

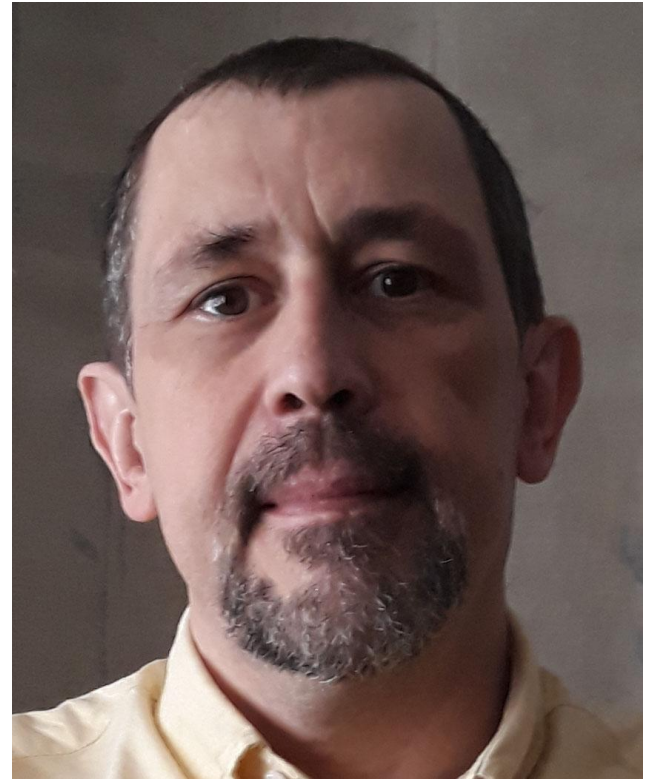
Fundamentos de HTML, CSS y JavaScript





 Alejandro Cerezo Lasne

alce65@hotmail.es



JavaScript Básico. ES6.

Variables y ámbitos. Funciones.





■ Introducción



■ Origen y evolución del lenguaje

1995 - Netscape incorpora un JS desarrollado por Brendan Eich

Objetivo: validación de formularios en cliente

(Respuesta a la lentitud de las conexiones)

1997 - Aparece ECMAScript

(European Computer Manufacturers Association)

1998 - OWA incorpora XMLHttp (AJAX)

2004 - GMail utiliza JS incorporando AJAX masivamente

2006 - John Resig crea JQuery

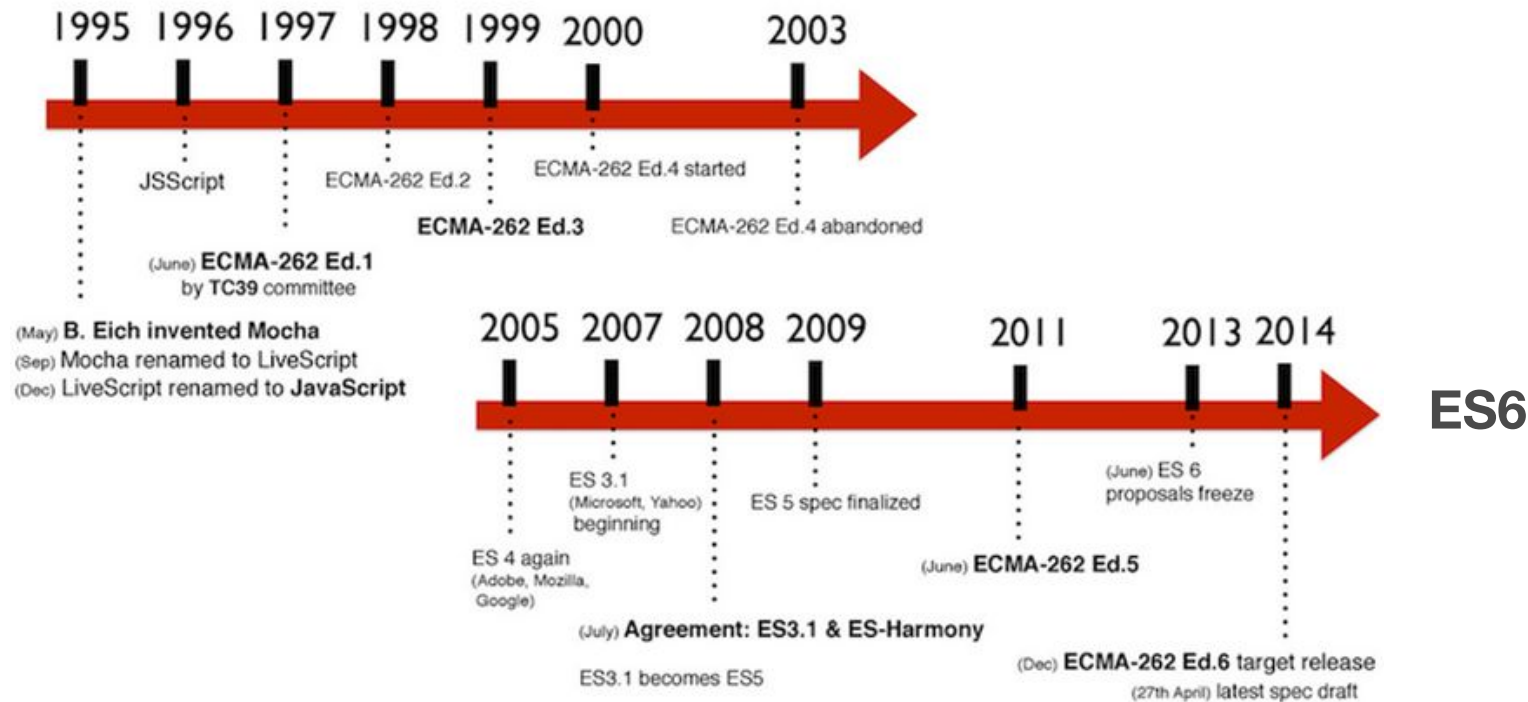
2009 - Aparece NodeJS, JS en el servidor, de mano de Ryan Dahl

