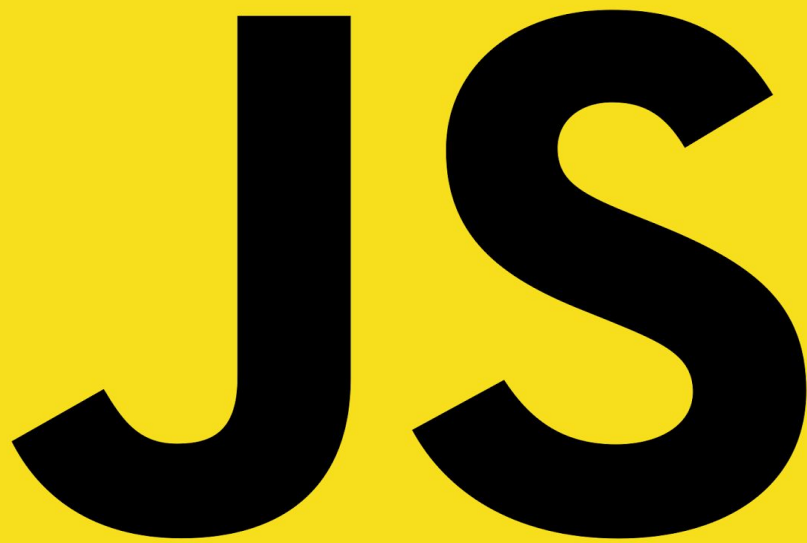


JavaScript

A large, bold, black 'JS' logo is centered on a solid yellow rectangular background. The letters are in a clean, sans-serif font.

Albert López Alcácer
Jordi Petit
Llenguatges de Programació
2017/2018

Índex

Com es va crear JavaScript?	3
Origen	3
Influències d'altres llenguatges	3
Estandarització i versions	4
Paradigmes de programació	4
Funcional	5
Basat en prototipus	5
Orientat a objectes	5
Imperatiu i procedural	5
Compilat o interpretat	6
Sistema de tipus	6
Principals aplicacions	7
Altres característiques	7
Tipus primitius i objectes	7
Scope, hoisting i els closures	8
JSON	8
Exemples de codi	9
Maneres de cridar funcions i la paraula clau "this":	9
Una pila amb "closures" i Fibonacci amb "cache":	10
Callbacks	10
Sobre les fonts d'informació	11
Bibliografia	12

Com es va crear JavaScript?

Origen

JavaScript va nèixer de la necessitat de fer les pàgines web interactives i ha esdevingut el llenguatge de scripting per a la WWW, i més.

L'any 1995 la companyia Netscape, propietària del navegador Netscape Navigator, va contractar a Brendan Eich amb l'objectiu de implementar Scheme (un dialecte de Lisp) en el navegador Navigator.

En aquell moment, però, Netscape també estava negociant amb Sun per a incloure Java en el navegador. De les explicacions dels que van defensar la necessitat d'un llenguatge de scripting, apart del llenguatge Java, va sorgir la idea de que aquest estaria orientat a dissenyadors i programadors web. Aquests farien servir els components fets amb Java (com els plugins o els Java applets) i implementarien les seves interaccions amb aquest llenguatge de scripting.

Netscape llavors va demanar a Brendan Eich un prototipus del llenguatge. Aquest va haver d'estar llest en un temps record de deu dies, amb el nom de Mocha.

Posteriorment es va anomenar LiveScript. Com que aquest nom no era suficientment confús i la popularitat de Java en aquell moment creixia, es va adoptar el nom de JavaScript com a estratègia comercial.

Influències d'altres llenguatges

JavaScript va ser influenciat per altres llenguatges:

- Sintàxi derivada de C i paraules claus i convencions pròpies de Java.
- Implementació de les funcions inspirades en AWK.
- JavaScript disposa de funcions de first-class i closures gràcies a Scheme.
- Orientació a objectes basada en prototipus per part de Self.
- Tractament de strings, expressions regulars i arrays per part de Perl y Python.

