (si lo aplicamos a `document`) o sílo aquellos de dicho tipo pero que cuelgan de un determinado nodo (como por ejemplo en `miNodo.getElementsByTagName`).

Destacar que el parámetro que acepta `getElementsByTagName` es una cadena, como por ejemplo `getElementsByTagName( 'h4')`

El siguiente ejemplo añade un evento a todos los enlaces (esto es, la etiqueta deber ser `a`) para que al pasar por ellos se cambie el color de fondo.

Ejemplo 6.5

```
<html>
<head>
<title>Modificando todos los enlaces de un documento </title>
</head>
<body>
<ul>
  <li><a href="uno.html">El primer enlace</a></li>
  <li class="miClase"><a href="dos.html" target="new">El segundo enlace</a></li>
  <li><a href="tres.html">El tercer enlace</a></li>
</ul>

<script>
var nodos=document.getElementsByTagName( "a" )
for ( var i=0;  i<nodos.length; ++i ) {
  nodos[i].onmouseover=function() {
    this.style.backgroundColor='green'
  }
  nodos[i].onmouseout=function() {
    this.style.backgroundColor=''
  }
}
</script>
</body>
</html>
```

3. Navegación a través del DOM

Una vez accedido a un nodo, podemos navegar a partir de él y acceder a otros nodos a ún cuando no sepamos sus identificadores (o estos no hayan sido definidos). Para ello podemos recorrer el Ã¡rbol del DOM utilizando las siguientes funciones y propiedades:

- **childNodes**: (propiedad) es una matriz que contiene todos los hijos de un nodo (pero no los hijos de éstos).
- **firstChild** y **lastChild** : (propiedades) contienen, respectivamente, el primer y último hijo de un nodo (es decir, son atajos para facilitar el uso de la matriz anterior).
- **parentNode**: (propiedad) permite acceder al nodo padre, esto es, al nodo del que cuelga el actual.
- **previousSibling** y **nextSibling**: (propiedades) permiten acceder a los hermanos del nodo.
- **parentNode**: (propiedad) permite acceder al nodo padre, esto es, al nodo del que cuelga el actual.
- **hasChildNodes()**: (método) devuelve un valor lógico indicando si el nodo tiene o no nodos hijos.

En caso de que no exista una determinada propiedad (por ejemplo, un nodo no tiene hermanos e intentamos acceder a su `previousSibling`), la propiedad tiene valor `null`.

El siguiente ejemplo muestra, para el segundo elemento de tipo `li` del ejemplo 6.5 anterior, el resultado de las anteriores propiedades. (**Atención:** este ejemplo provoca distintos resultados en función del navegador en que se utilice; así, Internet Explore no tendrá en cuenta el texto -saltos de línea- que hay entre cada etiqueta y la siguiente, por ejemplo entre `<ul>` y la primera etiqueta `<li>`, sin embargo otros navegadores sí que lo harán).

Ejemplo 6.6

```
<html>
<head>
<title>Navegando por los elementos del documento </title>
</head>
<body>
```

```
<ul>
  <li><a href="uno.html">El primer enlace</a></li>
  <li class="miClase"><a href="dos.html" target="new">El segundo enlace</a></li>
  <li><a href="tres.html">El tercer enlace</a></li>
</ul>

<script>

var nodo=document.getElementsByTagName( "li" )[1]
document.write( "El propio nodo: ", nodo.nodeName, "<br/>" )
document.write( "Su hermano anterior: ", nodo.previousSibling.nodeName,
"<br/>" )
document.write( "El hermano anterior a ese: ",
nodo.previousSibling.previousSibling.nodeName, "<br/>" )
document.write( "Su hermano posterior: ", nodo.nextSibling.nodeName, "<br/>" )
document.write( "El padre del nodo: ", nodo.parentNode.nodeName, "<br/>" )
document.write( "¿Tiene nodos hijo?: ", nodo.hasChildNodes() , "<br/>" )
document.write( "Número de hijos: ", nodo.childNodes.length, "<br/>" )
document.write( "Primer hijo: ", nodo.firstChild.nodeName, "<br/>" )
document.write( "Último hijo: ", nodo.lastChild.nodeName, "<br/>" )

</script>
</body>
</html
```

4. Acceso a los atributos de un nodo

Además de poder acceder a cada uno de los nodos del DOM, podemos acceder a los atributos de cada nodo, así como modificarlos o añadírselos.