2012 - Windows 8 incorpora JS como lenguaje nativo

2014 - Nuevo estandar ES6 / ES2015

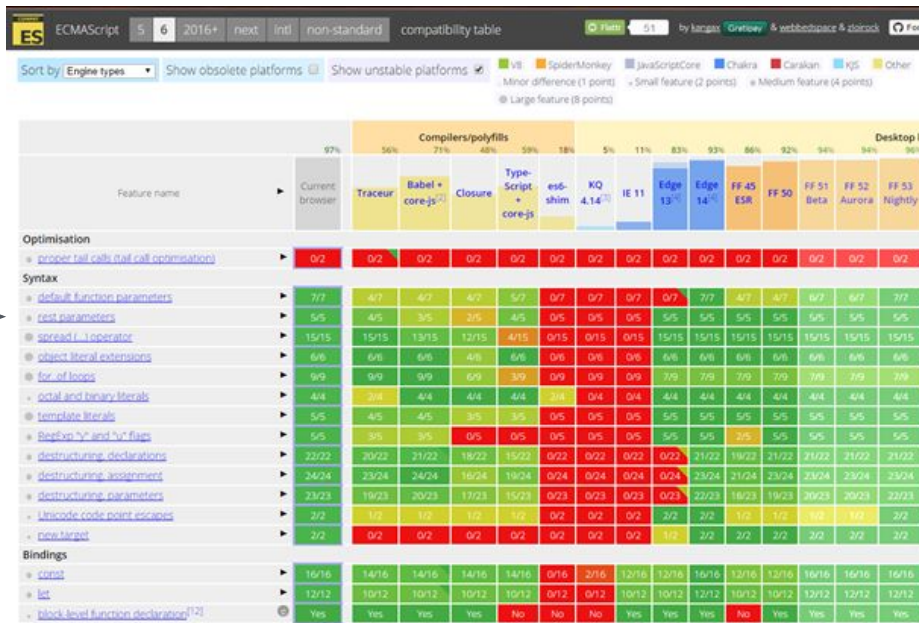


Estándares y versiones



Más información

<http://kangax.github.io/compat-table/es6/>



The screenshot shows the ES6 compatibility table by kangax. The table lists various ES6 features and their compatibility status across different browsers and compilers. The features are categorized into Optimisation, Syntax, and Bindings. The browsers and compilers listed are: Current browser, Traceur, Babel + core-js, Closure, Type-Script + core-js, es-shim, KQ 4.14, IE 11, Edge 13, Edge 14, FF 45 ESR, FF 50, FF 51 Beta, FF 52 Aurora, and FF 53 Nightly. The compatibility status is indicated by a color-coded cell: green for 'Yes', red for 'No', and yellow for 'Partial' or 'Not implemented'. The table also includes a 'Sort by' dropdown menu and checkboxes for 'Show obsolete platforms' and 'Show unstable platforms'.

Feature name	Current browser	Traceur	Babel + core-js	Closure	Type-Script + core-js	es-shim	KQ 4.14	IE 11	Edge 13	Edge 14	FF 45 ESR	FF 50	FF 51 Beta	FF 52 Aurora	FF 53 Nightly
Optimisation															
• proper tail calls (tail call optimisation)	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2
Syntax															
• default function parameters	7/7	4/7	4/7	4/7	5/7	0/7	0/7	0/7	0/7	7/7	4/7	4/7	6/7	6/7	7/7
• rest parameters	5/5	4/5	5/5	2/5	4/5	0/5	0/5	0/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5
• spread (...) operator	15/15	15/15	13/15	12/15	4/15	0/15	0/15	0/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15
• object literal extensions	6/6	6/6	6/6	4/6	6/6	0/6	0/6	0/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6
• for...of loops	9/9	9/9	9/9	3/9	0/9	0/9	0/9	0/9	7/9	7/9	7/9	7/9	7/9	7/9	7/9
• octal and binary literals	4/4	3/4	4/4	4/4	4/4	3/4	0/4	0/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4
• template literals	5/5	4/5	4/5	3/5	3/5	0/5	0/5	0/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5
• BigInt "n" and "N" flags	5/5	3/5	3/5	0/5	0/5	0/5	0/5	0/5	3/5	5/5	0/5	5/5	5/5	5/5	5/5
• destructuring declarations	22/22	20/22	21/22	18/22	15/22	0/22	0/22	0/22	0/22	21/22	19/22	21/22	21/22	21/22	21/22
• destructuring assignment	24/24	23/24	24/24	18/24	0/24	0/24	0/24	0/24	0/24	23/24	21/24	23/24	23/24	23/24	23/24
• destructuring parameters	23/23	19/23	20/23	17/23	15/23	0/23	0/23	0/23	0/23	22/23	18/23	19/23	20/23	20/23	22/23
• Unicode code point escapes	2/2	1/2	1/2	1/2	1/2	0/2	0/2	0/2	2/2	2/2	1/2	1/2	1/2	1/2	2/2
• new target	2/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	1/2	2/2	2/2	2/2	2/2	2/2	2/2
Bindings															
• const	16/16	14/16	14/16	14/16	14/16	0/16	2/16	12/16	12/16	16/16	12/16	12/16	16/16	16/16	16/16
• let	12/12	10/12	10/12	10/12	10/12	0/12	0/12	10/12	10/12	12/12	10/12	10/12	12/12	12/12	12/12
• block-level function declaration	Yes	Yes	Yes	Yes	No	No	No	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes

