JAVASCRIPT

Lección 7: ES6, strict mode, setTimeout

eip ESCUELA INTERNACIONAL POSGRADOS

ÍNDICE

ES6, strict mode, setTimeout

PRESENTACIÓN Y OBJETIVOS

En esta unidad se estudiarán más en detalle las versiones de ECMAScript y en particular las novedades de ES6.

Objetivos

En esta unidad aprenderás:

- ✓ **Cuáles son las versiones de ECMAScript**
- ✓ **Qué funcionalidad nueva aparecen en ES6**
- ✓ **La diferencia entre JS en strict mode y JS en sloppy mode**
- ✓ **Qué hacen las funciones setTimeout / setInterval**
- ✓ **Qué es el operador ternario**

1 ECMAScript

Como lo sabemos desde la primera clase, JS es el lenguaje y ECMAScript es el estándar.

Unas de las cosas que ECMAScript define:

- La sintaxis del lenguaje (palabras claves, flujos de control, etc.)
- Mecanismos para el control de errores
- Tipos (object, number, function, string, etc.)
- Funciones y objetos incorporadas (Math, métodos de arreglos, etc.)
- strict mode

Ediciones ECMAScript:

- ES1 – 1997
  - Aparece el primer estándar oficinal para JS
- ES2 – 1998
  - Cambios editoriales
- ES3 – 1999
  - Añade: try / catch, switch, do while
- ES4
  - Nunca llego a ser publicado
- ES5 – 2009
  - La primera mayor revisión
  - Incluye: strict mode, soporte para JSON, métodos de arreglos
- ES6 – 2015
  - El segundo release mayor
  - Incluye: arrow functions, const, let, promesas, clases, symbol

Desde la edición ES6, cada año se publica una nueva edición, que lleva en el nombre el año (ECMAScript 2016, ECMAScript 2017, ..., ECMAScript 2022, ESNext). Se prefiere esta forma para ir añadiendo menor funcionalidad cada año, que publicar grandes cambios cada mucho tiempo – como paso con ES6.

ESNext es un nombre dinámico que se usa cuando nos referimos a las versiones futuras de ECMAScript.

Una cosa muy importante de esas ediciones es que son backwards compatible. Eso quiere decir que si, en el momento actual, escribes código siguiendo las reglas de ES1, ese código aún funcionara. El motor de JS de hoy es capaz de entender código JS viejo.

Todo esto es parte de la solidez de JS y la prevención sobre la ruptura de la web en general. Se impide añadir breaking changes o borrar funcionalidad vieja.

Hay que recordar que ECMAScript es un estándar. Eso quiere decir que la manera en la cual los motores y compiladores implementan ese estándar es una historia completamente diferente. Por eso, hay casos donde alguna funcionalidad de ES2017 puede funcionar en un navegador, pero no en otro.

ES5 está implementado en todos los navegadores. A partir de esta, los siguientes estándares tienen implementación en los navegadores más modernos.

Una manera de verificar con qué navegadores esta compatible una funcionalidad es buscarla en MDN. Abajo de todo hay una sección llamada Browser Compatibility.

¿Por qué nos interesa la compatibilidad? Pues porque, aunque nosotros estemos desarrollando y probando en la última versión de Chrome, no todos los que van a acceder a nuestro website (una vez subido a producción) van a usar el mismo navegador que nosotros.

Eso no quiere decir que no podemos usar sintaxis del nuevo estándar. Para este tipo de casos, necesitamos transpilar (transformar) nuestro código JS a ES5. Para hacer eso, se puede usar una herramienta conocida como Babel.

El motto de Babel es: ¡Use next generation JavaScript, today!

1.1 ES6

const

let

arrow functions

default parameter values

- permiten que los parámetros de una función tengan un valor por defecto

```
function greetings(message = 'Good morning') {
   console.log(message)
}

greetings() // Good morning
greetings('Good afternoon ') // Good afternoon
```

rest parameter

- permiten a una función recibir un numero variable de argumentos almacenados en un arreglo

```
function restParam(a, b, ...restOfParams) {
   console.log(a)
   console.log(b)
   console.log(restOfParams)
}

restParam(1, 2, 3, 4, 5)
// 1
// 2
// [3,4,5]
```

spread operator

- permite que un iterable sea expandido en 0 o más argumentos

```
const numbersToAdd = [10, 20, 4]

function sum (a, b, c) {
   console.log('Sum is', a + b + c)
}

sum(...numbersToAdd) // 34
```

- template literals (string interpolation)

```
const person = {name: 'Roxana', age: 30}
console.log(`My name is ${person.name} and I am ${person.age} years old`)
//"My name is Roxana and I am 30 years old"
```

- computed property names

```
let obj = {
 ['roxana'.toUpperCase()]: 'Cestari'
}
console.log(obj) //  { ROXANA: 'Cestari' }
```

- property shorthand
- destructuring
- class

2 STRICT MODE

Es un modo especial que nos permite escribir JS de una forma más segura. Fue introducido en ES5.

