JavaScript Avanzado

Adolfo Sanz De Diego

Octubre 2015

# Acerca de

## Autor

* **Adolfo Sanz De Diego**
  + Blog: [asanzdiego.blogspot.com.es](http://asanzdiego.blogspot.com.es/)
  + Correo: [asanzdiego@gmail.com](mailto:asanzdiego@gmail.com)
  + GitHub: [github.com/asanzdiego](http://github.com/asanzdiego)
  + Twitter: [twitter.com/asanzdiego](http://twitter.com/asanzdiego)
  + Linkedin: [in/asanzdiego](http://www.linkedin.com/in/asanzdiego)
  + SlideShare: [slideshare.net/asanzdiego](http://www.slideshare.net/asanzdiego/)

## Licencia

* **Este obra está bajo una licencia:**
  + [Creative Commons Reconocimiento-CompartirIgual 3.0](http://creativecommons.org/licenses/by-sa/3.0/es/)
* **El código fuente de los programas están bajo una licencia:**
  + [GPL 3.0](http://www.viti.es/gnu/licenses/gpl.html)

## Ejemplos

* Las slides y los códigos de ejemplo los podéis encontrar en:
  + <https://github.com/asanzdiego/curso-javascript-avanzado-2015>

# JavaScript

## Historia

* Lo crea **Brendan Eich en Netscape en 1995** para hacer páginas web dinámicas
* Aparece por primera vez en Netscape Navigator 2.0
* Cada día más usado (clientes web, videojuegos, windows 8, servidores web, bases de datos, etc.)

## El lenguaje

* Orientado a objetos
* Basado en prototipos
* Funcional
* Débilmente tipado
* Dinámico

# Orientación a objetos

## ¿Qué es un objeto?

* **Colección de propiedades** (pares nombre-valor).
* Todo son objetos (las funciones también) excepto los primitivos: **strings, números, booleans, null o undefined**
* Para saber si es un objeto o un primitivo hacer **typeof variable**

## Propiedades

* Podemos acceder directamente o como si fuese un contenedor:

objeto.nombre === objeto[nombre] // true

* Podemos crearlas y destruirlas en tiempo de ejecución

var objeto = {};  
objeto.nuevaPropiedad = 1; // añadir  
delete objeto.nuevaPropiedad; // eliminar

## Objeto iniciador

* Podemos crear un objeto así:

var objeto = {  
 nombre: "Adolfo",  
 twitter: "@asanzdiego"  
};

## Función constructora

* O con una función constructora y un new.

function Persona(nombre, twitter) {  
 this.nombre = nombre;  
 this.twitter = twitter;  
};  
var objeto = new Persona("Adolfo", "@asanzdiego");

## Prototipos

* Las funciones son objetos y tienen una propiedad llamada **prototype**.
* Cuando creamos un objeto con new, la referencia a esa propiedad **prototype** es almacenada en una propiedad interna.
* El prototipo se utiliza para compartir propiedades.
* Podemos acceder al objeto prototipo de un objeto:

// Falla en Opera o IE <= 8  
Object.getPrototypeOf(objeto);  
  
// No es estandar y falla en IE  
objeto.\_\_proto\_\_;

## Eficiencia

* Si queremos que nuestro código se ejecute una sola vez y que prepare en memoria todo lo necesario para generar objetos, la mejor opción es usar una **función constructora solo con el estado de una nueva instancia, y el resto (los métodos) añadirlos al prototipo**.
* Ejemplo:

function ConstructorA(p1) {  
 this.p1 = p1;  
}  
  
// los métodos los ponenmos en el prototipo  
ConstructorA.prototype.metodo1 = function() {  
 console.log(this.p1);  
};

## Herencia

* Ejemplo:

function ConstructorA(p1) {  
 this.p1 = p1;  
}  
  
function ConstructorB(p1, p2) {  
 // llamamos al super para que no se pierda p1.  
 ConstructorA.call(this, p1);  
 this.p2 = p2;  
}  
  
// Hacemos que B herede de A  
// Prototipo de Función Constructora B apunta al  
// Prototipo de Función Constructora A  
ConstructorB.prototype = Object.create(ConstructorA.prototype);

## Cadena de prototipos

* Cuando se invoca una llamada a una propiedad, **JavaScript primero busca en el propio objeto, y si no lo encuentra busca en su prototipo**, y sino en el prototipo del prototipo, así hasta el prototipo de Object que es null.

