JavaScript Avanzado

Adolfo Sanz De Diego

Octubre 2015

1 Acerca de

1.1 Autor

- **Adolfo Sanz De Diego**
  - Blog: asanzdiego.blogspot.com.es
  - Correo: asanzdiego@gmail.com
  - GitHub: github.com/asanzdiego
  - Twitter: twitter.com/asanzdiego
  - Linkedin: in/asanzdiego
  - SlideShare: slideshare.net/asanzdiego

1.2 Licencia

- **Este obra está bajo una licencia:**
  - Creative Commons Reconocimiento-CompartirIgual 3.0
- **El código fuente de los programas están bajo una licencia:**
  - GPL 3.0

1.3 Ejemplos

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

2 JavaScript

2.1 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.)

2.2 El lenguaje

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

3 Orientación a objetos

3.1 ¿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**

3.2 Propiedades (I)

- Podemos acceder directamente o como si fuese un contenedor:

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

3.3 Propiedades (II)

- Podemos crearlas y destruirlas en tiempo de ejecución

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

3.4 Objeto iniciador

- Podemos crear un objeto así:

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

3.5 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");
```

3.6 Prototipos (I)

- 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.

3.7 Prototipos (II)

- 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__;
```

3.8 Eficiencia (I)

- 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**.

3.9 Eficiencia (II)

- Ejemplo:

```
function ConstructorA(p1) {
  this.p1 = p1;
}

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

3.10 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);
```

3.11 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.

3.12 Cadena de prototipos de la instancia

- En el ejemplo anterior:

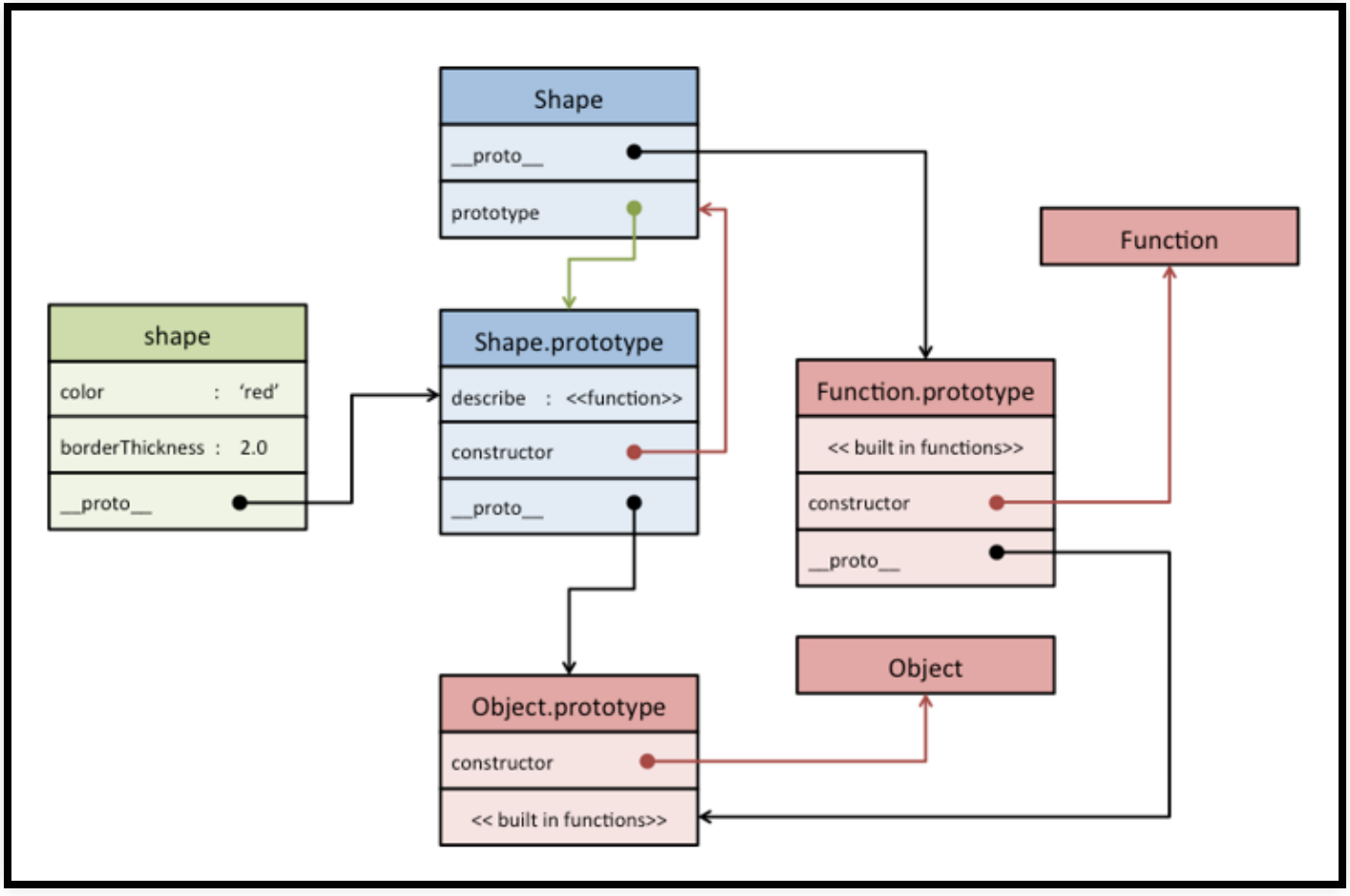
```
instanciaB.__proto__ == ConstructorB.prototype // true
instanciaB.__proto__.__proto__ == ConstructorA.prototype // true
instanciaB.__proto__.__proto__.__proto__ == Object.prototype // true
instanciaB.__proto__.__proto__.__proto__.__proto__ == null // true
```

