Curso HTML, CSS y JavaScript

Junio de 2017

Autor: Fran Linde Blázquez

TEMA 10:
ECMASCRIPT 6

ÍNDICE

1. INTRODUCCIÓN

ECMAScript v6 (Abreviado como ES6 o ES2015) es el estándar que seguirá JavaScript a partir de junio de este año (2015). Hasta el momento la versión de JS que estamos usando en nuestros navegadores y Node.js, es la v5.

En junio de 2013 quedó parado el borrador de la versión 6, pero en diciembre de 2014 se aprobó al fin y se espera su estandarización a partir de Junio de 2015.

Veamos las principales novedades de ES6:

2. LET Y CONST

Ahora podemos declarar variables con let en lugar de var si no queremos que sean accesibles más allá de un ámbito.

Por ejemplo:

```
//ES5
(function() {
    console.log(x); // x no está definida aún.
    if (true) {
        var x = "hola mundo";
    }
    console.log(x);
    // Imprime "hola mundo", porque "var" hace que sea global
    // a la función;
})();

//ES6
(function() {
    if (true) {
        let x = "hola mundo";
    }
    console.log(x);
    //Da error, porque "x" ha sido definida dentro del "if"
})();
```

Ahora con const podemos crear constantes que sólo se puedan leer y no modificar a lo largo del código. Veamos un ejemplo

```
(function() {
    const PI;
    PI = 3.15;
    // ERROR, porque ha de asignarse un valor en la
    // declaración
})();

(function() {
    const PI = 3.15;
    PI = 3.14159;
    // ERROR de nuevo, porque es sólo-lectura
})();
```

3. CLASES EN ES6

Con la llegada de la nueva versión del estándar de JavaScript (ECMAScript 6 o ECMAScript 2015) la definición de una función como clase ha cambiado. ES6 aporta lo que es conocido como azúcar sintáctico, para declarar una clase como en la mayoría de los lenguajes de programación orientados a objetos. Aun así, internamente sigue siendo una función que hace uso de Prototype.

3.1 Definiendo clases en ES6

Veamos cómo definir clases mediante **"class",** para ello haremos uso del ejemplo de la clase animal que hemos usado anteriormente:

```
class Animal{
    constructor(nombre, sonido){
        this._nombre = nombre;
        this._sonido = sonido;
    }

    emitirSonido(){
        console.log("El " + this._nombre + " hace " + this._sonido);
    }
}

var miPerro = new Animal("Perro", "Guau!");
var miGato = new Animal("Gato", "Miau!");

miPerro.emitirSonido();
miGato.emitirSonido();
```

Si imprimimos por consola las variables **"miPerro"** y **"miGato"** podremos observar cómo a pesar de haber creado esta clase mediante **class,** internamente las instancias generadas son idénticas.

También podemos observar cómo ha cambiado la manera de definir la clase, pero no cambia la manera de crear nuevas instancias o hacer uso de estas.

3.2 Herencia en clases de ES6 (extends y super)

Gracias a esta nueva sintaxis podemos realizar herencia de una manera más sencilla, ya que disponemos de **extends** para indicar que una clase hereda de otra. También podemos hacer uso de **super** para llamar al constructor de la clase heredada.

Veamos un ejemplo con vehículos. Empezamos definiendo la clase padre:

```
class Vehiculo {
    constructor(tipo, nombre, ruedas) {
        this.tipo = tipo;
        this.nombre = nombre;
        this.ruedas = ruedas
    }
    getRuedas() {
        return this.ruedas
    }
    arrancar() {
        console.log("Arrancando el " + this.nombre});
    }
    aparcar() {
        console.log("Aparcando el " + this.nombre});
    }
}
```

Si quisiéramos heredar de la clase **Vehículo** para hacer uso de sus métodos deberemos usar **extends** para heredarla y **super** para invocar a su constructor:

```
class Coche extends Vehiculo {
    constructor(nombre) {
        super('coche', nombre, 4)
    }
}
```

De esta manera podremos hacer uso de los métodos del padre:

```
var beetle = new Coche('Volkswagen Beetle')
beetle.getRuedas() // 4
beetle.arrancar() // Arrancando el beetle
```

4. ARROW FUNCTIONS

4.1 Usando arrow functions

Las arrow functions aparecen para evitar que sigamos escribiendo código como este:

```
// ES5
// Imaginemos una variable data que incluye un array de objetos
var data = [{... }, {... }, {... }, ...];
data.forEach(function(elem) {
    // Tratamos el elemento
    console.log(elem)
});
```

Con la función arrow => de ES6, el código anterior se sustituiría por:

```
//ES6
var data = [{... }, {... }, {... }, ...];
data.forEach(elem => {
    console.log(elem);
});
```

4.2 This dentro de un arrow function

La variable this muchas veces se vuelve un dolor de cabeza. antíguamente teníamos que cachearlo en otra variable ya que solo hace referencia al contexto en el que nos encontremos y posteriormente, con la aparición de ES5 podíamos hacer uso del método bind para hacer referencia a un contexto concreto.