## Cadena de prototipos de la instancia

* En el ejemplo anterior:

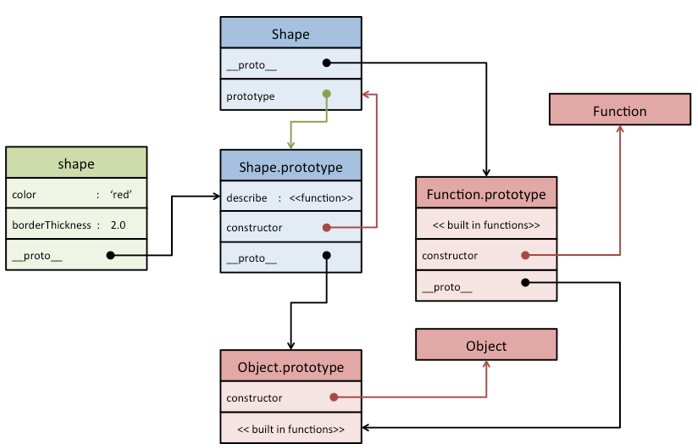
instanciaB.\_\_proto\_\_ == ConstructorB.prototype // true  
instanciaB.\_\_proto\_\_.\_\_proto\_\_ == ConstructorA.prototype // true  
instanciaB.\_\_proto\_\_.\_\_proto\_\_.\_\_proto\_\_ == Object.prototype // true  
instanciaB.\_\_proto\_\_.\_\_proto\_\_.\_\_proto\_\_.\_\_proto\_\_ == null // true

## Cadena de prototipos de la función constructora

* En el ejemplo anterior:

expect(ConstructorB.\_\_proto\_\_).toEqual(Function.prototype);  
expect(ConstructorB.\_\_proto\_\_.\_\_proto\_\_).toEqual(Object.prototype);  
expect(ConstructorB.\_\_proto\_\_.\_\_proto\_\_.\_\_proto\_\_).toEqual(null);

## Esquema prototipos



## Operador instanceof

* La expresión **instanciaB instanceof ConstructorA** devolverá true, si el prototipo de la Función ConstructorA, se encuentra en la cadena de prototipos de la instanciaB.
* En el ejemplo anterior:

instanciaB instanceof ConstructorB; // true  
instanciaB instanceof ConstructorA; // true  
instanciaB instanceof Object; // true

## Extensión

* Con los prototipos podemos extender la funcionalidad del propio lenguaje.
* Ejemplo:

String.prototype.hola = function() {  
 return "Hola "+this;  
}  
  
"Adolfo".hola(); // "Hola Adolfo"

## Propiedades y métodos estáticos

* Lo que se define dentro de la función constructora va a ser propio de la instancia.
* Pero como hemos dicho, en JavaScript, una función es un objeto, al que podemos añadir tanto atributos como funciones.
* **Añadiendo atributos y funciones a la función constructora obtenemos propiedades y métodos estáticos.**
* Ejemplo:

function ConstructorA() {  
  
 ConstructorA.propiedadEstatica = "propiedad estática";  
}  
  
ConstructorA.metodoEstatico = function() {  
 console.log("método estático");  
}

## Propiedades y métodos privados

* La visibilidad de objetos depende del contexto.
* Los contextos en JavaScript son bloques de código entre dos {} y en general, desde uno de ellos, solo tienes acceso a lo que en él se defina y a lo que se defina en otros contextos que contengan al tuyo.
* Ejemplo:

function ConstructorA(privada, publica) {  
 var propiedadPrivada = privada;  
 this.propiedadPublica = publica;  
 var metodoPrivado = function() {  
 console.log("-->propiedadPrivada", propiedadPrivada);  
 }  
 this.metodoPublico = function() {  
 console.log("-->propiedadPublica", this.propiedadPublica);  
 metodoPrivado();  
 }  
}

## Polimorfismo

* Poder llamar a métodos sintácticamente iguales de objetos de tipos diferentes.
* Esto se consigue mediante herencia.