Estandarització i versions

Microsoft va implementar el mateix llenguatge amb el nom de JScript. Amb això, al 1996 Netscape va treballar en una estandarització del llenguatge amb el suport de Ecma International i es va registrar amb el nom de ECMAScript (per a respectar els drets de Sun sobre la marca JavaScript).

El llenguatge ha anat evolucionant amb les diferents versions:

- ECMAScript 1 (1997), ECMAScript 2 (1998) i ECMAScript 3 (1999) establint les bases del llenguatge actual.
- ECMAScript 3 (1999) inclou funcionalitats com el do-while, expressions regulars, tractament d'excepcions i nous mètodes per als strings.
- ECMAScript 5 (2009) i ECMAScript 5.1 (2011) on el primer afegeix el mode estricte i suport a JSON.
- ECMAScript 6 (2015) on es fan incorporacions com classes, mòduls, iteradors o constants.

JavaScript també ha tingut un paper important fora dels navegadors. Des de l'any 2000, han aparegut implementacions del llenguatge per la banda del servidor, com Node.js (2009) o el projecte CommonJS.

Paradigmes de programació

JavaScript és conegut per ser un llenguatge multiparadigma. Això significa que permet escriure programes fent servir diversos estils de programació, sent alguns més adequats que altres per segons quins problemes.

JavaScript doncs es compatible amb els paradigmes de programació següents:

1. Funcional.
2. Basat en prototipus.
3. Orientat a objectes.
4. Imperatiu i procedural.

Funcional

Partint de la base de que les funcions a JavaScript son de primera classe, disposem de closures, i a més a més tenim una sintaxi per a simular funcions lambda, fer servir programació funcional vol dir escriure programes fent ús de:

- *Funcions pures* (funcions reutilitzables que donats uns paràmetres, sempre donaran el mateix resultat sense efectes laterals). Mentre que normalment en el model imperatiu l'estat és emmagatzemat en els objectes i compartit fent ús dels seus mètodes, en la programació funcional l'estat flueix mentre passa d'una funció pura a una altra.
- *Composició de funcions* per a crear noves funcions a partir de la combinació d'altres. Com per exemple compondre f i g ($f \cdot g$) per obtenir la funció que fa $f(g(x))$.
- *Currificació de funcions* per a poder cridar funcions amb menys paràmetres dels que espera i que aquestes retornin funcions que esperen els paràmetres restants.
- Altres com Mònades (IO, Maybe, Either) o Functors ben coneguts en llenguatges com Haskell.

Basat en prototipus

Partint de la base de que no hi han classes, el model basat en prototipus permet fer programes amb funcionalitats iguals als fets amb classes.

Tenim objectes amb atributs i mètodes, tres maneres de crear objectes (nihilo, clonatge i extensió), i la manera d'enviar missatges (cridar mètodes).

Al model basat en prototipus tenim la idea de delegació: Si un objecte no entén un missatge, aquest es delega al seu pare. L'objecte pare al qual es delega i del que el fill hereta propietats es anomena prototipus.

Orientat a objectes

Amb JavaScript també podem programar amb un estil imperatiu i orientat a objectes. Amb el sistema de prototipus podem simular les funcionalitats bàsiques dels llenguatges amb classes, com l'herència o el polimorfisme.

Molts "frameworks" que fan servir aquest llenguatge utilitzen aquest paradigma. Fins i tot disposem de TypeScript, un superconjunt de JavaScript que incorpora la sintaxi corresponent per a poder fer servir classes i interfícies.

Imperatiu i procedural

Paradigma en el que s'escriu una seqüència d'instruccions amb l'objectiu d'anar modificant l'estat del programa. D'aquesta manera estem indicant *có*m realitzar la lògica del programa, al contrari d'un llenguatge declaratiu en el que especifiquem *què* volem realitzar.

JavaScript, igual que altres com C, és procedural en el sentit que podem tenir un seguit de funcions que criden altres funcions i retornen seguint un fil d'execució.

Compilat o interpretat

JavaScript es defineix sovint com un llenguatge interpretat pel fet de que el codi és distribuït als navegadors en el seu codi font i aquest és executat en el moment per donar uns resultats i no pas mitjançant binaris executables. Els navegadors fan servir un “engine” de JavaScript per executar el codi distribuït.