3.13 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);
```

3.14 Esquema prototipos



Esquema prototipos

3.15 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
```

3.16 Extensión

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

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

"Adolfo".hola(); // "Hola Adolfo"
```

3.17 Propiedades y métodos estáticos (I)

- 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.**

3.18 Propiedades y métodos estáticos (II)

- Ejemplo:

```
function ConstructorA() {

  ConstructorA.propiedadEstatica = "propiedad estática";
}

ConstructorA.metodoEstatico = function() {
  console.log("método estático");
}
```

3.19 Propiedades y métodos privados (I)

- 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.

3.20 Propiedades y métodos privados (II)

- 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();
  }
}
```

3.21 Polimorfismo

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

4 Técnicas avanzadas

4.1 Funciones

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

4.2 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
```

4.3 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])
```

4.4 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.

4.5 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");
```

4.6 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();
```

4.7 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);
```

4.8 Funciones anónimas (I)

- Hemos dicho que las funciones se pueden declarar.
- Pero también **podemos no declararlas y dejarlas como anónimas**.

4.9 Funciones anónimas (II)

- Una función anónima así declarada **no se podría ejecutar**.

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

4.10 Funciones anónimas (III)

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

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

var saluda = saludador("mundo");
saluda(); // Hola mundo
```

4.11 Funciones autoejecutables

- Podemos autoejecutar funciones anónimas.

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

4.12 Clousures (I)

- 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
```

4.13 Clousures (II)

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

4.14 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 () {
    }
  }
}());
```

4.15 Eficiencia (I)

- 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.

4.16 Eficiencia (II)

```
miModulo = (function(window, undefined) {

  // El código va aquí

})( window );
```

4.17 El patrón Modulo Revelado (I)

- 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**.

4.18 El patrón Modulo Revelado (II)

```
miModulo = (function() {

  function metodoA() { };

  function metodoB() { };

  function metodoC() { };

  // API publica
  return {
    metodoPublico1: metodoA,
    metodoPublico2: metodoB
  }
}());
```

4.19 Espacios de nombres (I)

- 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() { },
};
```

4.20 Espacios de nombres (II)

- 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));
```

5 Document Object Model

5.1 ¿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.

5.2 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.

5.3 Recorrer el DOM

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

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

5.4 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)
```

5.5 Propiedades Nodos (I)

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

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

5.6 Propiedades Nodos (II)

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

```
childNodes[]
firstChild
lastChild
previousSibling
nextSibling
ownerDocument
parentNode
```

6 Librerías y Frameworks

6.1 jQuery

- jQuery: libreria que reduce código ("write less, do more").

```
// Vanilla JavaScript
var elem = document.getElementById("miElemento");