# Técnicas avanzadas

## Funciones

* Son objetos con sus propiedades.
* Se pueden pasar como parámetros a otras funciones.
* Pueden guardarse en variables.
* Son mensajes cuyo receptor es **this**.

## This

* Ejemplo:

var nombre = "Laura";  
  
var alba = {  
 nombre: "Alba",  
 saludo: function() {  
 return "Hola "+this.nombre;  
 }  
}  
  
alba.saludo(); // Hola Alba  
  
var fn = alba.saludo;  
  
fn(); // Hola Laura

## call y apply

* Dos funciones permiten manipular el this: **call y apply** que en lo único que se diferencian es en la llamada.

fn.call(thisArg [, arg1 [, arg2 [...]]])

fn.apply(thisArg [, arglist])

## Número variable de argumentos

* Las funciones en JavaScript aunque tengan especificado un número de argumentos de entrada, **pueden recibir más o menos argumentos** y es válido.

## Arguments

* Es un objeto que **contiene los parámetros** de la función.

function echoArgs() {  
 console.log(arguments[0]); // Adolfo  
 console.log(arguments[1]); // Sanz  
}  
echoArgs("Adolfo", "Sanz");

## Declaración de funciones

* Estas 2 declaraciones son **equivalentes**:

function holaMundo1() {  
 console.log("Hola Mundo 1");  
}  
holaMundo1();  
  
var holaMundo2 = function() {  
 console.log("Hola Mundo 2");  
}  
holaMundo2();

## Transfiriendo funciones a otras funciones

* Hemos dicho que las funciones son objetos, así que **se pueden pasar como parámetros**.

function saluda() {  
 console.log("Hola")  
}  
function ejecuta(func) {  
 func()  
}  
ejecuta(saluda);

## Funciones anónimas

* Hemos dicho que las funciones se pueden declarar.
* Pero también **podemos no declararlas y dejarlas como anónimas**.
* Una función anónima así declarada **no se podría ejecutar**.

function(nombre) {  
 console.log("Hola "+nombre);  
}

* Pero **una función puede devolver una función anónima**.

function saludador(nombre) {  
 return function() {  
 console.log("Hola "+nombre);  
 }  
}  
  
var saluda = saludador("mundo");  
saluda(); // Hola mundo

## Funciones autoejecutables

* Podemos autoejecutar funciones anónimas.

(function(nombre) {  
 console.log("Hola "+nombre);  
})("mundo")

## Clousures

* Un closure **combina una función y el entorno en que se creó**.

function creaSumador(x) {  
 return function(y) {  
 return x + y;  
 };  
}  
  
var suma5 = creaSumador(5);  
var suma10 = creaSumador(10);  
  
console.log(suma5(2)); // muestra 7  
console.log(suma10(2)); // muestra 12

* En una closures la función interna almacena una **referencia al último valor** de la variable establecido cuando la función externa termina de ejecutarse.

