



Formación

Curso JavaScript

2024



Formador

Javier Martín



CONSULTOR / FORMADOR TÉCNICO en el área de PROGRAMACIÓN, DESARROLLO y CALIDAD DE SOFTWARE, cuenta con + de 25 años de experiencia tanto en la gestión y desarrollo de proyectos de software como en la formación técnica de esta área.

Especialista FULLSTACK, imparte formación tanto del BACKEND (Java, .Net, Node JS, Python y Bases de Datos, como del FRONTEND (Angular, React, TypeScript, JavaScript; HTML, CSS3, LESS, SASS, XML, Accesibilidad...), DEVOPS y Herramientas de TESTING, entre otras tecnologías.



training@iconotc.com

▣ **Duración:** 16 horas

▣ **Modalidad:** Presencial / Remoto

▣ **Fechas/Horario:**

- Días 13 y 14 de Marzo de 2024
- Horario de 8:45 a 14:00 y 15:00 a 17:45 hs

▣ **Contenidos:**

- Ciclo de vida de un programa JavaScript
- Elementos y constructos esenciales del lenguaje
- Programación orientada a objetos con JavaScript
- Nociones de programación funcional en JavaScript
- Colecciones de objetos
- Los objetos estándar del navegador
- Interactividad de las páginas: el sistema de eventos
- Gestión de formularios HTML
- Biblioteca estándar de JavaScript
- Introducción a AJAX

[Volver al índice](#)

HERRAMIENTAS DE DESARROLLO

IDEs

- Visual Studio Code - <http://code.visualstudio.com/>
 - VS Code is a Free, Lightweight Tool for Editing and Debugging Web Apps.
 - Visual Studio Code for the Web provides a free, zero-install Microsoft Visual Studio Code experience running entirely in your browser: <https://vscode.dev>
- StackBlitz - <https://stackblitz.com>
 - The online IDE for web applications. Powered by VS Code and GitHub.
- CodeSandbox - <https://codesandbox.io/>
 - Create, share, and get feedback with collaborative sandboxes.
- IntelliJ IDEA - <https://www.jetbrains.com/idea/>
 - Capable and Ergonomic Java * IDE
- Webstorm - <https://www.jetbrains.com/webstorm/>
 - Lightweight yet powerful IDE, perfectly equipped for complex client-side development and server-side development with Node.js

© JMA 2020. All rights reserved

Instalación de utilidades

Consideraciones previas

- Las utilidades son de línea de comandos.
- Para ejecutar los comandos es necesario abrir la consola comandos (Símbolo del sistema)
- Siempre que se realice una instalación o creación es conveniente “Ejecutar como Administrador” para evitar otros problemas.
- En algunos casos el firewall de Windows, la configuración del proxy y las aplicaciones antivirus pueden dar problemas.

GIT: Software de control de versiones

- Descargar e instalar: <https://git-scm.com/>
- Verificar desde consola de comandos:
 - git

Node.js: Entorno en tiempo de ejecución

- Descargar e instalar: <https://nodejs.org>
- Verificar desde consola de comandos:
 - node --version

© JMA 2020. All rights reserved

npm: Node Package Manager

- Aunque se instala con el Node es conveniente actualizarlo:
 - `npm update -g npm`
- Verificar desde consola de comandos:
 - `npm --version`
- Configuración:
 - `npm config edit`
 - `proxy=http://usr:pwd@proxy.dominion.com:8080` ← Símbolos: %HEX ASCII
- Generar fichero de dependencias `package.json`:
 - `npm init`
- Instalación de paquetes:
 - `npm install -g grunt-cli karma karma-cli` ← Global (CLI)
 - `npm install jasmine-core tslint --save --save-dev`
 - `npm install` ← Dependencias en `package.json`
- Arranque del servidor:
 - `npm start`

© JMA 2020. All rights reserved

Bower: Gestor de dependencias del frontend web

- <http://bower.io/>
- Instalación:
 - `npm install -g bower`
- Generar fichero de dependencias `bower.json`:
 - `bower init`
- Descargar, instalar y registrar dependencia de una librería o framework:
 - `bower install jquery -save`

© JMA 2020. All rights reserved

Grunt: Automatización de tareas

- Instalación general:
 - npm install -g grunt-cli
 - npm install -g grunt-init
- Instalación local de los módulos en la aplicación (añadir al fichero package.json de directorio de la aplicación o crearlo si no existe):

```
{
  "name": "my-project-name",
  "version": "0.1.0",
  "devDependencies": {
    "grunt": "~0.4.5",
    "grunt-contrib-jshint": "~0.10.0",
    "grunt-contrib-nodeunit": "~0.4.1",
    "grunt-contrib-uglify": "~0.5.0",
    "grunt-shell": "~0.7.0"
  }
}
```