//jQuery
var elem = $("#miElemento");
```

6.2 jQuery UI & Mobile

- jQuery UI: diseño interfaces gráficas.
- jQuery Mobile: versión adaptada para móviles (eventos y tamaño).

6.3 Frameworks CSS

- Bootstrap y Foundation.
- Fácil maquetación, sistema rejilla, clases CSS, temas, etc.

6.4 MVC en el front

- BackboneJS: ligero y flexible.
- EmberJS: "Convention over Configuration", muy popular entre desarrolladores Ruby on Rails.
- AngularJS extiende etiquetas HML (g-app, ng-controller, ng-model, ng-view), detrás está Google, tiene gran popularidad, abrupta curva de aprendizaje.

6.5 NodeJS

- NodeJS permite ejecutar JS fuera del navegador.
- Viene con su propio gestor de paquetes: npm

6.6 Automatización de tareas

- GruntJS: más popularidad y más plugins.
- GulpJS: más rápido tanto al escribir ("Code over Configure") como al ejecutar (streams).

6.7 Gestión de dependencias

- Bower: para el lado cliente. Puede trabajar con repositorios Git.
- Browserify: permite escribir módulos como en NodeJS y compilarlos para que se puedan usar en el navegador.
- RequeriJS: las dependencias se cargan de forma asíncrona y solo cuando se necesitam.
- WebPack: es un empaquetador de módulos

6.8 Aplicaciones de escritorio multiplataforma

- AppJS, y su fork DeskShell: los más antiguos, un poco abandonados.
- NW.js: opción más popular y madura hoy en día.
- Electron: creada para el editor Atom de GitHub: está creciendo en popularidad.

6.9 Aplicaciones móviles híbridas

- cordova: una de los primeros. Hoy en día, otros frameworks se basan en él.
- ionic: utiliza AngularJS, tiene una CLI, muy popular.
- React Native: recién liberado por facebook.

6.10 WebComponents

- WebComponents es una especificación de la W3C para permitir crear componentes y reutilizarlos.
- polymer: proyecto de Google para poder empezar a usar los WebComponents en todos los navegadores.

6.11 Otros

- React: librería hecho por Facebook para crear interfaces que se renderizan muy rápido, ya sea en cliente o servidor.
- Flux: framework hecho por Facebook que utiliza React.
- Meteor: es una plataforma que permite desarrollar aplicaciones real-time con JS Isomófico (se ejecuta en front y back)

7 Eventos

7.1 El patrón PubSub (I)

```
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 || {});
    });
  }
};
```

7.2 El patrón PubSub (II)

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

7.3 El patrón PubSub (III)

```
var Mailer = function() {
  EventBus.subscribe('order/new', this.sendPurchaseEmail);
};

Mailer.prototype = {
  sendPurchaseEmail: function(userEmail) {
    console.log("Sent email to " + userEmail);
  }
};
```

7.4 El patrón PubSub (IV)

```
var Order = function(params) {
  this.params = params;
};

Order.prototype = {
  saveOrder: function() {
    EventBus.publish('order/new', this.params.userEmail);
  }
};
```

7.5 El patrón PubSub (V)

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

7.6 Principales eventos (I)

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

7.7 Principales eventos (II)

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

7.8 Principales eventos (III)

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

7.9 Principales eventos (IV)

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

7.10 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);
```

7.11 Eventos personalizados (I)

- Podemos crear **eventos personalizados**:

```
var event = new Event('build');

elem.addEventListener('build', function (e) { ... }, false);
```

7.12 Eventos personalizados (II)

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

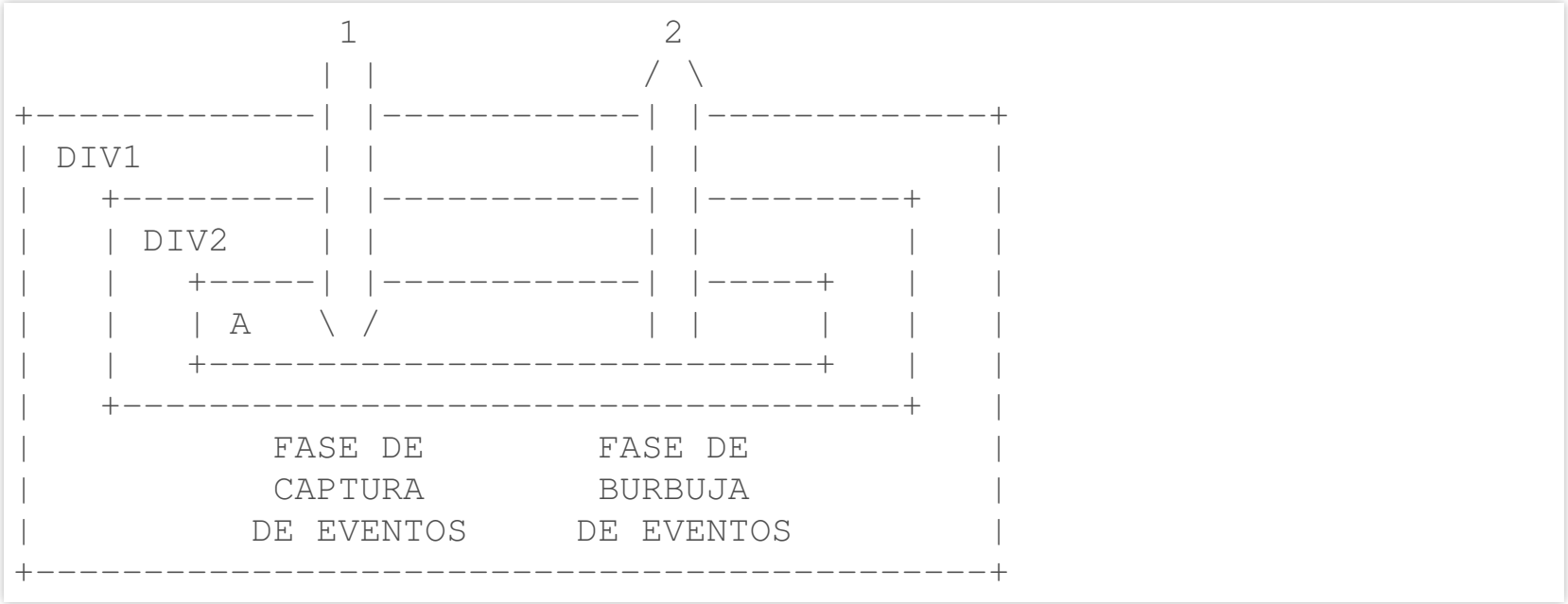
```
var event = new CustomEvent('build', { 'detail': detail });

elem.addEventListener('build', function (e)  {
  log('The time is: ' + e.detail);
}, false);
```

