DWC

(Desarrollo Web en entorno cliente)

Índice

1.- Añadiendo Scripts al HML

Los programas de JavaScript se pueden insertar en cualquier parte de un documento HTML con la ayuda de la etiqueta <script>.

Por ejemplo:

```
<!DOCTYPE HTML>
<html>
<body>
  <p>Before the script...</p>
  <script>
    alert( 'Hello, world!' );
  </script>
  <p>...After the script.</p>
</body>
</html>
```

La etiqueta <script> contiene código JavaScript que se ejecuta automáticamente cuando el navegador procesa la etiqueta.

2.- Etiquetado moderno

La etiqueta <script> tiene algunos atributos que rara vez se usan hoy en día, pero aún se pueden encontrar en el código antiguo:

El atributo type:<script type=…>

El antiguo estándar HTML, HTML4, requería tener un type. Por lo general era type="text/javascript". Ya no se requiere. Además, el estándar HTML moderno cambió totalmente el significado de este atributo. Ahora, se puede usar para módulos JavaScript.

El atributo language:<script language=…>

Este atributo estaba destinado a mostrar el idioma del script. Este atributo ya no tiene sentido porque JavaScript es el idioma predeterminado. No hay necesidad de usarlo.

Comentarios antes y después de los guiones.

En código más antiguo, se puede encontrar comentarios dentro de las <script>etiquetas, como este:

```
<script type="text/javascript">
<!--    ...  -->
</script>
```

Este truco no se usa en JavaScript moderno. Estos comentarios ocultan el código JavaScript de los navegadores antiguos que no sabían cómo procesar la etiqueta <script>. Dado que los navegadores lanzados en los últimos 15 años no tienen este problema, este tipo de comentario puede ayudarlo a identificar un código realmente antiguo.

3.- Scripts externos

Si tenemos mucho código JavaScript, podemos ponerlo en un archivo separado. Los archivos de script se adjuntan a HTML con el atributo src:

```
<script src="/path/to/script.js"></script>
```

Aquí, /path/to/script.js es una ruta absoluta al script desde la raíz del sitio. También se puede proporcionar una ruta relativa desde la página actual. Por ejemplo, src="script.js"significaría un archivo "script.js"en la carpeta actual.

También podemos dar una URL completa. Por ejemplo:

```
<script src="https://cdnjs.cloudflare.com/ajax/ajax.js"></script>
```

Para adjuntar varios scripts, usamos varias etiquetas:

```
<script src="/js/script1.js"></script>
<script src="/js/script2.js"></script>
```

A tener en cuenta:

- Como regla general, solo los scripts más simples se colocan en HTML. Los más complejos residen en archivos separados.
- El beneficio de un archivo separado es que el navegador lo descargará y lo almacenará en su caché .
- Otras páginas que hacen referencia al mismo script lo tomarán del caché en lugar de descargarlo, por lo que el archivo se descarga solo una vez.

- Eso reduce el tráfico y hace que las páginas sean más rápidas.

Si `src` se establece, el contenido del script se ignora. Una etiqueta `<script>` no puede tener el atributo `src` y el código dentro.
Esto no funcionará

```
<script src="file.js">
  alert(1); // the content is ignored, because src is set
</script>
```

Debemos elegir un código externo `<script src="…">` o en el mismo html `<script>` con código.
El ejemplo anterior se puede dividir en dos scripts para trabajar:

```
<script src="file.js"></script>
<script>
  alert(1);
</script>
```

4.- Estructura del código

Lo primero que estudiaremos son los bloques de construcción de código.

4.1.- Declaraciones

Las declaraciones son construcciones de sintaxis y comandos que realizan acciones.

Ya hemos visto una declaración `alert('Hello, world!')` que muestra el mensaje "¡Hola, mundo!".

Podemos tener tantas declaraciones en nuestro código como queramos. Las declaraciones se pueden separar con un punto y coma.

Por ejemplo, aquí dividimos "Hello World" en dos alertas:

```
alert('Hello'); alert('World');
```

Por lo general, las declaraciones se escriben en líneas separadas para que el código sea más legible:

```
alert('Hello');
alert('World');
```

4.2.- Punto y coma

Se puede omitir un punto y coma en la mayoría de los casos cuando existe un salto de línea.

Esto también funcionaría:

```
alert('Hello')
alert('World')
```

Aquí, JavaScript interpreta el salto de línea como un punto y coma "implícito". Esto se llama **inserción automática de punto y coma**.

En la mayoría de los casos, una nueva línea implica un punto y coma. ¡Pero "en la mayoría de los casos" no significa "siempre"!

Hay casos en que una nueva línea no significa un punto y coma. Por ejemplo:

```
alert(3 +
1
+ 2);
```

El código `6` sale porque JavaScript no inserta punto y coma aquí. Es intuitivamente obvio que si la línea termina con un signo más "+", entonces es una "expresión incompleta", por lo que no es necesario el punto y coma. Y en este caso eso funciona según lo previsto.

Pero hay situaciones en las que JavaScript "no puede" asumir un punto y coma donde realmente se necesita.

Los errores que ocurren en tales casos son bastante difíciles de encontrar y corregir.

Un ejemplo de error.

Si tenéis curiosidad por ver un ejemplo concreto de tal error, mirad este código:

```
[1, 2].forEach(alert)
```

No es necesario pensar en el significado de los corchetes `[]` y `forEach` aún. Los veremos más tarde. Por ahora, solo recordad el resultado del código: se muestra `1` y luego `2`.