Dit això, javascript es compilat (“Just In Time” compilats, i no pas “Ahead of Time” com C o Java). Els engines de JavaScript realitzen els passos que son propis d’un compilador com identificar els tokens, fer el parsing per a construir l’AST i generar instruccions de codi màquina per a poder executar el programa. Aquestes tasques de compilació de l’engine es realitzen microsegons abans de que el codi sigui executat.

Concretament, l’engine V8 de Google compila el codi font de la següent manera: Un analitzador o parser genera un AST per a que després un intèrpret, anomenat Ignition, generi bytecode a partir d’aquest arbre. A més a més disposa d’un optimitzador que eventualment (per exemple quan es detecta que una funció amb uns certs tipus es fa servir sovint) genera codi màquina específic i optimitzat a partir d’aquest bytecode .

Sistema de tipus

JavaScript té un tipatge dinàmic, de manera que no li donem prèviament informació del tipus de les variables al compilador. Aquestes es declaren amb “var” o “let”. Els objectes son col·leccions dinàmiques de parells clau/valor, que poden ser mutats per referència en temps d’execució.

Això ens aporta als desenvolupadors beneficis importants a l’hora de programar. Per exemple no cal que coneguem exactament les propietats que tindrà un objecte obtingut a partir d’una crida que ens retorni un JSON. Per al compilador és un desavantatge, ja que aquest no podrà genera un codi màquina eficient per a executar des del primer moment, i caldran tècniques com les mencionades al paràgraf anterior.

Per un altre banda, no tindrem la seguretat de llenguatges com Haskell, en els que coneixem en temps de compilació que una funció s’està cridant amb els tipus correctes.

Mencionar que el llenguatge realitza un número limitat de comprovacions de tipus en temps d’execució i, algunes, fallen i no aturen el programa quan es donen. Per exemple, podem fer: `var foo = null; foo.prop` i obtindrem un `TypeError: Cannot read property 'prop' of null` Mentre que si fem: `var bar = {}; bar.prop` obtindrem un `undefined` de manera silenciosa. A més a més té un mecanisme de coerció per a poder operar variables de diferents tipus fent conversions implícites -i estranyes, com `[] + 1 = '1'`.

Principals aplicacions

Com s'ha mencionat al principi d'aquest document, JavaScript neix principalment de la necessitat de dotar de dinamisme i interacció a les pàgines web HTML des de la part del client (navegador). (com validar un formulari, o modificar el contingut de la pàgina segons la interacció concreta de l'usuari).

La manera amb la que és pot fer això és executant codi que pugui manipular el DOM (Document Object Model) associat a la pàgina que es visualitza. Es disposa de tecnologies com Ajax, que atorguen grans funcionalitats per al dinamisme i la interacció, i jQuery (el DOM fàcil de manipular i abstracte).

JavaScript també disposa de implementacions en el costat del servidor. La més coneguda és Node.js, que permet implementar servidors i generar contingut des d'aquests. També existeixen bases de dades NoSQL que funcionen amb objectes JSON i funcions (de creació de vistes, índexos, etc...) en JavaScript.

Pero els usos del llenguatge no s'aturen aquí! També podem desenvolupar aplicacions natives i multiplataforma per a Smartphones (Cordova), aplicacions d'escriptori multiplataforma (framework Electron), i fins i tot "sistemes operatius" com Chrome OS.

Altres característiques

En aquest apartat es vol fer una petita menció a un conjunt de característiques o funcionalitats del llenguatge que son importants i que cal conèixer quan es desenvolupa una aplicació amb aquest.

Aquestes característiques son:

Tipus primitius i objectes

Els tipus primitius son: *boolean*, *number*, *string*, *null*, i *undefined*. La resta de valors son *Object*. Una característica important d'aquests dos es com son comparats.

Els tipus primitius fan servir `==` per compara els valors amb *coerció* habilitada. Això vol dir que podem comparar valors com `'42' == 42`, que donarà *true*. Així mateix fer servir l'operador `===` ens permetrà comparar els valors però sense *coerció* de tipus, de manera que `'42' === 42` donara *false*.