7.13 Disparar un evento

- Podemos **disparar** eventos:

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

7.14 Propagación (I)



7.15 Propagación (II)

```
// en fase de CAPTURA
addEventListener("eventName",callback, true);

// en fase de BURBUJA
addEventListener("eventName",callback, false); // por defecto
```

7.16 Propagación (III)

```
// detiene la propagación del evento
event.stopPropagation();

// elimina las acciones por defecto (ejemplo: abrir enlace)
event.preventDefault();
```

8 WebSockets

8.1 ¿Qué son los WebSockets?

- Nos permiten **comunicación bidireccional** entre cliente y servidor.

8.2 Socket.IO

- Librería cliente y servidor (NodeJS) para utilizar WebSockets:
- Simplifica la API.
- Permite envíar no sólo texto.
- Permite crear eventos propios.
- Permite utilizar navegadores sin soporte de WebSockets.

9 AJAX

9.1 ¿Qué es AJAX?

- Acrónimo de **Asynchronous JavaScript And XML**.
- Técnica para crear **aplicaciones web interactivas** o RIA (Rich Internet Applications).
- Estas aplicaciones se ejecutan en el cliente, es decir, en el navegador de los usuarios.
- Mientras se mantiene la comunicación asíncrona con el servidor en segundo plano.
- De esta forma es posible realizar **cambios sobre las páginas sin necesidad de recargarlas**.

9.2 Tecnologías AJAX

- AJAX no es una tecnología en sí misma, en realidad, se trata de varias tecnologías independientes que se unen de formas nuevas y sorprendentes.
- Las tecnologías que forman AJAX son:
  - **XHTML y CSS**, como estándares de presentación.
  - **DOM**, para la manipulación dinámica de la presentación.
  - **XML, JSON y otros**, para la la manipulación de información.
  - **XMLHttpRequest**, para el intercambio asíncrono de información.
  - **JavaScript**, para unir todas las demás tecnologías.

9.3 ¿Qué es el XMLHttpRequest?

- El intercambio de datos AJAX entre cliente y servidor se hace mediante el objeto XMLHttpRequest, disponible en los navegadores actuales.
- **No es necesario que el contenido esté formateado en XML**.
- Su manejo puede llegar a ser complejo, aunque librerías como **jQuery** facilitan enormemente su uso.

9.4 Ejemplo

```
var http_request = new XMLHttpRequest();
var url = "http://example.net/jsondata.php";

// Descarga los datos JSON del servidor.
http_request.onreadystatechange = handle_json;
http_request.open("GET", url, true);
http_request.send(null);

function handle_json() {
  if (http_request.status == 200) {
    var json_data = http_request.responseText;
    var the_object = eval("(" + json_data + ")");
  } else {
    alert("Ocurrio un problema con la URL.");
  }
}
```

10 JSON

10.1 ¿Qué es JSON?

- Acrónimo de **JavaScript Object Notation**.
- Es un subconjunto de la notación literal de objetos de JavaScript.
- Sirve como formato ligero para el intercambio de datos.
- **Su simplicidad ha generalizado su uso, especialmente como alternativa a XML en AJAX**.
- En JavaScript, un texto JSON se puede analizar fácilmente usando la **función eval()**.