## El patrón Modulo

* Se trata de una función que actúa como contenedor para un contexto de ejecución.

miModulo = (function() {  
  
 var propiedadPrivada;  
  
 function metodoPrivado() { };  
  
 // API publica  
 return {  
 metodoPublico1: function () {  
 },  
  
 metodoPublico2: function () {  
 }  
 }  
}());

## Eficiencia

* Si se ejecuta desde el navegador, **se suele pasar como parámetro el objeto window para mejorar el rendimiento**. Así cada vez que lo necesitemos el intérprete lo utilizará directamete en lugar de buscarlo remontando niveles.
* Y también **se suele pasar el parámetro undefined, para evitar los errores que pueden darse si la palabra reservada ha sido reescrita** en alguna parte del código y su valor no corresponda con el esperado.

miModulo = (function(window, undefined) {  
  
 // El código va aquí  
  
})( window );

## El patrón Modulo Revelado

* El problema del patrón Modulo es pasar un método de privado a público o viceversa.
* Por ese motivo lo que que se suele hacer es definir todo en el cuerpo, y luego **referenciar solo los públicos en el bloque return**.

miModulo = (function() {  
  
 function metodoA() { };  
  
 function metodoB() { };  
  
 function metodoC() { };  
  
 // API publica  
 return {  
 metodoPublico1: metodoA,  
 metodoPublico2: metodoB  
 }  
}());

## Espacios de nombres

* Para simular espacios de nombres, en JavaScript se anidan objetos.

miBiblioteca = miBiblioteca || {};  
  
miBiblioteca.seccion1 = miBiblioteca.seccion1 || {};  
  
miBiblioteca.seccion1 = {  
 priopiedad: p1,  
 metodo: function() { },  
};  
  
miBiblioteca.seccion2 = miBiblioteca.seccion2 || {};  
  
miBiblioteca.seccion2 = {  
 priopiedad: p2,  
 metodo: function() { },  
};

* Se puede combinar lo anterior con módulos autoejecutables:

miBiblioteca = miBiblioteca || {};  
  
(function(namespace) {  
  
 var propiedadPrivada = p1;  
  
 namespace.propiedadPublica = p2;  
  
 var metodoPrivado = function() { };  
  
 namespace.metodoPublico = function() { };  
  
}(miBiblioteca));

# Document Object Model

## ¿Qué es DOM?

* Acrónimo de **Document Object Model**
* Es un conjunto de utilidades específicamente diseñadas para **manipular documentos XML, y por extensión documentos XHTML y HTML**.
* DOM transforma internamente el archivo XML en una estructura más fácil de manejar formada por una jerarquía de nodos.

## Tipos de nodos

* Los más importantes son:
  + **Document**: representa el nodo raíz.
  + **Element**: representa el contenido definido por un par de etiquetas de apertura y cierre y puede tener tanto nodos hijos como atributos.
  + **Attr**: representa el atrributo de un elemento.
  + **Text**: almacena el contenido del texto que se encuentra entre una etiqueta de apertura y una de cierre.

## Recorrer el DOM

* JavaScript proporciona **funciones** para recorrer los nodos:

getElementById(id)  
getElementsByName(name)  
getElementsByTagName(tagname)  
getElementsByClassName(className)  
getAttribute(attributeName)  
querySelector(selector)  
querySelectorAll(selector)

## Manipular el DOM

* JavaScript proporciona **funciones** para la manipulación de nodos:

createElement(tagName)  
createTextNode(text)  
createAttribute(attributeName)  
appendChild(node)  
insertBefore(newElement, targetElement)  
removeAttribute(attributename)  
removeChild(childreference)  
replaceChild(newChild, oldChild)

## Propiedades Nodos

* Los nodos tienen algunas **propiedades** muy útiles:

attributes[]  
className  
id  
innerHTML  
nodeName  
nodeValue  
style  
tabIndex  
tagName  
title

* Los nodos tienen algunas **propiedades** muy útiles:

childNodes[]  
firstChild  
lastChild  
previousSibling  
nextSibling  
ownerDocument  
parentNode

# Librerías y Frameworks

## jQuery

* [jQuery](https://jquery.com/): libreria que reduce código ("write less, do more").

// Vanilla JavaScript  
var elem = document.getElementById("miElemento");  
  
//jQuery  
var elem = $("#miElemento");

## jQuery UI & Mobile

* [jQuery UI](http://jqueryui.com/): diseño interfaces gráficas.
* [jQuery Mobile](https://jquerymobile.com/): versión adaptada para móviles (eventos y tamaño).