© JMA 2020. All rights reserved

Grunt: Automatización de tareas

- Descargar e instalar localmente los módulos en la aplicación (situarse en el directorio de la aplicación)
 - npm install --save-dev
- Fichero de tareas: gruntfile.js
- Ejecutar tareas:
 - grunt
 - grunt jshint
 - grunt reléase
 - grunt serve

© JMA 2020. All rights reserved

Karma: Gestor de Pruebas unitarias de JavaScript

- Instalación general:
 - npm install -g karma
 - npm install -g karma-cli
- Generar fichero de configuración karma.conf.js:
 - karma init
- Ingenierías de pruebas unitarias disponibles:
 - <http://jasmine.github.io/>
 - <http://qunitjs.com/>
 - <http://mochajs.org>
 - <https://github.com/caolan/nodeunit>
 - <https://github.com/nealxyc/nunit.js>

© JMA 2020. All rights reserved

Yeoman: Generador del esqueleto web

- <http://Yeoman.io>
- Instalación:
 - npm install -g yo
 - npm install -g generator-angular
 - npm install -g generator-karma
- Generar un servidor y sitio web:
 - yo angular
- Descargar dependencias si es necesario
 - bower install & npm install
- Preparar entorno de ejecución y levantar el servidor en modo prueba:
 - grunt serve

© JMA 2020. All rights reserved



JavaScript



ECMA-262
ISO/IEC-16262

© JMA 2020. All rights reserved

Referencias

-
- [Standard ECMA-262](#)
 - [Latest ECMA spec](#)
 - [El Tutorial de JavaScript Moderno](#)
 - [ES6 Compatibility Table](#)
 - [Comprehensive Overview of ES6 Features](#)
 - [Airbnb JavaScript Style Guide](#)

© JMA 2020. All rights reserved

INTRODUCCIÓN

© JMA 2020. All rights reserved

Introducción

- El JavaScript es un lenguaje de secuencias de comandos multiplataforma e independiente de cualquier empresa, heredero de la sintaxis del C, que originalmente fue diseñada para ir inmerso dentro de las páginas web, ser interpretado por el navegador y hacer programable el HTML.
- Cuando JavaScript debutó en 1996, agregó interactividad a una web que, hasta entonces, estaba compuesta por documentos estáticos. La web se convirtió no solo en un lugar para leer cosas, sino para hacer cosas. La popularidad de JavaScript aumentó constantemente.
- Los desarrolladores que trabajaron con JavaScript escribieron herramientas para resolver los problemas que enfrentaban y las empaquetaron en paquetes reutilizables llamados bibliotecas, para que pudieran compartir sus soluciones con otros. Este ecosistema compartido de bibliotecas ayudó a dar forma al crecimiento de la web.
- Ahora, JavaScript es una parte esencial de la web, utilizado en el 95% de todos los sitios web, y la web es una parte esencial de la vida moderna. Los usuarios escriben artículos, administran sus presupuestos, transmiten música, ven películas y se comunican con otros a grandes distancias instantáneamente, con chat de texto, audio o video.
- La web nos permite hacer cosas que solían ser posibles solo en aplicaciones nativas instaladas localmente. Estos sitios web modernos, complejos e interactivos a menudo se denominan aplicaciones web .

© JMA 2020. All rights reserved

Introducción

- Creado por Brendan Eich para el navegador Netscape Navigator 2.0, que iba a lanzarse en 1995. Inicialmente, se denominó LiveScript.
- Justo antes del lanzamiento, tras la alianza de Netscape con Sun Microsystems, se decidió cambiar el nombre por el de JavaScript. La razón del cambio de nombre fue exclusivamente por marketing, ya que Java era la palabra de moda en el mundo informático y de Internet de la época, no existe ninguna relación con el lenguaje Java de SUN salvo la común herencia de la sintaxis con el C.
- Al mismo tiempo, Microsoft lanzó JScript con su navegador Internet Explorer 3. JScript era una copia de JavaScript al que le cambiaron el nombre para evitar problemas legales.
- En 1997, Netscape decidió liberar el lenguaje y lo estandarizó a través del ECMA. El primer estándar se denominó ECMA-262, en el que se definió por primera vez el lenguaje ECMAScript, quedando el nombre de JavaScript como la implementación que realizó la empresa Netscape del estándar ECMAScript. La ISO adoptó el estándar ECMA-262 dando lugar al estándar ISO/IEC-16262.