<http://es6-features.org/>

ECMAScript 6 — New Features: Overview & Comparison



■ Entornos de ejecución

- El núcleo (core) del lenguaje JavaScript API mínima
 - Manejo de datos, textos, arrays, y expresiones regulares
 - No hay ninguna funcionalidad de las entradas y salidas
 - No hay funciones más sofisticadas: redes, almacenamiento, gráficos...
- Navegadores. Soporte de ES6.
 - Transpilación (babel)
- NodeJS



JavaScript 101

- “use strict” en ES5
- sintaxis ES6
- variables y scopes
- funciones
- control de flujo
- “objetos”

Entorno de trabajo: NodeJS

- Permite conocer el core del lenguaje sin prestar atención a los elementos específicos del navegador
- Salida de datos mediante `console.log()`



JavaScript 101: variables y scopes

- Variables. Ámbitos: funciones y bloques.
 - Var. ES6: let, const
- Tipos de datos. Tipado dinámico. Casting automático
 - Number y String. ES6: string template
 - Boolean: valores falsy
 - Undefined y null (object).
 - Object: tipos referenciados. Arrays
- Operadores
 - Particularidades de la suma
 - Comparación (==) y comparación estricta (===)
 - Operador ternario



Código. Conceptos de JS

Variables en ES6: var, let, const

Template string

Booleans en JS. Operador ternario

Primitivos (elementales) y referencias



JavaScript 101: control de flujo

- Condicionales
 - If / else if / else
 - switch / case
- Iteraciones
 - for
 - do / while



JavaScript 101: funciones

- Declaraciones v. asignación a variables de funciones anónimas
 - Hoisting
 - ES6: Array functions
- Ámbitos de las variables. Variables locales
- Argumentos y parámetros.
 - Declaración: **parámetros** (reales)
 - Invocación: parámetros formales o **argumentos**
 - Valores y referencias
 - ES6: valores por defecto
 - ES6: spread operator



Código. Funciones en JS

Creación de funciones.
Argumentos. Valores por defecto
Spread operator



JavaScript 101: funciones como objetos

- Callbacks
- Funciones anidadas
- Funciones autoinvocadas: patrón IIFE
(Immediately-invoked function expression)
 (function (parámetros) {
 código de la función
 } (argumentos))
- Closures



Código. Funciones en JS

Callbacks
Funciones anidadas
Funciones autoinvocadas



■ Closures

- función que incluye variables locales y funciones
- generalmente autoinvocada:
- devuelve una función local o un objeto con un conjunto de ellas

- encapsula el código siguiendo un patrón “módulo”
- crea un interface al exterior
- mantiene un ámbito local que existe en sucesivas invocaciones al interfaz





JavaScript Básico. ES6.

Objetos y clases



■ Concepto de Objetos en JS

- No se aplica el concepto de otros lenguajes de los objetos como instancias de las clases -> **NO EXISTEN CLASES**
- Un objeto es una **colección de propiedades**, cada una con un nombre y un valor (técnicamente la referencia al área de memoria donde se almacenan las propiedades)
- Las propiedades pueden ser de cualquier tipo, incluyendo otros objetos, y por tanto **funciones**
- Todo objeto se crea en base a un prototipo, que en última instancia es el prototipo de Object -> **lenguaje prototípico**



■ Objetos literales o JSON

Objetos literales = "Diccionarios" en Java...

```
let persona = {nombre; 'Pepe', edad: 12, isAlumno = true}
```

En realidad el compilador ejecuta 2 operaciones

```
let persona = new Object()  
persona = nombre; 'Pepe', edad: 12, isAlumno = true}
```

El operador new crea un objeto vinculado al prototipo de Object, cuyas propiedades pueden ser definidas dinámicamente



■ Acceso a las propiedades

```
let persona = {nombre; 'Pepe', edad: 12, isAlumno = true}
```

Dos mecanismos para acceder a las propiedades

- con el sufijo [] persona['nombre'] -> Pepe
- notación de puntos persona.nombre -> Pepe

El primero es útil para utilizar una variable con el nombre de una propiedad

```
let propiedad = "nombre"  
persona[propiedad] -> Pepe
```



■ Métodos

- Los objetos son bastante más que estructuras de árbol (grafo) para recopilar y organizar datos
- entre sus propiedades se incluyen funciones = **métodos**
- El uso de los métodos es equivalente al de las otras propiedades , incorporando el () y dentro los parámetros que existan
-
- Cualquier método puede hacer referencia al propio objeto que lo contiene, mediante **this**



■ Iteraciones con objetos

```
for (clave in objeto) {  
    objeto[clave]  
}
```

- la notación [] permite recorrer el objeto
- conviene distinguir, para tratarlas de forma diferente:
 - las propiedades propias, no del prototipo
hasOwnProperty(key)
 - las propiedades de tipo función o métodos
typeof === 'function'



Código. Objetos en JS

Creación de un objeto y acceso a sus propiedades
Iteraciones con objetos



■ Objetos de tipos primitivos

JavaScript definen objetos para cada uno de los tipos de datos primitivos

objetos String
objetos Number
objetos Boolean

Almacenan los mismos valores de los tipos de datos primitivos y añaden propiedades y métodos para manipular sus valores.