## Frameworks CSS

* [Bootstrap](http://getbootstrap.com/) y [Foundation](http://foundation.zurb.com/).
* Fácil maquetación, sistema rejilla, clases CSS, temas, etc.

## MVC en el front

* [BackboneJS](http://backbonejs.org/): ligero y flexible.
* [EmberJS](http://emberjs.com/): "Convention over Configuration", muy popular entre desarrolladores [Ruby on Rails](http://rubyonrails.org/).
* [AngularJS](http://angularjs.org/) extiende etiquetas HML (g-app, ng-controller, ng-model, ng-view), detrás está Google, tiene gran popularidad, abrupta curva de aprendizaje.

## NodeJS

* [NodeJS](http://nodejs.org/) permite ejecutar JS fuera del navegador.
* Viene con su propio gestor de paquetes: [npm](https://www.npmjs.com/)

## Automatización de tareas

* [GruntJS](http://gruntjs.com/): más popularidad y más plugins.
* [GulpJS](http://gulpjs.com/): más rápido tanto al escribir ("Code over Configure") como al ejecutar (streams).

## Gestión de dependencias

* [Bower](http://bower.io/): para el lado cliente. Puede trabajar con repositorios Git.
* [Browserify](http://browserify.org/): permite escribir módulos como en [NodeJS](http://nodejs.org/) y compilarlos para que se puedan usar en el navegador.
* [RequeriJS](http://requirejs.org/): las dependencias se cargan de forma asíncrona y solo cuando se necesitam.
* [WebPack](https://webpack.github.io/): es un empaquetador de módulos

## Aplicaciones de escritorio multiplataforma

* [AppJS](http://appjs.com/), y su fork [DeskShell](http://deskshell.org/): los más antiguos, un poco abandonados.
* [NW.js](http://nwjs.io/): opción más popular y madura hoy en día.
* [Electron](http://electron.atom.io/): creada para el [editor Atom de GitHub](https://atom.io/): está creciendo en popularidad.

## Aplicaciones móviles híbridas

* [cordova](https://cordova.apache.org/): una de los primeros. Hoy en día, otros frameworks se basan en él.
* [ionic](http://ionicframework.com/): utiliza AngularJS, tiene una CLI, muy popular.
* [React Native](https://facebook.github.io/react-native/): recién liberado por facebook.

## WebComponents

* [WebComponents](http://www.w3.org/wiki/WebComponents/) es una especificación de la W3C para permitir crear componentes y reutilizarlos.
* [polymer](https://www.polymer-project.org/): proyecto de Google para poder empezar a usar los WebComponents en todos los navegadores.

## Otros

* [React](https://facebook.github.io/react/): librería hecho por Facebook para crear interfaces que se renderizan muy rápido, ya sea en cliente o servidor.
* [Flux](https://facebook.github.io/flux/): framework hecho por Facebook que utiliza React.
* [Meteor](https://www.meteor.com/): es una plataforma que permite desarrollar aplicaciones real-time con JS Isomófico (se ejecuta en front y back)

# Eventos

## Principales eventos

|  |  |
| --- | --- |
| Evento | Descripción |
| onblur | Un elemento pierde el foco |
| onchange | Un elemento ha sido modificado |
| onclick | Pulsar y soltar el ratón |
| ondblclick | Pulsar dos veces seguidas con el ratón |

|  |  |
| --- | --- |
| Evento | Descripción |
| onfocus | Un elemento obtiene el foco |
| onkeydown | Pulsar una tecla y no soltarla |
| onkeypress | Pulsar una tecla |
| onkeyup | Soltar una tecla pulsada |
| onload | Página cargada completamente |

|  |  |
| --- | --- |
| Evento | Descripción |
| onmousedown | Pulsar un botón del ratón y no soltarlo |
| onmousemove | Mover el ratón |
| onmouseout | El ratón "sale" del elemento |
| onmouseover | El ratón "entra" en el elemento |
| onmouseup | Soltar el botón del ratón |

|  |  |
| --- | --- |
| Evento | Descripción |
| onreset | Inicializar el formulario |
| onresize | Modificar el tamaño de la ventana |
| onselect | Seleccionar un texto |
| onsubmit | Enviar el formulario |
| onunload | Se abandona la página |