Ahora, agreguemos un `alert` antes del código y *no lo* terminemos con un punto y coma:

```
alert("There will be an error")
[1, 2].forEach(alert)
```

Ahora, si ejecutamos el código, solo `alert` se muestra el primero y luego tenemos un error.

Pero todo vuelve a estar bien si agregamos un punto y coma después de `alert`:

```
alert("All fine now");
[1, 2].forEach(alert)
```

Ahora tenemos el mensaje "Todo bien ahora" seguido de `1` y `2`.

El error en la variante sin punto y coma ocurre porque JavaScript no asume un punto y coma antes de los corchetes [...].

Entonces, debido a que el punto y coma no se inserta automáticamente, el código en el primer ejemplo se trata como una sola declaración. Así es como el motor lo ve:

```
alert("There will be an error")[1, 2].forEach(alert)
```

Pero deberían ser dos declaraciones separadas, no una. Tal fusión en este caso es simplemente incorrecta, de ahí el error. Esto puede suceder en otras situaciones.

Se recomienda poner punto y coma entre las declaraciones, incluso si están separadas por líneas nuevas. Esta regla es ampliamente adoptada por la comunidad. Tengamos en cuenta una vez más: *es posible* dejar de punto y coma la mayor parte del tiempo. Pero es más seguro, especialmente para un principiante, usarlos.

4.3.- Comentarios

A medida que pasa el tiempo, los programas se vuelven cada vez más complejos. Se hace necesario agregar *comentarios* que describan lo que hace el código y por qué.

Los comentarios se pueden poner en cualquier lugar de un script. No afectan su ejecución porque el motor simplemente los ignora.

Los comentarios de una línea comienzan con dos caracteres de barra diagonal `//`.

El resto de la línea es un comentario. Puede ocupar una línea completa propia o seguir una declaración.

Como aquí:

```
// This comment occupies a line of its own
alert('Hello');
alert('World'); // This comment follows the statement
```

Los comentarios de varias líneas comienzan con una barra diagonal y un asterisco `/*` y terminan con un asterisco y una barra diagonal `*/`.

```
/* An example with two messages.
This is a multiline comment.
*/
alert('Hello');
alert('World');
```

El contenido de los comentarios se ignora, por lo que si ponemos código dentro `/* … */`, no se ejecutará.

A veces puede ser útil deshabilitar temporalmente una parte del código:

```
/* Commenting out the code
alert('Hello');
*/
alert('World');
```

¡Los comentarios anidados no son compatibles!

Puede que no haya `/*...*/`dentro de otro `/*...*/`.

Este código mostrará con un error:

```
/*
  /* nested comment ?!? */
*/
alert( 'World' );
```

No dudéis en comentar el código.

Los comentarios aumentan la huella general del código, pero eso no es un problema en absoluto. Hay muchas herramientas que minimizan el código antes de publicar en un servidor de producción. Quitan los comentarios, por lo que no aparecen en los scripts de trabajo. Por lo tanto, los comentarios no tienen ningún efecto negativo en la producción.

4.4.- "Use estrict"

Durante mucho tiempo, JavaScript evolucionó sin problemas de compatibilidad. Se agregaron nuevas funciones al lenguaje mientras que la funcionalidad anterior no cambió. Eso tenía el beneficio de nunca romper el código existente. Pero la desventaja era que cualquier error o una decisión imperfecta hecha por los creadores de JavaScript se atascó en el lenguaje para siempre.

Este fue el caso hasta 2009 cuando apareció ECMAScript 5 (ES5). Agregó nuevas características al lenguaje y modificó algunas de las existentes. Para mantener el código antiguo funcionando, la mayoría de las modificaciones están desactivadas de manera predeterminada. Es necesario que les permita explícitamente de una directiva especial: `"use strict"`.

La directiva es una cadena: `"use strict"` o `'use strict'`. Cuando se encuentra en la parte superior de un script, todo el script funciona de la manera "moderna".

Por ejemplo:

```
"use strict";
// this code works the modern way
...
```

Muy pronto vamos a aprender funciones (una forma de agrupar comandos), así que tengamos en cuenta de antemano que `"use strict"` se puede poner al comienzo de una función. Hacer eso habilita el modo estricto solo en esa función. Pero generalmente la gente lo usa para todo el script.

Comprobad que "use estrict" esté en la parte superior

Comprobad que `"use strict"` esté en la parte superior de sus scripts; de lo contrario, es posible que el modo estricto no esté habilitado.

El modo estricto no está habilitado aquí:

```
alert("some code");
// "use strict" below is ignored--it must be at the top

"use strict";
// strict mode is not activated
```

Solo los comentarios pueden aparecer arriba `"use strict"`.

No hay forma de cancelar `use strict`

No existe una directiva como `"no use strict"` que revierta el motor al comportamiento anterior.

Una vez que ingresamos al modo estricto, no hay vuelta atrás.

¿Deberíamos utilizar "use estrict"?

La pregunta puede sonar obvia, pero no es así.

Uno podría recomendar comenzar los scripts con `"use strict"`... pero JavaScript moderno soporta "clases" y "módulos" - estructuras de lenguaje avanzadas, que se habilitan `use strict` automáticamente. Por lo tanto, no necesitamos agregar la `"use strict"`directiva, si las usamos.

Entonces, por ahora `"use strict"` es un invitado bienvenido en la parte superior de los scripts. Más tarde, cuando el código esté todo en clases y módulos, podremos omitirlo.