Los números, cadenas y booleanos se asocian automáticamente y de forma dinámica a objetos envolventes (Wrapper Objects)



■ Operaciones con cadenas

- length
- concat()
- split(separator)
- toUpperCase()
- toLowerCase()
- slice(inicio, final)
- substring(inicio, final)
- substr(inicio, desplazamiento)
- charAt(position)
- charCodeAt(position)
- indexOf(caracter)
- lastIndexOf(caracter)
- match(regExp)
- search(expresion)
- replace(expresion1, expresion2)

String.fromCharCode(numero)



■ Arrays

- sucesión de valores
- en lugar de una clave, se acceden por su posición
- se cuentan desde 0
- son objetos con su propio prototipo, Array
- no suelen incorporar propiedades definidas individualmente para un array
- se crean de forma similar a los objetos, utilizando el operador {}
let aDatos = [....]

Internamente corresponde a `let aDatos = new Array()`



■ Operaciones con Arrays

- `length`
- `concat()`
- `join(separator)`
- `sort()`
- `reverse()`
- `slice(inicio,fin)`
- `splice(inicio, cuantos, [nuevo valor, ...])`
- `push()`
- `pop()`
- `shift()`
- `unshift()`



■ Operaciones “funcionales” con arrays (ES5)

- `map()`
- `filter()`
- `some()`
- `every()`
- `forEach()`
- `reduce()`
- `reduceRight()`

En todos los nuevos métodos se utiliza una función callback, es decir una función que es pasada como parámetro para que el método la utilice de la forma en que tiene previamente definida.

Los parámetros de dicha función son (element, index, array)



Código. Objetos “nativos” del lenguaje.

Objetos envolventes

- Propiedades asociadas al objeto number
- Propiedades asociadas al objeto string

Arrays

- Propiedades de arrays. Propiedades funcionales



■ OOP en JS

- Basada en prototipos y con ausencia de clases
- Potente notación literal de objetos (de ahí el JSON)
- Objetos dinámicos: pueden modificarse en todo momento
- Operador new para invocar al método constructor
- Herencia prototipada a partir de otros objeto
- Muy orientado a Eventos



■ Operador this y formas de invocación

- Patrón de invocación Function
this es el objeto global del programa
- Patrón de invocación Method
this es el objeto en el que está el método
- Patrón de invocación Constructor
this es el nuevo objeto creado
- Patrón de invocación Apply
permite definir el valor de this

apply() y call() son métodos del objeto Function, que Permiten ejecutar una función como si fuera un método de otro objeto: cambian el binding de la función ligándola al objeto, que pasa a ser this dentro de la función



■ This en los callbacks

- Cuando un método se ejecuta en el objeto, **this** hace referencia a dicho objeto
- En el marco de la función que ejecuta un método recibido callback, this hace referencia al entorno de la función, generalmente el objeto global (window en el caso del navegador)
 - El método debe ser aplicado (apply o call) con referencia al objeto original, operación incorporada al método bind() de Object.prototype
 - En las funciones arrow, this es consistente y siempre hace referencia al objeto en que se definió el método, con independencia de su ejecución



Código. Objetos “propios”.

- Formas de invocación. Significado de this
- Problema de this en los callbacks
- Soluciones



■ Objetos propios. Funciones constructoras.

- Internamente utilizan la referencia `this`, definiendo una serie de propiedades
- Si se invocan utilizando `new` producen la instanciación de un nuevo objeto, que queda ligado al prototipo de la función constructora.
 - se crea un **objeto genérico** del tipo *Object*
 - se **asigna** al prototipo de ese objeto genérico la función constructora invocada mediante `new`
 - se **ejecuta** la función constructora
 - se devuelve el objeto inicialmente de tipo *Object* pero cuyo prototipo a cambiado a la clase constructora



■ Constructoras y métodos. Prototype

- Se pueden incorporar métodos en las funciones constructoras propias que creamos.
- Cada instancia en la que invocamos un constructor copiaría en el nuevo objeto los métodos que contiene dicho constructor
- Para evitar ese problema, podemos reescribir la función constructora utilizando el objeto prototype.



■ ¿Que es el prototype?

- Todos los objetos tienen una propiedad denominada prototipo (prototype) que apunta al objeto (función) a partir del que han sido creados:
 - el objeto genérico Object.
 - alguno de los objetos predefinidos, Array, Date ...
 - un objeto de igual nombre, en caso de una función
 - un objeto definido por el usuario



Código. Objetos “propios”.