En ES6 y aciendo uso de arrow functions podemos hacer cosas así:

```
//ES6
var obj = {
    foo: function() {... },
    bar: function() {
        document.addEventListener("click", (e) => this.foo());
    }
}
```

5. TEMPLATE STRINGS

Con ES6 podemos interpolar Strings de una forma más sencilla que como estábamos haciendo hasta ahora. Fíjate en este ejemplo:

```
//ES6
let nombre1 = "JavaScript";
let nombre2 = "awesome";
console.log(`Sólo quiero decir que ${nombre1} is ${nombre2`);

//Solo quiero decir que JavaScript is awesome
```

También podemos tener String multilínea sin necesidad de concatenarlos con +.

```
//ES5
var saludo = "hola " +
"que " +
"tal?";

//ES6
var saludo = "hola
que
tal?";

console.log("hola
que
tal?");
```

6. DESTRUCTURING

Tenemos nuevas formas de asignar valores a Arrays y a Objetos.

Veamos unos ejemplos:

```
var [a, b] = ["hola", "mundo"];
console.log(a); // "hola"
console.log(b); // "mundo"

var obj = { nombre: "Carlos", apellido: "Azaustre" };
var { nombre, apellido } = obj;
console.log(nombre); // "Carlos"
```

No se parece en nada a la manera tradicional de asignar variables ¿verdad?

Pongamos algún ejemplo más:

```
var foo = function() {
    return ["175", "75"];
};
var [estatura, peso] = foo();
console.log(estatura); //175
console.log(peso); //75
```

7. VALORES POR DEFECTO

Otra novedad es asignar valores por defecto a las variables que se pasan por parámetros en las funciones. Antes teníamos que comprobar si la variable ya tenía un valor. Ahora con ES6 se la podemos asignar según creemos la función.

```
//ES5
function(valor) {
    valor = valor || "foo";
}

//ES6
function(valor = "foo") {... };
```

8. MÓDULOS

Gracias a ES6 podremos llamar a las funciones desde los propios Scripts, sin tener que importarlos en el HTML, si usamos JavaScript en el navegador.

Ejemplo:

```
//File: lib/person.js
module "person" {
    export function hello(nombre) {
        return nombre;
    }
}
```

Y para importar en otro fichero:

```
//File: app.js
import { hello } from "person";
var app = {
    foo: function() {
        hello("Carlos");
    }
}
export app;
```

9. PETICIONES AJAX CON FETCH

La API Fetch proporciona una interfaz para recuperar recursos (incluyendo recursos remotos a través de redes). Le resultará familiar a cualquiera que haya usado XMLHttpRequest, pero ésta nueva API ofrece un conjunto de características más potente y flexible.

Veamos un ejemplo descargando una imagen:

```
var misCabeceras = new Headers();

var miInit = {
    method: 'GET',
    headers: misCabeceras,
    mode: 'cors',
    cache: 'default'
};

fetch('flores.jpg', miInit)
    .then(function(response) {
        return response.blob();
    })
    .then(function(miBlob) {
        var objectURL = URL.createObjectURL(miBlob);
        miImagen.src = objectURL;
    });
```

10. TRANSPILANDO CON BABEL

Puede que hasta el título te haya sonado a chino. ¿Qué es eso de transpilar?

Transpilar consiste en traducir lenguajes que operan en aproximadamente el mismo nivel de abstracción, mientras que un compilador tradicional traduce de un lenguaje de alto nivel a un lenguaje de nivel inferior.

Actualmente existen varias herramientas que nos ofrecen diferentes posibilidades a la hora de transformar nuestro nuevo código JavaScript. Entre las más utilizadas tenemos Browsserify y por otro lado los task-runners Grunt y Gulp, pero en todos se suele usar el core de Babel para transformar el código de ES6 a ES5.

Puedes encontrar el proyecto y ejemplos en: https://babeljs.io/