10.2 Parse

```
miObjeto = eval('(' + json_datos + ')');
```

- Eval es muy rápido, pero como compila y ejecuta cualquier código JavaScript, las consideraciones de seguridad recomiendan no usarlo.
- Lo recomendable usar las librerías de JSON.org:
  - JSON in JavaScript - Explanation
  - JSON in JavaScript - Downloads

10.3 Ejemplo

```
{
    curso: "AJAX y jQuery",
    profesor: "Adolfo",
    participantes: [
        { nombre: "Isabel", edad: 35 },
        { nombre: "Alba", edad: 15 },
        { nombre: "Laura", edad: 10 }
    ]
}
```

10.4 JSONP

- Por seguridad XMLHttpRequest sólo puede realizar peticiones al mismo dominio.
- JSONP envuelve el JSON en una función definida por el cliente.
- Esto nos permite hacer peticiones GET (sólo GET) a dominios distintos.

10.5 CORS (I)

- Protocolo Cross-Origin Resource Sharing (Compartición de recursos de distintos orígenes).
- Realizar peticiones a otros dominios siempre y cuando el dominio de destino esté de acuerdo en recibir peticiones del dominio de origen.
- Tanto navegador como servidor tienen que implementar el protocolo.

10.6 CORS (II)

- Desde el servidor, se envía en cabecera:

```
Access-Control-Allow-Origin: http://dominio-permitido.com
```

11 APIs REST

11.1 ¿Qué es un API REST?

- REST (Representational State Transfer) es una técnica de arquitectura software para sistemas hipermedia distribuidos como la World Wide Web.
- Es decir, una URL (Uniform Resource Locator) **representa un recurso** al que se puede acceder o modificar mediante los métodos del protocolo HTTP (POST, GET, PUT, DELETE).

11.2 ¿Por qué REST?

- Es **más sencillo** (tanto la API como la implementación).
- Es **más rápido** (peticiones más ligeras que se puede cachear).
- Es **multiformato** (HTML, XML, JSON, etc.).
- Se complementa muy bien con **AJAX**.

11.3 Ejemplo API

- **GET** a http://myhost.com/person
  - Devuelve todas las personas
- **POST** a http://myhost.com/person
  - Crear una nueva persona
- **GET** a http://myhost.com/person/123
  - Devuelve la persona con id=123
- **PUT** a http://myhost.com/person/123
  - Actualiza la persona con id=123
- **DELETE** a http://myhost.com/person/123
  - Borra la persona con id=123

11.4 Errores HTTP

- 200 OK
- 201 Created
- 202 Accepted
- 301 Moved Permanently
- 400 Bad Request
- 401 Unauthorised
- 402 Payment Required
- 403 Forbidden
- 404 Not Found
- 405 Method Not Allowed
- 500 Internal Server Error
- 501 Not Implemented

12 Gestión de dependencias

12.1 AMD

- Definición de Módulos Asíncronos (AMD)
- La implementación más popular de este estándar es RequireJS.
- Sintaxis un poco complicada.
- Permite la carga de módulos de forma asíncrona.
- Se usa principalmente en navegadores.

12.2 RequireJS (I)

- index.html

```
<!DOCTYPE html>
<html>
    <head>
        <title>Page 1</title>
        <script data-main="js/index" src="js/lib/require.js"></script>
    </head>
    <body>
        <h1>Hola Mundo</h1>
    </body>
</html>
```

12.3 RequireJS (II)

- js/index.js

```
requirejs(['./common'], function (common) {
    requirejs(['app/main']);
});
```

12.4 RequireJS (III)

- app/main.js

```
define(function (require) {
    var $ = require('jquery');
    var persona = require('./persona');

    $('h1').html("Hola requery.js");

    var p = new persona("Adolfo", 30);
    p.saludar();
});
```

12.5 RequireJS (IV)

- app/persona.js

```
define(function () {

  var Persona = function(nombre, edad) {

      this.nombre = nombre;

      Persona.prototype.saludar = function() {
          alert("Hola, mi nombre es " + this.nombre);
      };
  }

  return Persona;
});
```

12.6 CommonJS

- La implementación usada en NodeJS y Browserify.
- Sintaxis sencilla.
- Carga los módulos de forma síncrona.
- Se usa principalmente en el servidor.