- Funciones constructoras
- Métodos en el prototipo



■ Herencia en JavaScript (1)

- Patrones que trabajan directamente con objetos
 - Herencia prototípica (prototypal inheritance)
 - Copia de propiedades del objeto o copia superficial (shallow copy)
 - Copia del objeto en profundidad (deep copy)
 - Extensión y aumento
 - Herencia múltiple
 - Herencia parasitaria



■ Herencia en JavaScript (2)

- Patrones que trabajan con constructores ("clásicos")
 - Encadenamiento de prototipos (Prototype chaining)
 - Herencia de prototipos
 - Encadenamiento con constructor temporal
 - Copia de las propiedades del prototipo



■ Herencia prototípica: Object.create()

- El método estático Object.create aparece en ES5 recogiendo el patrón de herencia prototípica propuesto por Douglas Crockford
- A partir de un objeto oPadre (e.g. creado literalmente)
oHijo = Object.create(oPadre, {nuevas propiedades})
devuelve un nuevo objeto con el primero como prototipo

El patrón empleado internamente es el siguiente



```
function object(o) {  
  function F() {}  
  F.prototype = o;  
  return new F();  
}
```



■ Nueva OOP en ES6

- Clases: constructor() y propiedades
- Métodos
- Herencia



Código. Objetos en ES6.

- Clases: constructor() y propiedades
- Métodos
- Herencia





JavaScript en La Web. BOM. DOM



JavaScript en la Web

- JavaScript inline
- JavaScript en bloques
- JavaScript desde archivos externos
 - Posición de los scripts
 - Agrupación del código en funciones (encapsulación)
Funciones autoinvocadas ->
IIEF (Immediately-invoked function expression)
 - Espera a la carga completa del HTML antes de ejecutar JS
eventos `window.load` / `document.DOMContentLoaded` (sin contar hojas de estilo, imágenes...)



■ Depuración en Chrome

- Puntos de ruptura desde código: debugger
- Uso del depurador
- Mensajes por consola.
 - `console.dir()`
 - `console.log()`
 - `console.group()` para recorrer objetos
 - `console.table()`



- Window
 - Document (dentro del BOM) →
 - anchors
 - forms
 - images
 - links
 - location
 - History
 - Location
 - Screen
 - Navigator



Window

alert(),
confirm()
prompt()
print()
open()
close()
createPopup()
focus()
blur()
stop()

moveBy()
moveTo()
resizeBy()
resizeTo()
scrollBy()
scrollTo()
setInterval()
setTimeout()
clearInterval()
clearTimeout()

screenX
screenY
innerWidth
innerHeight
outerWidth
outerHeight

(APIS HTML5)
localStorage
sessionStorage



■ Navigator

appName

appVersion

appMinorVersion

userAgent

product

productSub

browserLanguage

language

systemLanguage

userLanguage

cpuClass

oscpu

platform

userProfile

securityPolicy

mimeTypes

plugins

cookieEnabled

javaEnabled()

onLine

preference()



location

href

protocol

host name

pathname

hash

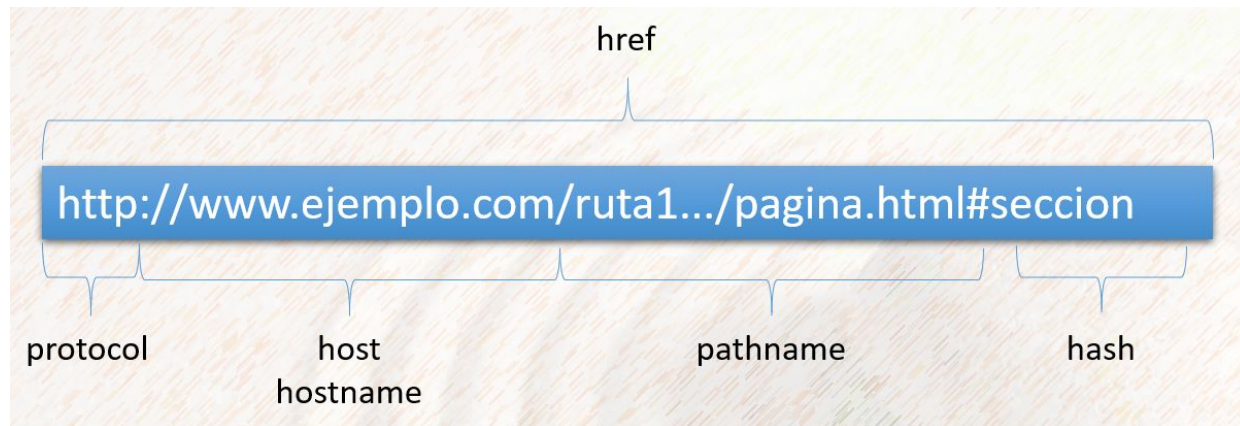
port

search

assign()

