MEDAC
Instituto Oficial de Formación Profesional

DESARROLLO DE WEB ENTORNO CLIENTE
TÉCNICO EN DESARROLLO DE APLICACIONES WEB

Eventos, expresiones y cookies

09

ÍNDICE

OBJETIVOS

Comprender el modelo de eventos de HTML.

Saber gestionar eventos de JavaScript en relación a su tipo.

Conocer el funcionamiento las cookies y los datos que almacenan.

Saber definir expresiones regulares y utilizarlas en HTML.

Conocer el funcionamiento de las expresiones regulares en JavaScript.

/ 1. Introducción y contextualización práctica

Para poder interactuar con el usuario en un documento web, debemos conocer cómo introducir dicha interacción en una página web. HTML ofrece unos mecanismos, basados en eventos, que también son utilizados por otras tecnologías. En calidad de programadores web, utilizaremos estos eventos para poder interactuar con el usuario o bien para poder desarrollar tareas de control. Como bien sabemos, la comunicación web está basada en el protocolo HTTP, que no es capaz de almacenar el estado de la conexión, por lo que es necesario implementar mecanismos para guardar y conocer el estado de la comunicación. En este sentido, estudiaremos las cookies para poder guardar este estado y facilitar la comunicación con el servidor.

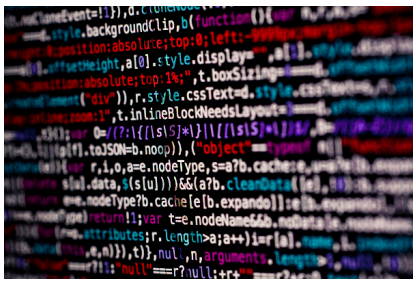

Fig. 1. Los eventos y las cookies requieren de la utilización de JavaScript

Escucha el siguiente audio donde describimos el caso práctico que iremos resolviendo a lo largo de la unidad:

Audio Intro. Eventos y carga del documento HTML

https://bit.ly/3dhGrwC

/ 2. Eventos en HTML

HTML incluye atributos en sus etiquetas, los cuales permiten definir manejadores de eventos. Gracias a estos manejadores, HTML tiene la habilidad de lanzar disparadores cuando se dan determinadas acciones en el documento. Si estamos utilizando JavaScript, podemos utilizar el lenguaje para definir el comportamiento que queramos realizar cuando un evento tenga lugar. Aunque nos refiramos a JavaScript, en la realidad podemos vincular un atributo de evento HTML a otros lenguajes como, por ejemplo, TypeScript.

Los eventos que proporciona HTML van dirigidos a diferentes tipos de objetos. Podemos distinguir eventos a diferentes niveles:

- **Ventana**: se refiere a eventos que pueden tener lugar en una ventana.
- **Formularios**: hace referencia a todos aquellos que ocurren dentro un formulario.
- **Teclado**: es capaz de detectar eventos que se producen al interactuar con el teclado.
- **Ratón**: lanza eventos cuando se detectan ciertos comportamientos en el ratón.
- **Arrastrar**: permite detectar el momento en el que el usuario intenta arrastrar y soltar componentes.
- **Portapapeles**: es capaz de lanzar un evento cuando se utiliza el portapapeles para cortar, copiar o pegar.

Fig. 2. HTML lanza los eventos para que se ejecuten

2.1. Modelo de gestión de eventos en JavaScript

Tal y como hemos comentado anteriormente, JavaScript permite implementar el comportamiento de los eventos que se detectan a través de los disparadores de HTML. Para ello, HTML permite incluir un 'literal' a cada uno de los eventos, que puede ser cualquier código en JavaScript.

Normalmente, el código en JavaScript suele ser muy largo. Por este motivo, si introducimos el código directamente en la etiqueta HTML ni sería legible ni podríamos reutilizar el código implementado.

Por todo ello, lo que se suele utilizar en un atributo de disparador de eventos es el nombre de una función de JavaScript. Al usar una función, solucionamos los dos problemas anteriores. Es decir, ganamos en reutilización de código y también mejoramos su legibilidad. Aunque HTML categoriza los eventos según el elemento que lanza el disparador, en JavaScript el código implementado es independiente de ese elemento. De esta forma, en JavaScript podemos hablar también de manejadores de eventos, que no son más que todas esas funciones que están vinculadas a un disparador HTML. Por ello, el modelo de eventos de JavaScript es reactivo a los diferentes disparadores de HTML en lo que se refiere a la interacción del usuario en el front-end.

Como se puede observar, existe una estrecha relación entre el código HTML y el código que podemos introducir en JavaScript.