12.7 Browserify (I)

- Instalar browserify

```
npm install -g browserify
```

12.8 Browserify (II)

- Instalar dependencias de **package.json**

```
npm install
```

12.9 Browserify (III)

- **package.json**

```
{
  "name": "browserify-example",
  "version": "1.0.0",
  "dependencies": {
    "jquery": "^2.1.3"
  }
}
```

12.10 Browserify (IV)

- Compilar las dependencias a **bundle.js**

```
browserify js/main.js -o js/bundle.js
```

12.11 Browserify (V)

- index.html

```
<!doctype html>
<html>
  <head>
    <title>Browserify Playground</title>
  </head>
  <body>
    <h1>Hola Mundo</h1>
    <script src="js/bundle.js"></script>
  </body>
</html>
```

12.12 Browserify (VI)

- js/app/main.js

```
var $ = require('jquery');
var persona = require('./persona');

$('h1').html('Hola Browserify');

var p = new persona("Adolfo", 30);
p.saludar();
```

12.13 Browserify (VII)

- js/app/persona.js

```
var Persona = function(nombre, edad) {

    this.nombre = nombre;

    Persona.prototype.saludar = function() {
        alert("Hola, mi nombre es " + this.nombre);
    };
}

module.exports = Persona;
```

12.14 ECMAScript 6

- Coje lo mejor de los 2 enfoques:
  - Similitudes con **CommonJS**: sintaxis sencilla.
  - Similitudes con **AMD**: soporte para carga asíncrona.

13 ES6

13.1 Como usarlo hoy

- Babel nos permite utilizar ES6 hoy en día.

13.2 Función Arrow (I)

```
// ES5
var data = [{...}, {...}, {...}, ...];
data.forEach(function(elem){
    console.log(elem)
});
```

13.3 Función Arrow (I)

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

13.4 Función Arrow (III)

```
// ES5
var miFuncion = function(num1, num2) {
    return num1 + num2;
}
```

13.5 Función Arrow (IV)

```
// ES6
var miFuncion = (num1, num2) => num1 + num2;
```

13.6 This (I)

```
//ES5
var objEJ5 = {
  data : ["Adolfo", "Isabel", "Alba"],
  duplicar : function() {
    var that = this;
    this.data.forEach(function(elem){
        that.data.push(elem);
    });
    return this.data;
  }
}
```

13.7 This (II)

```
//ES6
var objEJ6 = {
  data : ["Adolfo", "Isabel", "Alba"],
  duplicar : function() {
    this.data.forEach((elem) => {
        this.data.push(elem);
    });
    return this.data;
  }
}
```

13.8 Definición de Clases (I)

```
//ES5
var Shape = function (id, x, y) {
    this.id = id;
    this.move(x, y);
};
Shape.prototype.move = function (x, y) {
    this.x = x;
    this.y = y;
};
```

13.9 Definición de Clases (II)

```
//ES6
class Shape {
    constructor (id, x, y) {
        this.id = id
        this.move(x, y)
    }
    move (x, y) {
        this.x = x
        this.y = y
    }
}
```

13.10 Herencia de Clases (I)

```
//ES5
var Rectangle = function (id, x, y, width, height) {
    Shape.call(this, id, x, y);
    this.width  = width;
    this.height = height;
};
Rectangle.prototype = Object.create(Shape.prototype);
Rectangle.prototype.constructor = Rectangle;

var Circle = function (id, x, y, radius) {
    Shape.call(this, id, x, y);
    this.radius = radius;
};
Circle.prototype = Object.create(Shape.prototype);
Circle.prototype.constructor = Circle;
```

13.11 Herencia de Clases (II)

```
//ES6
class Rectangle extends Shape {
    constructor (id, x, y, width, height) {
        super(id, x, y)
        this.width  = width
        this.height = height
    }
}
class Circle extends Shape {
    constructor (id, x, y, radius) {
        super(id, x, y)
        this.radius = radius
    }
}
```

13.12 let (I)

```
//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;
})();
```

13.13 let (II)

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

13.14 Scopes (I)

```
//ES5
(function () {
    var foo = function () { return 1; }
    foo() === 1;
    (function () {
        var foo = function () { return 2; }
        foo() === 2;
    })();
    foo() === 1;
})();
```

13.15 Scopes (II)

```
//ES6
{
    function foo () { return 1 }
    foo() === 1
    {
        function foo () { return 2 }
        foo() === 2
    }
    foo() === 1
}
```

13.16 const (I)

```
//ES6
(function() {
    const PI;
    PI = 3.15;
    // ERROR, porque ha de asignarse un valor en la declaración
})();
```

13.17 const (II)

```
//ES6
(function() {
    const PI = 3.15;
    PI = 3.14159;
    // ERROR de nuevo, porque es de sólo-lectura
})();
```

13.18 Template Strings (I)

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

13.19 Template Strings (II)

```
//ES5
var saludo = "ola " +
"que " +
"ase ";
```

13.20 Template Strings (III)

```
//ES6
var saludo = `ola
que
ase`;
```

13.21 Destructuring (I)

```
//ES6
var [a, b] = ["hola", "mundo"];
console.log(a); // "hola"
console.log(b); // "mundo"
```

13.22 Destructuring (II)

```
//ES6
var obj = { nombre: "Adolfo", apellido: "Sanz" };
var { nombre, apellido } = obj;
console.log(nombre); // "Adolfo"
console.log(apellido); // "Sanz"
```