Para este cometido, Javascript implementa dos mÃ©todos que posee todo nodo:

- **getAttribute ('nombre_del_atributo')**: para acceder al valor del atributo
- **setAttribute ('nombre_del_atributo', 'nuevo_valor')**: para modificar el valor del atributo, o añadir nuevos atributos.

Así, en el siguiente ejemplo se acceden, modifican y añaden algunos atributos de los nodos del documento.

Ejemplo 6.7

```
<html>
<head>
<title>Atributos de los nodos</title>
</head>
<body>
<ul>
  <li><a href="uno.html">El primer enlace</a></li>
  <li class="miClase"><a  target="top">El segundo enlace</a></li>
  <li><a href="tres.html">El tercer enlace</a></li>
</ul>

<script>
var nodo=document.getElementsByTagName( "a" )[1]
alert( nodo.getAttribute( 'target' ) )

// Modificamos el destino de un enlace
nodo.setAttribute( 'href', 'http://www.google.es' )

// Añadimos un atributo al nodo
```

```
nodo.setAttribute( 'atributo', 'Soy un atributo!' )

var nodos=document.getElementsByTagName( "li" )

// Asignamos un identificador a un nodo
nodos[1].setAttribute( "id", "nodoA" )
</script>
</body>
</html>
```

Uno de los atributos que podemos cambiar es la clase con que se ha definido el nodo, la cual estará directamente relacionada con la hoja de estilos que se haya definido.

El siguiente ejemplo, crea elementos que pueden estar ocultos o visibles. Para hacerlos "aparecer" y "desaparecer" utilizamos la propiedad `className`, que permite modificar el atributo "class" de cada nodo.

Ejemplo 6.8

```
<html>
<head>
<title>Cambiando la clase de un nodo</title>
</head>
<body>

<style>
.menu_item {
  font-size: 12px; color: red; text-align: left; border: 2px solid red;
}
.submenu_item {
  color: blue; border: 1px solid blue;
}
.submenu_off {
  visibility: hidden; width: 10px;
}
.submenu_on {
  visibility: show;
}
</style>

<span id="m1" class="menu_item">Uno</span>
<span id="m2" class="menu_item">Dos</span>

<br/><br/>
<span id="subm1" class="submenu_off">
  <span id="m1_1" class="submenu_item">Uno -> Uno</span>
  <span id="m1_2" class="submenu_item">Uno -> Dos</span>
</span>
<span id="subm2" class="submenu_off">
  <span id="m2_1" class="submenu_item">Dos -> Uno</span>
  <span id="m2_2" class="submenu_item">Dos -> Dos</span>
</span>

<script>
// Añadimos eventos al menú de primer nivel
for( i=1; i<=2; ++i ) {
  var el=document.getElementById( "m"+i ); //Cada menú se llama m1, m2 ...
  el.onmouseover=function setOn() {
    var auxEl=document.getElementById( "sub"+this.id ); // Cada sumenú: sub1,
sub2...
    auxEl.className="submenu_on";
  }
```

```
  el.onmouseout=function setOn() {
    var auxEl=document.getElementById( "sub"+this.id );
    auxEl.className="submenu_off";
  }
}
</script>
</body>
</html>
```

5. Modificando el contenido de un nodo

En ciertas ocasiones, querremos modificar el contenido de un nodo, de modo que donde diga "Digo", diga "Diego". Para ello podemos optar por dos soluciones: `innerHTML` y utilización del árbol DOM.

innerHTML

`innerHTML` es aparentemente la solución más rápida, aunque no es un estándar del DOM. No obstante, prácticamente todos los navegadores la soportan.

