Programación

DOM y excepciones

Decimocuarta semana

Febrero 2022

Cruz García, Iago

Introducción

Hemos hablado mucho sobre el DOM (Document Object Model), la interfaz de acceso al documento, el HTML. Usamos el método más común (getElementByID) pero hay muchos más que debemos conocer para poder dotar de interfaz e interactividad a nuestras páginas web en el futuro. A pesar de poder utilizar JQuery, siempre es buena idea tener presente estas funciones y estructuras para utilizarlas en caso de no poder acceder a JQuery o no realizar todas las funciones que deseamos.

Además veremos las excepciones, formas de controlar los errores de nuestros programas y reproducir los mensajes o código que debe ejecutarse en caso de ocurrir algún error.

Recomendación: A partir de ahora es necesario que a la hora de realizar ejercicios nos acostumbremos a buscar información en la documentación oficial. En caso de no poder resolver las dudas con la documentación ofrecida, el siguiente paso será preguntar las dudas en el foro de clase.

Documentación oficial de JavaScript

Foro de la asignatura

Funciones del DOM

Categorizamos las funciones dependiendo de su uso, desde recoger valores en HTML a modificar el propio HTML.

Búsqueda

Las siguientes funciones nos permiten localizar elementos en el HTML.

- **document.getElementById('id'):** Permite acceder a un elemento mediante su identificador único.
- **document.getElementsByTagName('nombre'):** Permite acceder a todos los elementos que comparten la misma etiqueta. A diferencia del primero, retorna un array de elementos en vez de un único valor.
- **document.getElementsByClassName('clase'):** Permite acceder a todos los elementos que comparten la misma clase. Igual que la anterior, devuelve un array de elementos.
- Hay muchos métodos actualmente para recoger valores en conjuntos concretos, por ejemplo:
    - **document.images** devuelve todas las imágenes del documento.
    - **document.URL** devuelve la URL del documento.
    - **document.body** devuelve el elemento <body> del documento.

- **document.lastModified** devuelve la fecha y hora de la última actualización del documento.

Modificación

Estas funciones son usadas para modificar valores de los elementos HTML que recogimos con los métodos anteriores.

- **'elemento'.innerHTML:** Modifica el código HTML del elemento. Por ejemplo, si fuese un título h1, modificaría su contenido para mostrar el texto que quisieramos.
- **'elemento'.'atributo':** Modica el valor del atributo seleccionado. Por ejemplo, si queremos cambiar el valor del atributo 'href' de un elemento, usaríamos 'elemento'.href = 'nuevo valor'
- **'elemento'.style.'propiedad':** Modifica el valor del atributo o propiedad seleccionada en la hoja de estilos. Por ejemplo. 'Elemento'.style.width modifica el ancho.
- Además, podremos modificar y añadir elementos dinámicamente mediante '**elemento'.setAttribute('atributo','valor')**.

Creación

Este apartado es el más delicado de todos, pues añadir o eliminar elementos en tiempo de ejecución tiene inconvenientes: si todo está estructurado por un layout habrá que tener en cuenta las deformaciones posibles del documento. Así mismo, perdemos el control inmediato sobre los elementos y

tendremos que realizar el diseño previo con sumo cuidado y siendo conscientes de los cambios que pueden ocurrir en nuestro programa.

- **document.createElement('elemento'):** crea un elemento dependiendo de la etiqueta escogida. Por ejemplo document.createElement('H1'). Para ver qué etiqueta corresponde a cada elemento podemos utilizar 'elemento'.tagName.
- **document.removeChild('elemento'):** Elimina el elemento seleccionado. Previamente debemos recoger dicho elemento del HTML.
- **document.appendChild('elemento'):** Añade el elemento seleccionado al documento HTML. Si concatenamos esta función con otro elemento lo añadirá como subetiqueta.
- **document.replaceChild('nuevoEelemento', 'antiguoElemento'):** Reemplaza el segundo elemento con el primero.
- **document.write('html'):** permite escribir directamente sobre el documento HTML. Lo que está entre comillas se interpretará como HTML, por lo que podemos utilizar etiquetas.

Eventos

Este apartado lo veremos más adelante junto con las técnicas para ejecutar código de forma asíncrona, es decir, sin seguir el curso de ejecución normal.

Excepciones

Seguramente a estas alturas, alguna excepción paró la ejecución de nuestro programa, pues algún error ocurría que no permitía que el resultado fuera a ser el esperado. Esto puede ser por muchas razones: variables con valor nulo con las que no se puede operar correctamente, sintaxis incorrecta en ejecución…

Para controlar el flujo del programa, podemos utilizar las sentencias try-catch-finally.

Try - catch

Como la traducción sugiere, indica que el bloque encerrado entre llaves se "probará". Si este código lanza un error, podremos capturarlo con 'catch' e indicar que debe ocurrir:

```
var n1 = null;
try {
        alert(n1.name);
        n1 = 5;
} catch (err) {
        alert("Error del tipo: " + err);
}
alert(n1);
```

En el código vemos que intentamos acceder a la propiedad 'name' de un valor nulo, por lo que la ejecución falla y nos muestra por pantalla el tipo de error del que se trata y no ejecuta el código debajo de "n1.name", pues en 'alert(n1)' el valor sigue siendo 'null'.

Finally

Además de los dos bloques anteriores, el bloque 'finally' permite ejecutar código independientemente de si se lanza un error o no.

```
var n1 = null;
try {
        alert(n1.name);
} catch (err) {
        alert("Error del tipo: " + err);
} finally {
        n1 = 5;
        alert(n1);
}
```

En este caso, a pesar de lanzar un error, el código en el bloque 'finally' se ejecuta igualmente.

Throw

La clausula 'throw' permite generar errores personalizados para controlar el flujo del programa completamente.

```
var n1 = 5;
var n2 = 3;
try {
        if (n1 + n2 > 5) {
            throw "La suma es mayor que 5";
        } else {
            throw "La suma no es mayor que 5";
        }
} catch (err) {
        alert("Error del tipo: " + err);
} finally {
        alert(n1 + n2);
}
```

Como vemos, al ejecutar el código, el error se capturará y nos informará de que "La suma es mayor que 5". Obviamente este tipo de errores es un ejemplo muy simplificado y puede parecerse a la estructura condicional pero cuando trabajamos con valores desconocidos o que es posible que sean nulos, es mejor manejarlos y evitar que la ejecución se interrumpa.