13.23 Destructuring (III)

```
//ES6
var foo = function() {
    return ["180", "78"];
};
var [estatura, peso] = foo();
console.log(estatura); //180
console.log(peso); //78
```

13.24 Parámetros con nombre (I)

```
//ES5
function f (arg) {
    var name = arg[0];
    var val  = arg[1];
    console.log(name, val);
};
function g (arg) {
    var n = arg.name;
    var v = arg.val;
    console.log(n, v);
};
function h (arg) {
    var name = arg.name;
    var val  = arg.val;
    console.log(name, val);
};
f([ "bar", 42 ]);
```

13.25 Parámetros con nombre (II)

```
//ES6
function f ([ name, val ]) {
    console.log(name, val)
}
function g ({ name: n, val: v }) {
    console.log(n, v)
}
function h ({ name, val }) {
    console.log(name, val)
}
f([ "bar", 42 ])
g({ name: "foo", val:  7 })
h({ name: "bar", val: 42 })
```

13.26 Resto parámetros (I)

```
//ES5
function f (x, y) {
    var a = Array.prototype.slice.call(arguments, 2);
    return (x + y) * a.length;
};
f(1, 2, "hello", true, 7) === 9;
```

13.27 Resto parámetros (II)

```
//ES6
function f (x, y, ...a) {
    return (x + y) * a.length
}
f(1, 2, "hello", true, 7) === 9
```

13.28 Valores por defecto (I)

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

13.29 Valores por defecto (I)

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

13.30 Exportar módulos

```
//ES6

// lib/math.js
export function sum (x, y) { return x + y }
export function div (x, y) { return x / y }
export var pi = 3.141593
```

13.31 Importar módulos

```
//ES6

// someApp.js
import * as math from "lib/math"
console.log("2π = " + math.sum(math.pi, math.pi))

// otherApp.js
import { sum, pi } from "lib/math"
console.log("2π = " + sum(pi, pi))
```

13.32 Generadores

```
//ES6
function *soyUnGenerador(i) {
  yield i + 1;
  yield i + 2;
  yield i + 3;
}

var gen = soyUnGenerador(1);
console.log(gen.next());
//  Object {value: 2, done: false}
console.log(gen.next());
//  Object {value: 3, done: false}
console.log(gen.next());
//  Object {value: 4, done: false}
console.log(gen.next());
//  Object {value: undefined, done: true}
```

13.33 Set

```
//ES6
let s = new Set()
s.add("hello").add("goodbye").add("hello")
s.size === 2
s.has("hello") === true
for (let key of s.values()) { // insertion order
  console.log(key)
}
```

13.34 Map

```
//ES6
let m = new Map()
m.set("hello", 42)
m.set(s, 34)
m.get(s) === 34
m.size === 2
for (let [ key, val ] of m.entries()) {
  console.log(key + " = " + val)
}
```

13.35 Nuevos métodos en String

```
//ES6
"hello".startsWith("ello", 1) // true
"hello".endsWith("hell", 4)   // true
"hello".includes("ell")       // true
"hello".includes("ell", 1)    // true
"hello".includes("ell", 2)    // false
```

13.36 Nuevos métodos en Number

```
//ES6
Number.isNaN(42) === false
Number.isNaN(NaN) === true
Number.isSafeInteger(42) === true
Number.isSafeInteger(9007199254740992) === false
```

13.37 Proxies

```
//ES6
let target = {
    foo: "Welcome, foo"
}
let proxy = new Proxy(target, {
    get (receiver, name) {
        return name in receiver ? receiver[name] : `Hello, ${name}`
    }
})
proxy.foo   === "Welcome, foo"
proxy.world === "Hello, world"
```

13.38 Internacionalization (I)

```
//ES6
var i10nUSD = new Intl.NumberFormat("en-US", { style: "currency", currer
var i10nGBP = new Intl.NumberFormat("en-GB", { style: "currency", currer
i10nUSD.format(100200300.40) === "$100,200,300.40"
i10nGBP.format(100200300.40) === "£100,200,300.40"
```

13.39 Internacionalization (II)

```
//ES6
var i10nEN = new Intl.DateTimeFormat("en-US")
var i10nDE = new Intl.DateTimeFormat("de-DE")
i10nEN.format(new Date("2015-01-02")) === "1/2/2015"
i10nDE.format(new Date("2015-01-02")) === "2.1.2015"
```

13.40 Promesas (I)

```
//ES6
var promise = new Promise(function(resolve, reject) {

  var todoCorrecto = true; // o false dependiendo de como ha ido

  if (todoCorrecto) {
    resolve("Promesa Resuelta!");
  } else {
    reject("Promesa Rechazada!");
  }
});
```