Audio 1. Modelo dirigido por eventos
https://bit.ly/2BnQsuW

2.2. Flujo de eventos

Hay muchas ocasiones en las que es importante determinar cuándo ejecutar el código JavaScript, pues es posible que el documento web no se encuentre cargado en su totalidad. Como sabemos, el servidor envía el código HTML que debe interpretar el navegador para generar una visualización de la página web. En este sentido, desde que el navegador recibe el resultado hasta que muestra la representación visual hay una cierta latencia. Si consideramos que el código JavaScript se va a ejecutar en el orden que aparece escrito, podría suceder que se acceda a elementos del DOM que no están generados y producir, de esta manera, un error. Por ello, es importante detectar el momento en el que el documento está listo para poder ejecutar las funciones JavaScript correspondientes. Cuando una página web finaliza su carga, se lanza el evento de finalización de carga onload. Una vez se detecta que el documento se ha cargado, se pasa a una nueva fase en la que el flujo de control dependerá de la interacción con el usuario. En esta fase, el usuario tomará el control del documento web, pues a través de las acciones que realice y de la detección de los eventos, se determinarán las tareas que la web ejecutará.

Como la estructura del documento web es arbórea, es imprescindible esperar a que se genere todo el árbol de la web para que el funcionamiento sea el esperado. A veces, algunos scripts se ponen al principio del documento web. Sin embargo, es posible que dichos scripts ejecuten el código directamente, lo cual provocaría una ejecución anómala (o simplemente que no realice nada). Además, cuando los scripts se incluyen al comienzo de la web, se introduce una latencia en la carga del documento. Por ello, hoy en día existe una práctica que consiste en incluir los documentos de script al final del documento web para, así, aligerar la carga y evitar comportamientos anómalos.

Fig. 3. Incluir la carga JavaScript al comienzo del documento puede generar errores

/ 3. Caso práctico 1: "Control de formulario"

Planteamiento: Como responsable en un equipo de desarrollo web, nuestro cliente nos pide que tenemos que añadir unas medidas de control para asegurarnos de que los campos de un formulario solo contengan valores numéricos para que un usuario no pueda incluir otro tipo de datos en dichos campos. Aunque no es lo habitual, el cliente (que cree que controla mucho de programación) nos dice que técnicamente no se pueden realizar los controles en el documento HTML.

Nudo: ¿Cómo lo harías? ¿Es posible? ¿Crees que el requisito del cliente tiene sentido?

Desenlace: Como hemos visto en el apartado de expresiones regulares, es posible introducir medidas de control en el código JavaScript para reconocer patrones. Se pueden realizar estos controles a través de HTML5. Sin embargo, el cliente no quiere que utilicemos esta alternativa. Realmente, es un requisito que puede influir negativamente en el desarrollo y en el propio funcionamiento de los cambios. Debemos tener en cuenta que JavaScript puede estar desactivado y, en consecuencia, no realizaría las comprobaciones. Para que esta funcionalidad se ejecute correctamente, debemos asegurarnos previamente de que el navegador no está bloqueando la ejecución de JavaScript. En caso de detectar un bloqueo, deberíamos indicar al usuario que lo active para permitir la ejecución de la web. Otra alternativa podría ser implementar una web «ligera» que no utilice JavaScript y, de esta forma, redirigir al usuario a esta web cuando se detecte que el navegador está bloqueando los scripts.

Fig. 4. El bloqueo de JavaScript puede suponer un problema en la ejecución de nuestra web

/ 4. Expresiones regulares

Una expresión regular permite definir patrones que se utilizan con el objetivo de encontrar una determinada combinación de caracteres dentro de una cadena de texto. Se trata de una solución utilizada no solo en el mundo web, sino también en cualquier contexto informático, por ejemplo, en la creación de lenguajes de programación.

Cuando nos centramos en el mundo web, las expresiones regulares se pueden introducir tanto a nivel de HTML como a nivel de JavaScript.

Vídeo 1. Eventos en HTML-JavaScript
https://bit.ly/37FYHyP

4.1. Expresiones regulares en JavaScript

En JavaScript las expresiones regulares se tratan como objetos, y permiten ejecutar diferentes métodos para determinar si existe o no el patrón en la cadena de texto que se proporciona. La expresión regular se puede crear mediante su representación textual o a través de una cadena.

```
var er1 = /ab+c/;
//Expresión regular creada a través del constructor RegExp
var er2 = new RegExp('ab+c');
```

Código 1. Expresión regular creada a partir de una cadena literal, utilizando "/"

Una expresión regular puede categorizarse dependiendo de los elementos que se incluyan en su definición. Por ejemplo, podemos indicar una combinación de caracteres simple, o bien utilizar caracteres especiales para señalar algún tipo de repetición o comportamiento especial en la definición del patrón. A la hora de definir expresiones regulares, existen caracteres especiales que determinan el significado del patrón que estamos especificando. De esta forma, se pueden crear patrones que, por ejemplo, tengan en cuenta cadenas que no comiencen con un carácter o que finalicen con un carácter determinado. También es posible crear expresiones regulares que permitan la repetición de ciertos caracteres un número determinado de veces. También es posible indicar que un carácter sea opcional.