En realidad, `innerHTML` es una propiedad de cada elemento, y está representada como una cadena. Cambiar esta propiedad equivale a cambiar el contenido de dicho elemento, y por tanto, lo que ve el usuario.

El siguiente ejemplo lo muestra más claramente:

Ejemplo 6.9

```
<html>
<head>
<title>Accediendo a la propiedad innerHTML</title>
</head>
<body>
<div id="parrafillo">
Este es un párrafo que contiene muchas cosas:
<ul>
  <li>Una</li>
  <li>lista</li>
</ul>
<table border=1><tr><td>Y una</td><td>tabla!</td></tr></table>
</div>
<script>
function cambia() {
  var el=document.getElementById( "parrafillo");
  el.innerHTML="Y lo cambio completamente"+
    "<ol><li>aunque</li><li>tambiém incluyo</li><li>una lista</li></ol>";
}

setTimeout( "cambia()", 1000 );
</script>
</body>
</html>
```

Aunque algunos sitios de la web desaconsejan el uso de esta propiedad (no está claro que se vaya a convertir enun estándar aprobado por la W3C), parece funcionar en todos los navegadores, y además no afecta a la futura navegabilidad sobre el árbol DOM (al menos en los navegadores que hemos probado).

Utilizando el árbol DOM

Otra forma un poco más enrevesada, pero totalmente aprobada por W3C como estándar, es mediante el uso de las funciones `document.createElement`, `document.createTextNode`, `document.createComment` y `NODO.appendChild`. De esta forma, se van creando los nodos poco a poco y se van añadiendo como hijos de nodos anteriores, generando así la estructura correspondiente en el árbol DOM (también se puede sustituir nodos, borrarlos, etc.)

Ejemplo 6.10

```
<html>
<head>
<title>Modificando nodos con el árbol DOM</title>
</head>
<body>
<div id="parrafillo">
Este es un párrafo que contiene muchas cosas:
<ul>
  <li>Una</li>
  <li>lista</li>
</ul>
<table border=1><tr><td>Y una</td><td>tabla!</td></tr></table>
</div>
<script>
function cambia() {
  var el=document.getElementById( "parrafillo");
  // Primero quitamos el contenido
  while( el.firstChild!=null ) {
    el.removeChild( el.firstChild );
  }
  // A continuación, creamos y añadimos nodos
  var texto=document.createTextNode( "Y lo cambio completamente" )
  el.appendChild( texto );
  var ol=document.createElement( "ol" );
  el.appendChild( ol );
  var li=document.createElement( "li" );
  ol.appendChild( li );
  texto=document.createTextNode( "aunque" );
  li.appendChild( texto );
  li=document.createElement( "li" );
  ol.appendChild( li );
  texto=document.createTextNode( "también incluyo" );
  li.appendChild( texto );
  li=document.createElement( "li" );
  ol.appendChild( li );
  texto=document.createTextNode( "una lista" );
  li.appendChild( texto );
}
setTimeout( "cambia()", 1000 );
</script>
</body>
</html>
```

6. Acceso a los elementos de un formulario

Los formularios son uno de los objetos con los que más se trabaja en JavaScript, dado que en ellos se pueden incluir desde objetos estáticos (texto o imágenes) hasta recuadros de texto o cajas de opciones que permiten al usuario interactuar con la página web. Esta posibilidad de cambiar después de haberse cargado la página, hace que los objetos contenidos en un formulario sean muy

versátiles y fáciles de manejar mediante JavaScript.

Una de las utilidades para las que se usa frecuentemente JavaScript es para comprobar que los datos introducidos en un formulario son correctos. Para ello se comprueban los datos en el momento en que se detecta el evento de enviar el formulario y se procede a dar curso a tal evento si los datos son correctos (devolviendo un valor `true`), o a paralizarlo si hay algún problema (devolviendo el valor `false`).