Per un altre banda `===` es *true* per a dos objectes si i només si son el mateix objecte. Per a comparar objectes no podem fer servir aquests operadors, ni disposem de sobrecàrrega d'aquests. Així mateix `==` amb un objecte i un tipus primitiu farà que l'objecte es transformi a primitiu.

Scope, hoisting i els closures

El *scope* és el nom que rep el context lèxic que inclou els noms de les variables que son accessibles en un determinat punt del programa. Quan es declara una variable dins d'una funció amb *var*, aquesta te *scope* de funció : Estarà disponible en tot el bloc de funció. (Amb *let* el *scope* és de bloc, per exemple dins d'un *if*).

A més a més, la declaració de les variables son mogudes pel compilador a l'inici de la funció, tot i que aquestes es declarin en un altre lloc. Això no inclou, però, la inicialització d'aquesta (*hoisting*).

Suposant que una funció, quan es va crear, feia referència a variables accessibles en aquell context fora del seu scope de funció, seguirà podent accedir a aquestes variables tot i que el context sigui diferent (per exemple quan la funció sigui retornada d'un altre funció).

JSON

La idea del format JSON es la d'utilitzar la sintàxi de JavaScript per a poder enmagatzemar, llegir i transportar dades. JSON doncs utilitza la mateixa notació dels literals: arrays, strings, números i booleans d'una manera estructurada per a representar els objectes.

Exemple:

```
{
  "first": "Jane",
  "last": "Porter",
  "married": true,
  "born": 1890,
  "friends": [ "Tarzan", "Cheeta" ]
}
```


Exemples de codi

En aquesta secció veurem alguns exemples de codi per tal de donar un cop d'ull a algunes característiques conegudes del llenguatge.

- Maneres de cridar funcions i la paraula clau “this”:

Invocació com a Mètode	Invocació com a Funció
<pre>var myObject = { value: 0, increment: function (inc) { this.value += typeof inc === 'number'? inc : 1; } }</pre> <p>/*En aquest cas, la paraula clau this farà referència a l'objecte myObject que crida el mètode increment*/</p>	<pre>myObject.double = function () { var helper = function () { this.value = 2 * this.value; // Objecte global!!!! }; helper(); }</pre> <p>/*En aquest cas, el mètode double de myObject fa servir una funció anònima (que no es propietat de l'objecte). El this d'aquesta funció farà referència a l'objecte global i no pas a myObject*/</p>
	<pre>myObject.double = function () { var that = this; var helper = function () { that.value = 2 * that.value; // that fa referència a myObject }; helper(); }</pre> <p>/*Possible solució</p>
Invocació com a Constructor	Invocació amb “apply/call”
<pre>function Range(from,to) { this.from = from; this.to = to; } var r = new Range(1,10);</pre> <p>DynamicRange.prototype = inherit(Range.prototype)</p> <p>/* Amb el new davant... -Es crea un nou objecte amb prototipus l'objecte referenciat per la propietat Range.prototype -This farà referència al nou objecte "r", retornat implicitament */</p>	<pre>var foo = function() { this.value = 2 * this.value; }; var obj = { value: 7 } foo.apply(obj) // { value: 14}</pre> <p>/*Podem especificar l'objecte al que farà referència this fent servir els mètodes apply y call d'una funció */</p>

- Una pila amb “closures” i Fibonacci amb “cache”:

Pila fent servir closures	Fibonacci fent servir propietats com a cache
<pre>function StackCr () { var index = 0; var arr = []; return { push: function(val) { arr[index] = val; index++; }, pop: function() { /* arr[-1] = num afegiria '-1' com a propietat de l'objecte arr i les funcions de array (con arr.length no els tindria en compte*/ if (index > 0) index--; }, top: function() { return arr[index - 1]; }, size: function() { return index; }, empty: function() { return index == 0; } } } var pila = StackCr();</pre>	<pre>function factorial(n) { if (!(n in factorial)) factorial[n] = n * factorial(n-1); return factorial[n]; } factorial[1] = 1; /*De manera que aprofitem que les funcions com a tal son objectes, i per tant poden tenir propietats, per a anar guardant els resultats de les crides */</pre>

- Callbacks

Callbacks retornant els resultats d'una crida HTTP
<pre>var display = function() {...} pictureService.getPicture(id) .subscribe((picture) => { console.log('Foto rebuda correctament') display(picture) }, (error) => console.log("S'ha produït un error: " + error))</pre>

Sobre les fonts d'informació

Durant la realització del treball s'han consultat diferents fonts.