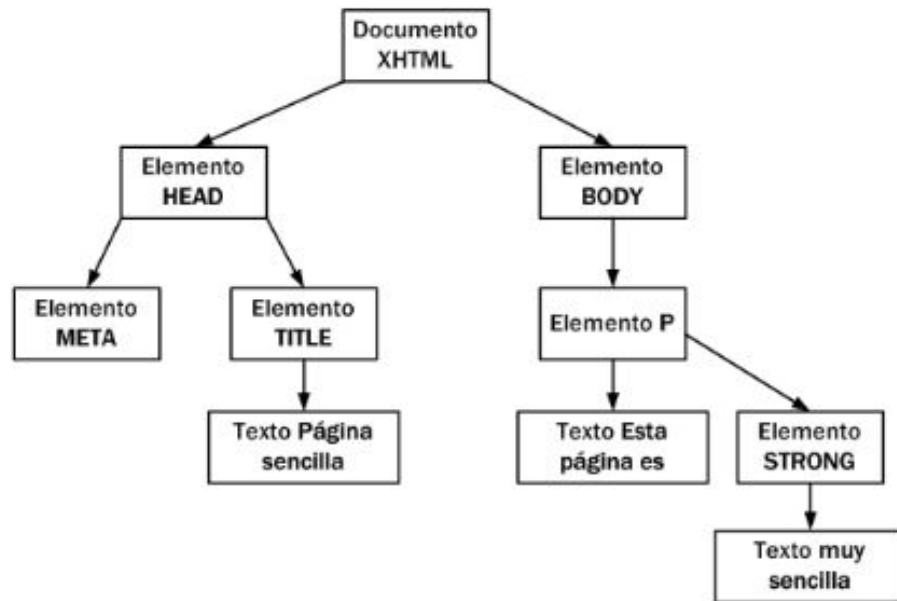
replace()

reload()



DOM

- Estructura jerárquica de nodos
 - base de los motores de renderizado
 - se refleja en los objetos Node de JS



- Nodos
- NodeList (similares a arrays)
- HTMLCollection (solo tipo element)



■ Nodos. Relaciones entre ellos

Propiedades de un nodo

- childNodes (NodeList)
- children ()HTMLCollection
- hasChildNodes()
- firstChild
- lastChild
- previousSibling
- nextSibling
- childElementCount
- firstElementChild
- lastElementChild
- previousElementSibling
- nextElementSiblin



DOM. Seleccionar elementos

Recuperando elementos

- `nodo.getElementById()`
- `nodo.getElementsByTagName()`
- `nodo.getElementsByName()`
- `nodo.getElementsByClassName()`
- `nodo.querySelector()`
- `nodo.querySelectorAll()`



■ Crear y borrar elementos

De forma tradicional

- `createElement()`
- `createTextNode()`
- `parent.appendChild(node)`
- `parent.removeChild(node)`

En HTML5

- `innerHTML()`
- `outerHTML()`



■ Manipular elementos

Elementos "en memoria"

- `cloneNode()`
- `createElement()`
- `appendElement()`
- `removeElement()`
- `replaceElement()`

Modificar elementos:

- `textContent`
- `innerHTML`



■ Manipulando Atributos

- attributes
- [nombre-atributo]
- setAttribute()
- getAttribute()
- hasAttribute()
- removeAttribute()

Mención aparte los atributos class y style



■ Manipulando CSS

- Aplicación directa de estilos: `style.[nombre-propiedad]`
(paso de kebab-case a camelCase)
- Uso de clases CSS.
 - `className`
 - `classList`
 - `classList.add(clase1, clase2,...)`
 - `classList.remove(clase1, clase2,...)`
 - `classList.toggle(clase1)`
 - `classList.contains(clase1)`



Proyecto. JS

- Revisión de detalles pendientes
- Añadimos JS al proyecto



■ Módulos en ES6

- Definición de un módulo: corresponde al fichero que lo incluye

```
//File: modulo.js
export const hello = (nombre) => {
  return "Hola " + nombre;
}
```

- Uso del módulo: importación de una función

```
//File: app.js
import { hello } from "./modulo.js";
```

- Acceso desde HTML a la estructura de módulos:
se vincula con script el fichero importador, utilizando el atributo type="module"

```
<script src="app.js" type="module"></script>
```



■ Eventos en JavaScript



Eventos en el navegador

- Resource Events
- Network Events
- **Focus Events**
- Websocket Events
- Session History Events
- CSS Animation Events
- CSS Transition Events
- **Form Events**
- Text Composition Events
- **View Events**
- Clipboard Events
- **Keyboard Events**
- **Mouse Events**
- Drag & Drop Events
- Media Events
- Progress Events

MDN web docs
mozilla

Search

Technologies ▾ References & Guides ▾ Feedback ▾

Sign In

Event reference

Languages Edit Settings

Web technology for
developers >

Event reference

Related Topics

Events

- ▶ Ambient Light events
- ▶ App Cache events
- ▶ Audio Channels API events
- ▶ Battery API events
- ▶ Broadcast Channel API events

DOM Events are sent to notify code of interesting things that have taken place. Each event is represented by an object which is based on the [Event](#) interface, and may have additional custom fields and/or functions used to get additional information about what happened. Events can represent everything from basic user interactions to automated notifications of things happening in the rendering model.

This article offers a list of events that can be sent; some are standard events defined in official specifications, while others are events used internally by specific browsers; for example, Mozilla-specific events are listed so that add-ons can use them to interact with the browser.