Existen muchas formas de acceder a un formulario, debido a la evolución que ha ido sufriendo Javascript. Así, podemos definir un formulario como:

```
<form name="nombre_1" id="identificador_1" >
```

Con lo cual podríamos acceder a él de las siguientes formas:

```
document.forms[0] // En caso de ser el primer formulario de la página
document.forms["nombre_1"]
document.nombre_1
document.getElementById( "identificador_1" )
```

De las anteriores formas, desaconsejamos la primera de ellas.

Una vez que hemos accedido a un formulario, podemos acceder y modificar sus propiedades y eventos de la misma forma que hacemos con el resto de nodos. Así, el siguiente ejemplo sólo envía los datos si hemos introducido algún término qué buscar.

Ejemplo 6.11

```
<html>
<head>
<title>Trabajando con formularios</title>
</head>
<body>
Introducimos un mensaje y se busca automáticamente en Google, pero sólo en el
dominio "vrivas.es"
<form name="n1" id="i1" action="http://www.google.es/search">
  <label for="q">Texto de búsqueda:</label>
  <input type="text" name="q" size="40">
  <input type="submit" value="Enviar">
</form>

<script>
el=document.getElementById( "i1" )
el.onsubmit=function test() {
  if ( el.q.value.length>0 ) {
    el.q.value+=" site:www.vrivas.es";
    return true;
  } else {
    return false;
  }
}

</script>
</body>
</html>
```

En el ejemplo anterior, se puede observar que un elemento de un formulario siempre tiene una propiedad llamada "value", a la cual podemos acceder para consultar (como en `if ( el.q.value.length>0 )`) o para modificar (como en `el.q.value+=" site:www.vrivas.es";`).

Una excepción a la norma anterior son los *checkbuttons* y los *radiobuttons*. Aunque tienen su

propiedad "value", la forma correcta de conocer si están pulsados o no es mediante la propiedad "checked", que es de tipo booleano y por lo tanto toma los valores verdadero o falso.

El siguiente ejemplo, comprobamos cómo se puede modificar el estado de un conjunto de *checkbox* al estilo de como lo hacen los lectores de correo electrónico de las webs.

Ejemplo 6.12

```
<html>
<head>
<title>Trabajando con formularios (II)</title>
</head>
<body>
Modificamos checkbox automáticamente
<form name="n1" id="i1" action="http://www.google.es/search">
  <input type="checkbox" name="seleccionar">Seleccionar todos<br/><br/>
  <input type="checkbox" name="linea">Línea 1<br/>
  <input type="checkbox" name="linea">Línea 2<br/>
  <input type="checkbox" name="linea">Línea 3<br/>
</form>

<script>
el=document.getElementById( "i1" )
el.seleccionar.onclick=function cambia() {
  for( var i=0; i<el.linea.length; ++i ) {
    el.linea[i].checked=el.seleccionar.checked;
  }
}
</script>
</body>
</html>
```

Otro aspecto interesante a destacar de los formularios es que cada uno de sus campos puede responder a distintos eventos, tales como `onclick` (como los botones, "radio" y "check") u `onchange` (principalmente los "select" y los campos de tipo texto). Pero todos ellos pueden responder a dos eventos: `onfocus` y `onblur`.

El primero, `onfocus` , se activa cuando el usuario llega al campo, ya sea por que ha pulsado el tabulador, ya sea porque ha pinchado el ratón sobre el campo.

En cuanto al segundo, `onblur` , se activa justo en el caso contrario: el usuario estaba en un campo y hacer cualquier acción que implica salir de ese campo, es decir, que el campo pierda el "foco".

El siguiente ejemplo, pone en mayúsculas un texto al salir de él.

Ejemplo 6.13

```
<html>
<head>
<title>Trabajando con formularios (III)</title>
</head>
<body>
Pasamos texto a mayúsculas
<form name="n1" id="i1" action="http://www.google.es/search">
  Nombre: <input type="text" name="nombre" size="30"><br/>
  <input type="submit" value="Enviar">
</form>

<script>
el=document.getElementById( "i1" )
el.nombre.onblur=function cambia() {
  el.nombre.value=el.nombre.value.toUpperCase();
}
```

```
</script>
</body>
</html>
```