4.2. Expresiones regulares: caracteres especiales

A continuación, mostramos una tabla con los principales caracteres especiales para definir una expresión regular en JavaScript:

A través de los caracteres especiales de las expresiones regulares se aumenta su potencia y versatilidad. Mediante ellas realmente se pueden establecer patrones. Igualmente, aumenta significativamente la facilidad de detectarlos.

Carácter	Descripción
*	Se utiliza para indicar la repetición de caracteres desde 0 a n veces.
+	Se utiliza para indicar la repetición de caracteres desde 1 a n veces.
	Permite indicar que un carácter es opcional.
.	Permite indicar cualquier carácter.
\	Se utiliza como carácter para introducir el reconocimiento de otros caracteres especiales.
^	Se utiliza para determinar el comienzo de una expresión.

Tabla 1. Caracteres para las expresiones regulares

/ 5. Cookies

Una cookie permite almacenar una pequeña cantidad de datos en un entorno web, pues el navegador puede asociar estos datos con el sitio. En su diseño inicial, se utilizaban en el lado del servidor y a través de una extensión del protocolo HTTP. Una cookie se transmite automáticamente entre el navegador y el servidor, lo cual habilita a este último para leer los datos almacenados en el cliente.

5.1. Consideraciones sobre las cookies

El término cookie se ha utilizado en la informática, tradicionalmente, para referirse a pequeñas porciones de datos que pueden almacenar información sensible, como contraseñas, por ejemplo. En el contexto de JavaScript, una cookie es capaz de guardar el estado de la conexión entre el cliente y el servidor. Por ello, no hay que considerar que las cookies en JavaScript guarden información criptográfica.

El API que proporciona JavaScript para la manipulación de cookies puede resultar bastante antiguo si lo comparamos con el resto del lenguaje. Para la manipulación de cookies no se usa ningún método para consultar, almacenar o borrar datos, sino que se utiliza el DOM con el objetivo de que podamos consultar una propiedad del documento. Una cookie puede tener asociado un tiempo de vida que se especifica de manera independiente para cada una.

5.2. Lectura de cookies

Como hemos visto anteriormente, para leer una cookie en JavaScript hay que acceder a una propiedad DOM (que se llama cookie). El valor de la cookie se almacena utilizando pares de valores en el formato nombre=valor, separado por ";". Para poder manejar correctamente la lista de valores de una cookie, es recomendable utilizar el método split() proporcionado por JavaScript para transformar la cadena en parejeras de valores. Una vez tenemos el valor de la cookie, podría ser necesario realizar una decodificación atendiendo a su juego de caracteres y al formato en el que esté representada.

Fig. 5. Las cookies ofrecen información de tendencias

5.3. Ejemplo de lectura de una cookie

Mostramos a continuación un ejemplo de lectura de una cookie:

```
function getCookies() {
  let cookies = new Map();   //Creamos una estructura map
  let all = document.cookie;  //Leemos toda la cookie a través del DOM
  let list = all.split("; "); //Dividimos la cookie utilizando el separador "; "
  for(let cookie of list) {
    if (!cookie.includes("=")) continue; //Saltamos los campos que no sean valores
    //Procesamos las clave-valor
    let p = cookie.indexOf("=");
    let name = cookie.substring(0, p);
    let value = cookie.substring(p+1);
    //Decoficamos el valor de la cookie atendiendo a la codificación
    value = decodeURIComponent(value);
    //Almacenamos la cookie en la estructura
    cookies.set(name, value);
  }
  return cookies;
}
```

Código 2. Lectura de una cookie

5.4. Almacenamiento de cookies

Para introducir un valor en una cookie simplemente hay que añadir un valor en formato nombre=valor a la propiedad document.cookie. De esta forma, la próxima vez que se lea la propiedad de la cookie, el valor que hemos añadido aparecerá en la lista. Es importante que guardemos el valor mediante la codificación en el formato correspondiente utilizando el método encodeURIComponent(). Siguiendo esta filosofía, la duración de la cookie se fija al tiempo de vida de la sesión, es decir, se pierde cuando el navegador se cierra. Si queremos crear una cookie permanente, tendremos que especificar el tiempo de vida en segundos mediante el atributo max-age.