## Suscripción

* Para añadir o eliminar un **Listener** de un evento a un elemento:

var windowOnLoad = function(e) {  
 console.log('window:load', e);  
};  
  
window.addEventListener('load', windowOnLoad);  
  
window.removeEventListener('load', windowOnLoad);

## Eventos personalizados

* Podemos crear **eventos personalizados**:

var event = new Event('build');  
  
elem.addEventListener('build', function (e) { ... }, false);

* Podemos crear **eventos personalizados con datos**:

var event = new CustomEvent('build', { 'detail': detail });  
  
elem.addEventListener('build', function (e) {  
 log('The time is: ' + e.detail);  
}, false);

## Disparar un evento

* Podemos **disparar** eventos:

function simulateClick() {  
 var event = new MouseEvent('click');  
 var element = document.getElementById('id');  
 element.dispatchEvent(event);  
}

## Propagación

## El patrón PubSub

var EventBus = {  
 topics: {},  
  
 subscribe: function(topic, listener) {  
 if (!this.topics[topic]) this.topics[topic] = [];  
 this.topics[topic].push(listener);  
 },  
  
 publish: function(topic, data) {  
 if (!this.topics[topic] || this.topics[topic].length < 1) return;  
 this.topics[topic].forEach(function(listener) {  
 listener(data || {});  
 });  
 }  
};

EventBus.subscribe('foo', alert);  
EventBus.publish('foo', 'Hello World!');

var Mailer = function() {  
 EventBus.subscribe('order/new', this.sendPurchaseEmail);  
};  
  
Mailer.prototype = {  
 sendPurchaseEmail: function(userEmail) {  
 console.log("Sent email to " + userEmail);  
 }  
};  
  
var Order = function(params) {  
 this.params = params;  
};  
  
Order.prototype = {  
 saveOrder: function() {  
 EventBus.publish('order/new', this.params.userEmail);  
 }  
};

var mailer = new Mailer();  
var order = new Order({userEmail: 'john@gmail.com'});  
order.saveOrder();  
"Sent email to john@gmail.com"

## WebSockets

# AJAX

## JSON, JSONP, CORS

## Uso de APIs

# Inyección de dependencias

## AMD (RequireJS)

## CommonJS (Browserify)

# ECMAScript6

## Principales Novedades

## Como usarlo hoy

# Enlaces

## General (ES)

* <http://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Obsolete_Pages/Gu%C3%ADa_JavaScript_1.5>
* <http://cevichejs.com/>
* <http://www.arkaitzgarro.com/javascript/>
* <http://www.etnassoft.com/category/javascript/>

## General (EN)

* <http://www.javascriptkit.com/>
* <http://javascript.info/>
* <http://www.howtocreate.co.uk/tutorials/javascript/>

## Orientación Objetos (ES)

* <http://www.programania.net/diseno-de-software/entendiendo-los-prototipos-en-javascript/>
* <http://www.programania.net/diseno-de-software/creacion-de-objetos-eficiente-en-javascript/>
* <http://blog.amatiasq.com/2012/01/javascript-conceptos-basicos-herencia-por-prototipos/>
* <http://albertovilches.com/profundizando-en-javascript-parte-1-funciones-para-todo>
* <http://albertovilches.com/profundizando-en-javascript-parte-2-objetos-prototipos-herencia-y-namespaces>
* <http://www.arkaitzgarro.com/javascript/capitulo-9.html>
* <http://www.etnassoft.com/2011/04/15/concepto-de-herencia-prototipica-en-javascript/>