© JMA 2020. All rights reserved

Especificaciones oficiales

- ECMA ha publicado varios estándares relacionados con ECMAScript.
- En Junio de 1997 se publicó la primera edición del estándar ECMA-262.
- Un año después, en Junio de 1998 se realizaron pequeñas modificaciones para adaptarlo al estándar ISO/IEC-16262 y se creó la segunda edición.
- La tercera edición del estándar ECMA-262, publicada en Diciembre de 1999, es la versión que soportan todos los navegadores actuales.
- La cuarta versión de ECMA-262 no llegó a publicarse.
- En junio de 2011, se publicó la edición 5.1
- En junio de 2015 aparece la 6ª edición del estándar ECMAScript (ES6) con importantes cambios, cuyo nombre oficial es ECMAScript 2015, y solo la soportan las versiones más modernas de los navegadores.
- A partir de la versión 6 cambia la política de versionado y se saca una revisión anual con pocos cambios: ECMAScript 2016 (ES7), ECMAScript 2017 (ES8), ECMAScript 2018 (ES9), ECMAScript 2019 (ES10), ...
- Se puede consultar la especificación completa en:
 - <http://www.ecma-international.org/publications/standards/Ecma-262.htm>

© JMA 2020. All rights reserved

Añadir JavaScript a una página

- Existen dos formas de insertar código JavaScript dentro de una página: escribiendo código en la misma (en inglés inline) o a través de un archivo externo utilizando la etiqueta script.
- El orden en el cual se incluye el código es importante:
 - Un código que depende de otro debe ser incluido después del que referencia.
 - El código es interpretado secuencialmente desde el inicio de la página.
 - Para mejorar el rendimiento de la página, el código JavaScript debe ser incluido al final del HTML.
 - Es conveniente, cuando se trabaja en un ambiente de producción con múltiples archivos JavaScript, que éstos sean combinados en un solo archivo minimizado.

© JMA 2020. All rights reserved

Etiquetas

- Interno:
`<script type="text/javascript">`
... Código JavaScript ...
`</script>`
- Externo:
`<script type="text/javascript" src="/js/codigo.js"></script>`
- En línea:
`<eti evento="javascript: ... Código JavaScript ..." ...>`
- Para los navegadores que no disponen de soporte completo de JavaScript o en los que ha sido bloqueado o inhabilitado por el usuario.
`<noscript>`
... Aviso en HTML ...
`</noscript>`

© JMA 2020. All rights reserved

TIPOS, VARIABLES Y EXPRESIONES

© JMA 2020. All rights reserved

Sintaxis

- Sensible a mayúsculas y minúsculas.
 - Sentencias separadas por punto y coma (;), aunque opcional en algunas ocasiones es recomendable utilizarlo siempre.
 - Formato libre en las sentencias, una sentencia puede utilizar varias líneas y una línea puede incluir varias sentencias.
 - Los espacios en blanco se compactan a uno solo.
 - Bloques marcados por llaves: { <sentencias> }
 - Comentarios:
 - // hasta el final de la línea
 - /* ... */ una o varias líneas
-

© JMA 2020. All rights reserved

Separadores

- () para contener listas de parámetros en los métodos o funciones, expresiones de control de flujo y establecer precedencias en la expresiones.
- { } para definir bloques de código e inicializar objetos.
- [] para declarar y referencias tablas.
- ; separar instrucciones.
- , separar identificadores en su declaración y concatenar sentencias dentro del for.
- . separador decimal y para hacer referencia a los atributos y métodos.

© JMA 2020. All rights reserved

Identificadores

- Compuestos por letras, números, _, \$.
- Sin límite en el número de caracteres.
- NO pueden comenzar con números.
- NO debe tener el mismo nombre que otro elemento del mismo ámbito.
- NO pueden ser palabras reservadas ni el nombre de un valor con nombre (true / false / null / undefined).
- Palabras reservadas:
 - break, case, catch, continue, debugger, default, delete, do, else, finally, for, function, if, in, instanceof, new, return, switch, this, throw, try, typeof, var, void, while, with
- Palabras reservadas para uso futuro:
 - *class, const, enum, export, extends, import, super, implements, interface, let, package, private, protected, public, static, yield*

© JMA 2020. All rights reserved

Tipos de datos

- JavaScript divide los distintos tipos de datos en dos grupos:
 - Tipos del lenguaje:
 - Tipos primitivos: undefined, null, boolean, string, number
 - Tipos de referencia: Undefined, Null, Boolean, String, Number, Object.
 - Tipos de especificación (meta-valores): Reference, List, Completion, Property Descriptor, Property Identifier, Lexical Environment, Environment Record
- JavaScript define algunos objetos de forma nativa, por lo que pueden ser utilizados directamente por las aplicaciones sin tener que declararlos:
 - Global, Function, Array, Math, Date, Error, JSON y RegExp

© JMA 2020. All rights reserved

Tipos de datos

- Tipo Undefined
 - Tipo con un solo valor llamado “undefined”.
 - Valor de las variables que no han sido aun declaradas o que no se les ha asignado un valor.
- Tipo Null
 - Tipo con un solo valor llamado “null”.
- Tipo Boolean
 - Sólo puede almacenar uno de los dos valores lógicos: “true” o “false”.
 - Los valores false, 0, null y undefined son considerados como false en las operaciones lógicas y el resto de los valores como true.