13.41 Promesas (II)

```
//ES6

// llamamos el metodo 'then' de la promesa
// con 2 callbacks (resolve y reject)
promise.then(function(result) {
  console.log(result); // "Promesa Resuelta!"
}, function(err) {
  console.log(err); // Error: "Promesa Rechazada!"
});
```

13.42 Promesas (III)

```
//ES6

// podemos también llamar al 'then' con el callback 'resolve'
// y luego al 'catch' con el callback 'reject'
promise.then(function(result) {
  console.log(result); // "Promesa Resuelta!"
}).catch(function(err) {
  console.log(err); // Error: "Promesa Rechazada!"
});
```

13.43 Promesas (IV)

```
//ES6

Promise.all([promesa1,promesa2]).then(function(results) {
  console.log(results); // cuando todas las promesas terminen
}).catch(function(err) {
  console.log(err); // Error: "Error en alguna promesa!"
});
```

13.44 Promesas (V)

```
//ES6

Promise.race([promesa1,promesa2]).then(function(firstResult) {
  console.log(firstResult); // cuando termine la primera
}).catch(function(err) {
  console.log(err); // Error: "Error en alguna promesa!"
});
```

14 Enlaces

14.1 General (ES)

- http://developer.mozilla.org/es/docs/Web/JavaScript/G
- http://cevichejs.com/
- http://www.arkaitzgarro.com/javascript/
- http://www.etnassoft.com/category/javascript/

14.2 General (EN)

- http://www.javascriptkit.com/
- http://javascript.info/
- http://www.howtocreate.co.uk/tutorials/javascript/

14.3 Orientación Objetos (ES) (I)

- 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/

14.4 Orientación Objetos (ES) (II)

- 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/

14.5 Orientación Objetos (EN)

- http://www.codeproject.com/Articles/687093/Understa JavaScript-Object-Creation-Patterns
- http://javascript.info/tutorial/object-oriented-programm
- http://www.howtocreate.co.uk/tutorials/javascript/obje

14.6 Técnicas avanzadas (ES) (I)

- http://www.etnassoft.com/2011/03/14/funciones-autoejecutables-en-javascript/
- http://www.etnassoft.com/2012/01/12/el-valor-de-this-javascript-como-manejarlo-correctamente/
- https://developer.mozilla.org/es/docs/Web/JavaScript/
- http://www.variablenotfound.com/2012/10/closures-en-javascript-entiendelos-de.html

14.7 Técnicas avanzadas (ES) (II)

- 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

14.8 DOM (ES)

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

14.9 DOM (EN)

- http://www.javascriptkit.com/domref/
- http://javascript.info/tutorial/dom

14.10 Frameworks (ES)

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

14.11 Frameworks (EN)

- http://www.slideshare.net/deepusnath/javascript-fram
- http://stackshare.io/stackups/backbone-vs-emberjs-vs
- http://www.hongkiat.com/blog/gulp-vs-grunt/
- https://mattdesl.svbtle.com/browserify-vs-webpack
- http://hackhat.com/p/110/module-loader-webpack-vs-
- http://devzum.com/2014/02/10-best-node-js-mvc-fram
- http://www.tivix.com/blog/nwjs-and-electronjs-web-tec
- http://stackshare.io/stackups/phonegap-vs-ionic-vs-re
- https://developer.salesforce.com/page/Native,_HTML5

14.12 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

14.13 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

14.14 WebSockets (ES)

- http://www.html5rocks.com/es/tutorials/websockets/b
- https://carlosazaustre.es/blog/websockets-como-utiliz
  socket-io-en-tu-aplicacion-web/

14.15 WebSockets (EN)

- https://davidwalsh.name/websocket
- http://code.tutsplus.com/tutorials/start-using-html5-websockets-today--net-13270

14.16 AJAX, JSON, REST (ES)

- https://fernetjs.com/2012/09/jsonp-cors-y-como-los-so
  desde-nodejs/
- http://blog.koalite.com/2012/03/sopa-de-siglas-ajax-js
  cors/
- https://eamodeorubio.wordpress.com/category/webse
- https://eamodeorubio.wordpress.com/category/webse

14.17 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
- http://www.cristalab.com/tutoriales/uso-de-modulos-en-javascript-con-ecmascript-6-c114342l/
- https://burabure.github.io/tut-ES6-promises-generators/

14.18 ES6 (EN)

- http://es6-features.org/
- http://kangax.github.io/compat-table/es5/
- http://www.2ality.com/2015/11/sequential-execution.html
- http://www.html5rocks.com/en/tutorials/es6/promises/
- http://www.datchley.name/es6-promises/