Vídeo 2. Depurando cookies en el navegador
https://bit.ly/2Nbbwrl

5.5. Ejemplo de almacenamiento de cookies

Para poder realizar el almacenamiento de las cookies es necesario que el navegador permita dicha tarea. En muchas ocasiones su almacenamiento está relacionado con la vulnerabilidad y seguridad del navegador.

Veamos en este caso un ejemplo de almacenamiento de cookies:

```
function setCookie(name, value, timeLife=null) {
   let cookie = `${name}=${encodeURIComponent(value)}`;
   if (timeLife !== null) {
      cookie += `; max-age=${timeLife*60*60*24}`;
   }
   document.cookie = cookie;
}
```

Código 3. Almacenamiento de cookies

5.6. Limitaciones de las cookies

Las cookies se crearon para almacenar pequeñas cantidades de datos por parte del servidor y permitir, después, esa transferencia de datos cada vez que se solicitara una URL determinada. El estándar que define el funcionamiento de las cookies en los navegadores permite un número indeterminado de cookies de cualquier tamaño, pero se recomienda no almacenar más de 300 cookies en total, 20 por sitio web, y no superar los 4 KB de tamaño por cada cookie. Sin embargo, en la actualidad estos parámetros no se cumplen.

Fig. 6. El almacenamiento de cookies puede ser masivo

/ 6. Caso práctico 2: "Tokens"

Planteamiento: Nuestro cliente más importante nos pide crear un sitio web para comercio electrónico. Una de las principales características es que no se quiere hacer responsable de almacenar las credenciales de sus potenciales clientes en el servidor, sino que quiere ofrecer un mecanismo de tokens.

A través de la utilización de tokens es posible securizar el acceso a determinados aspectos de la web/servicio. Cuando realizamos un login en un portal web, el servidor nos incluye el valor de validación o token en la cabecera del protocolo HTTP. Normalmente, este token tiene un tiempo de vida, y puede ir vinculado a otro tipo de parámetros como fecha o a valores almacenados en la máquina a través de una cookie. Nuestro cliente nos pide extender este mecanismo, pues quiere que el servidor devuelva un token a cada usuario que será almacenado en una cookie.

Nudo: ¿Crees que es una buena solución? ¿Es segura?

Desenlace: Como hemos visto en el apartado sobre las limitaciones de las cookies, no es recomendable guardar información sensible en las cookies, pues no se almacena de forma segura. En este caso, el cliente nos pide almacenar un token en una cookie. Aunque no se trate de información sensible, es cierto que la información que contiene sí permitiría realizar tareas críticas. Por lo tanto, no es una buena opción. Desde el punto de vista de la seguridad, no es nada aconsejable utilizar este mecanismo de almacenamiento. Por ejemplo, si alguien nos robara la cookie con la información del login, podría iniciar sesión sin problemas.

Fig. 7. Los tokens permiten realizar la validación de usuarios

/ 7. Resumen y resolución del caso práctico de la unidad

A lo largo de este tema hemos presentado las principales características de la gestión de eventos en HTML y JavaScript. Hemos detallado cómo implementar eventos en cualquier etiqueta en HTML y cómo deberíamos organizar correctamente estos eventos en el lenguaje para mantener el código lo más limpio posible.

Hemos estudiado también el proceso de carga de un documento HTML para poder introducir la gestión de eventos y descubrir el modelo de eventos que se ofrece en el documento web. Como hemos podido observar, HTML puede categorizarlos dependiendo del elemento que lo genere. Sin embargo, JavaScript no tiene en cuenta el elemento que ha lanzado el evento.

En este tema también se han estudiado las diferentes alternativas para la definición de expresiones regulares, que pueden servir para la validación de formularios en un documento web y evitar que el servidor realice estas tareas. Por último, hemos visto cómo se debe realizar el almacenamiento de datos de conexión a través de las cookies. Como hemos comprobado, la elección de las cookies depende del tipo de almacenamiento que queramos, persistente o temporal.

Resolución del caso práctico de la unidad

Muchas veces cuando intentamos acceder a un elemento del documento web, este no está disponible, pues no ha finalizado la carga de la página visualizada. Esto sucede porque estamos intentado acceder al documento antes de que finalice su carga. Un solución posible para corregir esto sería incluir el código JavaScript al final del documento web, ya que la carga se realiza de arriba abajo. Otra alternativa sería utilizar el evento onload en el cuerpo del documento (body) para poder lanzar el código correspondiente.

/ 8. Bibliografía

Frisbie, M. (2020). *Professional JavaScript® for web developers*. Indianapolis: Wrox, a Wiley brand.

Camden, R. (2016). *Client-side data storage: keeping it local*. Sebastopol: O'Reilly Media, Inc.