<https://developer.mozilla.org/en-US/docs/Web/Events>



■ Eventos frecuentes

- Mouse Events
 - mouseenter / mouseleave
 - mouseover / mouseout
 - click / dblclick
 - select
- View Events
 - scroll
- Focus / Form ... Events
 - focus / blur
 - reset / submit
 - change / input



■ Respuesta a Eventos

- Eventos en línea: los atributos HTML `on<evento>` mala práctica si es fuera de un determinado framework.
- Manejadores semánticos o event handlers: las propiedades `on<evento>` de los nodos del DOM
- Event Listener: buena práctica.
 - `addEventListener()`
 - `removeEventListener()`



Objeto Event

- currentTarget, target: elemento que desencadena el evento
- type: nombre del evento
- bubbles
- cancelable
- defaultPrevented
- eventPhase
- isTrusted
- timeStamp
- view
- initEvent()
- preventDefault():
comportamientos por defecto
- stopImmediatePropagation()
- stopPropagation()



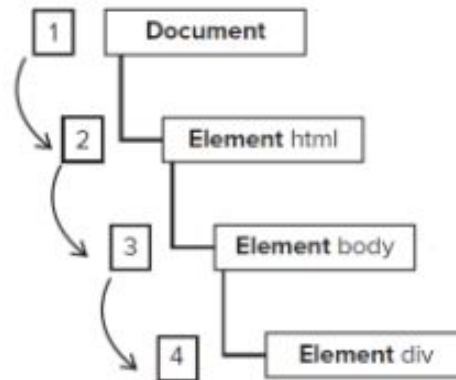
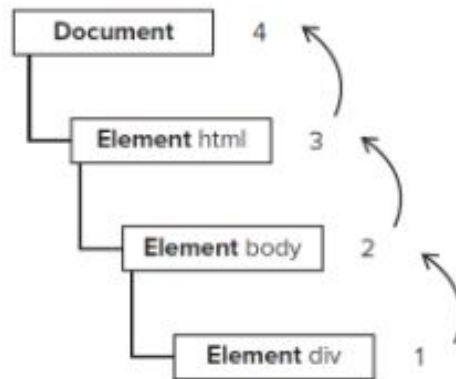
Proyecto. Eventos.

- Comportamiento de los botones
“Older Posts”, “Send”, “Icono de navegación”
- Manejadores con console.log -> Objeto Event
Escribiendo en la Web : innerHTML
- “Icono de navegación” -> cambios en las clases CSS
- input -> eventos change / input y Objeto Evento



Flujo de Eventos

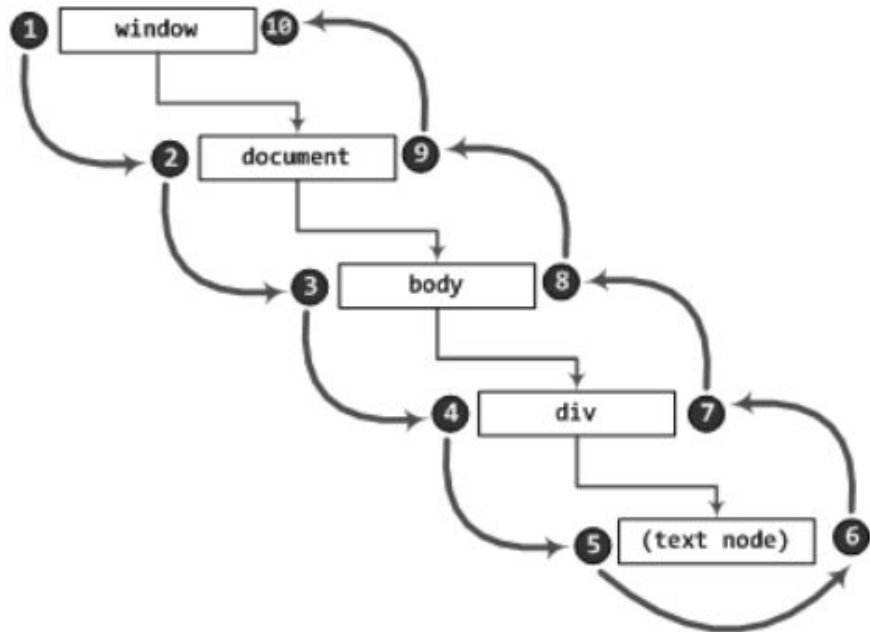
- event bubbling
(propagación de eventos)
- event capture
(captura de eventos)



Flujo de Eventos en DOM2

1. **event capture**
(captura de eventos)
2. event bubbling
(propagación de eventos)

en `addEventListener`, el tercer parámetro es activa/desactiva el flujo bubbling (default false)



■ Eventos propios

API EventTarget del DOM2

- addEventListener() / removeEventListener()
- **dispatchEvent(oEvento)**

capaz de lanzar (trigger o dispatch), desde un nodo del DOM, un evento como si se tratara de un evento de lanzamiento directo. Puede traducir eventos del sistema (e.g. click) a eventos semánticos (e.g. formularioEnviado)

```
elemento.dispatchEvent(new Event(<nombre evento>))
```



Proyecto. Eventos.

- Scroll-spy y navegación





■ Formularios



■ Controles de formularios

`<input> -- type text`

`<input> -- type submit/ reset / button`

`<button><button> -- type submit/ reset / button`

`<textarea> </textarea>`

`<input> type checkbox / radio`

`<select> <option> valor </ option>..... </select>`

`<input> -- type text/ email / tel/ URL / search`

`<input> type color / number/ range/ date...`



Proyecto. Frormularios.