Elimina algunos de los errores silenciosos de JS y lanza errores explícitos.

Se puede activar añadiendo 'use strict ' al principio de tu script. Es necesario que sea la prima ‘instrucción’ en tu fichero. Se pueden añadir comentarios antes, porque JS los ignora. Pero si se añade código antes de escribir 'use strict ', el modo estricto no se activará.

Esta expresión ('use strict') fue diseñada para ser compatible con versiones más antiguas de JS. Simplemente se ignora.

El modo estricto se puede también activar sólo para una función o para un bloque especifico.

```
function hello() {
  'use strict';
  // codigo
}
```

Esta forma se puede usar cuando trabajas con legacy code, donde si pasas todo en modo estricto, el código se puede romper.

Modo estricto:

> Previene la creación accidental de variables globales

```
'use strict';

 let hasDrivingLicense
 hasDrivinLicense = true // Reference error, hasDrivinLicense is not defined
```

En sloppy mode, hasDrivinLicense = 10 (nótese que falta la “g” en “driving”) hubiese simplemente creado otra variable global.

> No te deja usar palabras claves reservadas para el futuro (implements, interface, let, package, private, protected, public, static, yield), como nombre de variable

```
'use strict';

 let interface = true // SyntaxError: Unexpected strict mode reserved word
```

3 SETTIMEOUT / SETINTERVAL

setTimeout es un método del objeto window (Window es un objeto que representa una ventana abierta en un navegador), que llama a una función después de un numero x de milisegundos.

La función setTimeout devuelve el id del temporizador. Ese id se puede pasar como parámetro para la función clearTimeout(). **clearTimeout** borrara el temporizador.

Sintaxis setTimeout

setTimeout (función, milisegundos)

Ejemplo

```
setTimeout(function() {
    console.log( '3 seconds have passed ')
}, 3000)
```

Usando arrow functions

```
setTimeout(() => console.log('3 seconds have passed'), 3000)
```

// en los 2 ejemplos, después de 3 segundos, el mensaje ***3 seconds have passed*** se muestrera en la consola.

Sintaxis clearTimeout

clearTimeout (temporizador)

Ejemplo

```
const myTimeout = setTimeout(() => console.log('3 seconds have passed'), 3000)
clearTimeout(myTimeout)
```

// 3 seconds have passed ya no se va a mostrar mas

La función dentro del setTimeout se ejecuta 1 sola vez (a menos de que se borre el temporizador). Para ejecuciones repetidas de la función, hay que usar setInterval().

setInterval es un metodo del objeto Window que llama a una función cada x milisegundos, hasta que la ventana cierra o invoque a **clearInterval**().

clearInterval() detiene al temporizador establecido con setInterval().

Sintaxis setInterval

setInterval (función, milisegundos)

Ejemplo

```
setInterval(function() {
    console.log( 'Hello every 3 seconds! ')
}, 3000)
```

Usando arrow functions

```
setIntervalt(() => console.log('Hello every 3 seconds'), 3000)
```

// en los 2 ejemplos, cada 3 segundos, el mensaje ***Hello every 3 seconds*** se muestrera en la consola.

Sintaxis clearInterval

clearInterval (temporizador)

Ejemplo

```
const myInterval = setInterval(() => console.log('Hello'), 3000)
clearInterval(myInterval)
```

// Hello ya no se va a mostrar mas

4 SINTAXIS PARA RESUMIR CODIGO (SHORTHAND CODING TECHNIQUES)

4.1 El operador condicional (ternario)

Sintaxis operador ternario

condición ? exprTrue : exprFalse

Ejemplo

```
let age = 30
let drink = age >= 21 ? 'Wine' : 'Coca Cola'

console.log(drink) // Wine
```

- condición – la condición que se va a evaluar
- exprTrue – la expresión que se va a ejecutar si condición se evalúa a true
- exprFalse – la expresión que se va a ejecutar si condición se evalúa a false

Esta es la versión corta de un if else. De una manera similar con if ... else if ... else, el operador ternario también puede ser encadenado.

4.2 Declaración de múltiples variables

```
let a, b, c = 5
```

en vez de

```
let a;
let b;
let c = 5
```

4.3 If checks

```
if(hasKids)
if(!hasKids)
```

en vez de

```
if(hasKids === true)
if(hasKids !== true)
```

5 PUNTOS CLAVE

- ES6 fue la primera revisión grande del estándar ECMAScript
- ES6 trae bastante funcionalidad nueva como: let, const, arrow functions, rest parameter, spreading operator, string interpolation
- setTimeout() nos permite llamar a una función después de unos milisegundos
- Para cancelar la llamada de esa función, usamos el método clearTimeout()
- setInterval() nos permite llamar a una función cada x milisegundos
- Ese temporizador se cancela sólo con cerrar la ventana o con llamar a clearInterval()
- El operador ternario es una forma más sencilla de escribir un if ... else