La majoria del contingut relacionat amb la història del llenguatge s'ha consultat a la versió en línia del llibre "Speaking JavaScript (An In-Depth Guide for Programmers)", de Axel Rauschmayer. Així mateix s'ha extret alguna dada de l'article de Wikipedia. Això inclou "l'origen", "influències d'altres llenguatges" i "estandarització i versions".

La llista dels paradigmes de programació que es mostra a l'article de Wikipedia m'ha servit de guia per a poder buscar informació sobre cadascun d'aquests. He consultat doncs el mateix article, el llibre mencionat anteriorment, una guia de GitBook anomenada "Professor Frisby's Mostly Adequate Guide to Functional Programming" i per a parlar sobre la programació basada en prototipus he fet servir la documentació de l'assignatura "Conceptes Avançats de Programació", del professor de la FIB Jordi Delgado Pin.

Per a poder respondre a si JavaScript és compilat o interpretat i entendre les característiques del sistema de tipus, he fet servir el llibre anterior i un article d'internet en el que es parla sobre l'engine V8 de Google. En aquest s'adjunta un vídeo d'una conferència d'un dels desenvolupadors d'aquest.

A la descripció general dels usos del llenguatge hi ha informació extreta d'un apartat del llibre. En aquest, es fa un recorregut per etapes fent menció de les tecnologies i aplicacions del llenguatge que han anat sorgint.

Finalment, tots els exemples de codi estan extrets de les transparències de "CAP" de Jordi Delgado.

Bibliografia

Rauschmayer, Alex (2014). "Chapter 1. Basic JavaScript" en *Speaking JavaScript An In-Depth Guide for Programmers*, O'Reilly Media. <<http://speakingjs.com/es5/ch01.html>>

Rauschmayer, Alex (2014). "Chapter 3. The Nature of JavaScript" en *Speaking JavaScript An In-Depth Guide for Programmers*, O'Reilly Media. <<http://speakingjs.com/es5/ch03.html>>

Rauschmayer, Alex (2014). "Chapter 4. How JavaScript Was Created" en *Speaking JavaScript An In-Depth Guide for Programmers*, O'Reilly Media.
<<http://speakingjs.com/es5/ch04.html>>

Rauschmayer, Alex (2014). "Chapter 6. Historical JavaScript Milestones" en *Speaking JavaScript An In-Depth Guide for Programmers*, O'Reilly Media.
<<http://speakingjs.com/es5/ch06.html>>

Rauschmayer, Alex (2014). "Chapter 8. Values" en *Speaking JavaScript An In-Depth Guide for Programmers*, O'Reilly Media. <<http://speakingjs.com/es5/ch08.html>>

Lonsdorf, Brian (2016). *Professor Frisby's Mostly Adequate Guide to Functional Programming* <<https://drboolean.gitbooks.io/mostly-adequate-guide/>> [Consulta: 30 de desembre de 2017]

Delgado Pin, Jordi. *Programació basada en Prototipus* [Consulta: 3 de gener de 2018], Facultat d'Informàtica de Barcelona.

Hinkelmann, Franziska (2017). *Understanding V8's Bytecode* <<https://medium.com/dailyjs/understanding-v8s-bytecode-317d46c94775>> [Consulta: 28 de desembre de 2017]

Simpson, Kyle (2014). *You Don't Know JS: Scope & Closures*, O'Reilly Media
<<https://github.com/getify/You-Dont-Know-JS/blob/master/scope%20%26%20closures/ch1.md>>
[Consulta: 3 de desembre de 2017]

Wikipedia, *JavaScript* <<https://es.wikipedia.org/wiki/JavaScript>> [Consulta: 27 de desembre de 2017]