- Completamos el formulario de contactos, para que incluya
 - checkbox
 - radiobuttons
 - selectOptions



■ Eventos frecuentes

- reset (Form Events)
- submit (Form Events)

- focus (Focus Events)
- blur (Focus Events)

- change
- input

- click (Mouse Events)

<input>
<textarea>
<select>

<input submit>
<input submit>
<buton>



■ Formularios: Obtención de datos

- input / textarea: propiedad/atributo value del nodo
- checkbox -> propiedad/atributo checked del nodo
- radiobutton -> `aNodos.foreach(
 item.checked = true -> item.value`
- Select/options -> del array options
 el `item[selectedIndex]`



Proyecto. Formularios.

- Recogemos todos los datos en un objeto al enviar (tendremos activado el `preventDefault()` en el evento `submit` del formulario)



■ Formularios: controles dinámicos

- Controles dinámicos (basados en código)
 - Cuando cambia el valor de un elemento de selección (radio - check - select/options)
-> evento change
 - el manejador correspondiente genera el control dinámico a partir de los datos incluidos en el código o recibidos desde un API



■ Formularios y validación. Planteamiento

- Validación basada en HTML5 más o menos customizada
- Botón enviar: puede deshabilitarse hasta que el formulario sea válido
- Mensajes de error
 - aparecen cuando se abandona un campo sin terminarlo correctamente
 - Si enviar no esta deshabilitado, pueden aparecer solo al pulsar el botón de enviar, en caso de que existan errores de validación

En este caso, el primero de los controles afectados puede recibir el foco -> `nodo.focus()`



■ Validación en HTML5

- Atributos de los controles
 - required
 - pattern / algunos asociados a tipos de input
 - min
 - max
- Atributos del formulario
 - novalidate



■ API de validación de restricciones

La validación definida en HTML se realiza de varias maneras

- mediante el método **checkValidity()**
 - de un input, textarea, select o button, que evalúa específicamente el elemento correspondiente
 - de un formulario : validación estática
- al principio del método **submit()**, cuando este es disparado por el clic en un control de tipo submit:
validación interactiva (mensajes emergentes)



Validación y submit

La validación interactiva aparece inmediatamente por delante de la respuesta al evento submit disparado por el formulario como consecuencia de un clic en un control de tipo submit



Si se define una función manejadora del evento submit del formulario:
la función se desencadena justo después de la validación
en ella es posible deshabilitar el comportamiento por defecto de envío y hacer cualquier operación con los datos (ya validados)



Proyecto. Formularios.

- Añadir validación





JavaScript Asíncrono



■ JS Asíncrono

- Controlando el tiempo: Eventos temporales timeouts e intervals
 - setTimeout / clearTimeout
 - setInterval / clearInterval
- Simulación con setTimeout de peticiones a servidores (ejemplo de código asíncrono)
- Callbacks y promesas
- Peticiones a servidores
 - AJAX
 - fetch



■ Promesas

Una **Promise** (promesa) es un **objeto** que representa la terminación o el fracaso eventual de una **operación asíncrona**.

- Una promesa puede ser creada usando su constructor.
- Sin embargo, la mayoría de las operaciones programadas son consumidoras de promesas ya creadas devueltas desde funciones.
- Esencialmente, una promesa es un objeto devuelto al cual enganchas las funciones callback, en vez de pasar funciones callback a una función.



■ Promesas en ES6. Implementación

new Promise((resolve, reject) => {})

el objeto promesa recibe como parámetros dos funciones:

- La función "resolve": se ejecutará cuando queramos finalizar la promesa con éxito.
- La función "reject": se ejecutará cuando queramos finalizar una promesa informando de un caso de fracaso.



■ Promesas en ES6. Utilización

a la función que retorna el objeto promesa se le encadenan el método `then` con dos funciones como parámetros

- la función que se ejecutará cuando la promesa haya finalizado con éxito.
- la función que se ejecutará cuando la promesa haya finalizado informando de un caso de fracaso.

Alternativamente puede utilizarse el método `catch` para declarar la función que se ejecutará cuando la promesa haya finalizado informando de un caso de fracaso.



■ API Fetch

La API Fetch proporciona una interfaz JavaScript para acceder y manipular partes del flujo HTTP (HTTP pipeline), incluyendo peticiones y respuestas.

Proporciona un método global `fetch()` que supone un mecanismo fácil y lógico de obtener recursos por la red de forma asíncrona

El método `fetch()` devuelve una promesa, incluso en los casos de errores 404 o 500

Es una mejor alternativa al uso de `XMLHttpRequest`:

- puede ser empleada fácilmente por otras tecnologías como Service Workers.
- aporta un único lugar lógico en el que definir conceptos como CORS y extensiones a HTTP.

<https://caniuse.com/#feat=fetch>



■ Metodo fetch()

```
fetch('url', {configuración})
```

```
.then((response) => {  
  return response.metod()  
})
```

```
.then((data) => {  
  // transformación de los  
  datos  
})
```

```
{ method: 'GET',  
  headers: misCabeceras,  
  body: data  
  mode: 'cors',  
  cache: 'default' }
```

```
arraybuffer()  
blob()  
formData()  
json()  
text()
```



Código. Fetch (AJAX)

- Accediendo a un API
`https://www.googleapis.com/books/v1/volumes?q=intitle:<clave>`
- Simulación de un API



GRACIAS

www.keepcoding.io