## Orientación Objetos (EN)

* <http://www.codeproject.com/Articles/687093/Understanding-JavaScript-Object-Creation-Patterns>
* <http://javascript.info/tutorial/object-oriented-programming>
* <http://www.howtocreate.co.uk/tutorials/javascript/objects>

## Técnicas avanzadas (ES)

* <http://www.etnassoft.com/2011/03/14/funciones-autoejecutables-en-javascript/>
* <http://www.etnassoft.com/2012/01/12/el-valor-de-this-en-javascript-como-manejarlo-correctamente/>
* <https://developer.mozilla.org/es/docs/Web/JavaScript/Closures>
* <http://www.variablenotfound.com/2012/10/closures-en-javascript-entiendelos-de.html>
* <http://www.webanalyst.es/espacios-de-nombres-en-javascript/>
* <http://www.etnassoft.com/2011/04/11/el-patron-de-modulo-en-javascript-en-profundidad/>
* <http://www.etnassoft.com/2011/04/18/ampliando-patron-modulo-javascript-submodulos/>
* <http://notasjs.blogspot.com.es/2012/04/el-patron-modulo-en-javascript.html>

## DOM (ES)

* <http://cevichejs.com/3-dom-cssom#dom>
* <http://www.arkaitzgarro.com/javascript/capitulo-13.html>

## DOM (EN)

* <http://www.javascriptkit.com/domref/>
* <http://javascript.info/tutorial/dom>

## Frameworks (ES)

* <https://carlosazaustre.es/blog/frameworks-de-javascript/>
* <https://docs.google.com/drawings/d/1bhe9-kxhhGvWU0LsB7LlJfMurP3DGCIuUOmqEOklzaQ/edit>
* <http://www.lostiemposcambian.com/blog/javascript/backbone-vs-angular-vs-ember/>
* <http://blog.koalite.com/2015/06/grunt-o-gulp-que-uso/>

## Frameworks (EN)

* <http://www.slideshare.net/deepusnath/javascript-frameworks-comparison-angular-knockout-ember-and-backbone>
* <http://stackshare.io/stackups/backbone-vs-emberjs-vs-angularjs>
* <http://www.hongkiat.com/blog/gulp-vs-grunt/>
* <https://mattdesl.svbtle.com/browserify-vs-webpack>
* <http://hackhat.com/p/110/module-loader-webpack-vs-requirejs-vs-browserify/>
* <http://devzum.com/2014/02/10-best-node-js-mvc-frameworks-for-javascript-developers/>
* <http://www.tivix.com/blog/nwjs-and-electronjs-web-technology-desktop/>
* <http://stackshare.io/stackups/phonegap-vs-ionic-vs-react-native>
* <https://developer.salesforce.com/page/Native,_HTML5,_or_Hybrid:_Understanding_Your_Mobile_Application_Development_Options>

## Eventos (ES)

* <http://cevichejs.com/3-dom-cssom#eventos>
* <http://www.arkaitzgarro.com/javascript/capitulo-15.html>
* <http://codexexempla.org/curso/curso_4_3_e.php>

## Eventos (EN)

* <https://developer.mozilla.org/en-US/docs/Web/API/EventTarget>
* <https://developer.mozilla.org/en-US/docs/Web/API/Event>
* <http://dev.housetrip.com/2014/09/15/decoupling-javascript-apps-using-pub-sub-pattern/>
* <https://stackoverflow.com/questions/5963669/whats-the-difference-between-event-stoppropagation-and-event-preventdefault>

## ES6 (ES)

* <http://rlbisbe.net/2014/08/26/articulo-invitado-ecmascript-6-y-la-nueva-era-de-javascript-por-ckgrafico/>
* <http://carlosazaustre.es/blog/ecmascript-6-el-nuevo-estandar-de-javascript/>
* <http://asanzdiego.blogspot.com.es/2015/06/principios-solid-con-ecmascript-6-el-nuevo-estandar-de-javascript.html>

## ES6 (EN)

* <http://es6-features.org/>
* <http://kangax.github.io/compat-table/es5/>