© JMA 2020. All rights reserved

Tipo numérico

- Acepta Valores aritméticos enteros y reales. Si el número es real, se debe utilizar el punto (.) para separar la parte entera de la decimal.
- Formatos:
 - Decimal
 - Hexadecimal (0x????)
 - Coma flotante (?.??e??).
- JavaScript define valores especiales muy útiles cuando se trabaja con números.
 - Infinity, para representar números demasiado grandes (positivos y negativos) y con los que JavaScript no puede trabajar.
 - NaN (Not a Number), resultado de las operaciones matemáticas con variables no numéricas o la indeterminación en calculo numérico. isNaN() informa de que el valor no puede ser usado como numérico.
- JavaScript define constantes y operaciones matemáticas adicionales a través del objeto Math.

© JMA 2020. All rights reserved

Tipo cadena

- Conjunto de caracteres encerrados entre comillas simples o dobles.
- Permite el anidamiento de una cadena entre comillas dobles dentro de una cadena entre comillas simples y viceversa.
- Las constantes de tipo cadena no pueden ocupar mas de una línea, para utilizar varias líneas requiere concatenación.
- El formato de las cadenas es UTF8 y pueden contener secuencias de escape:
 - \n, \r, \\\, \", \', \t, \v, \e, \f
 - \[0-9]{1,3}
 - \x[0-9A-Fa-f]{2}
 - \u[0-9A-Fa-f]{4}

© JMA 2020. All rights reserved

Template Strings (ES6)

- Interpolación: sustituye dentro de la cadena las variables por su valor:

```
let var1 = "JavaScript";  
let var2 = "Templates";  
console.log(`El ${var1} ya tiene ${var2}.`); // El JavaScript ya tiene Templates.
```

- Se puede personalizar la interpolación con una función que recibe como un array los fragmentos de cadena y un argumento por expresión:

```
frmt`El ${var1} ya tiene ${var2}.` // frmt(["El ", " ya tiene "], var1, var2);
```

- Las constantes de cadena pueden ser multilinea sin necesidad de concatenarlos con +.

```
let var1 = "El JavaScript  
ya tiene  
Templates";
```

- Incorpora soporte extendido para el uso de Unicode en cadenas y expresiones regulares (caracteres en hebreo, árabe, cirílico, ...).

© JMA 2020. All rights reserved

Conversión entre tipos

- JavaScript es un lenguaje "no tipado", lo que significa que una misma variable puede guardar diferentes tipos de datos a lo largo de la ejecución de la aplicación. En JavaScript las conversiones entre tipos numéricos y cadenas se realiza de forma implícita. Los valores aritméticos se convierten automáticamente en cadenas en las operaciones de concatenación.
- En caso de ser necesario, para realizar las conversiones de forma explícita se utiliza:
 - El método `toString()` que permite convertir variables de cualquier tipo a variables de tipo cadena.
 - Las funciones `parseInt()` y `parseFloat()` convierten la cadena en un número entero o decimal respectivamente.
 - Si la cadena no contiene un valor aritmético o no empieza por un dígito devuelve el valor `NaN`.
 - La conversión numérica de una cadena se realiza carácter a carácter empezando por el de la primera posición hasta que encuentra un carácter que no es un dígito o termina la cadena.

© JMA 2020. All rights reserved

Operadores

- **Asignación**
Simple: <Destino> = <Expresión>
- **Aritméticos**
 - + : Suma, si el algún operando es una cadena se concatena.
 - : Resta, cambio de signo cuando es unario.
 - * : Producto
 - / : División, el resultado es siempre real.
 - % : Resto de la división entera
- **Acumulativos: += , -= , *= , /= , %=**
- **Operadores relacionales**
 - > : mayor
 - >= : mayor o igual
 - < : menor
 - <= : menor o igual
 - == : igual
 - != : distinto
 - === : identidad (coincide el tipo)
 - !== : no identidad

© JMA 2020. All rights reserved

Operadores

- **Operadores lógicos (los operandos son booleanos)**
 - && : AND
 - || : OR
 - ! : NOT
- **Incremento/decremento:**
 - ++<Variable> :Se incrementa en 1 y se consulta.
 - <Variable> :Se decrementa en 1 y se consulta.
 - <Variable>++ :Se consulta y se incrementa en 1.
 - <Variable>-- :Se consulta y se decrementa en 1.
- **Binarios**
 - >> : Desplazamiento Derecha
 - << : Desplazamiento Izquierda
 - >>> : Desplazamiento Derecha sin signo
 - & : AND binario
 - | : OR binario
 - ^ : XOR binario
 - ~ : complemento
- **Acumulativos: >>=, <<=, >>>=, &=, |=, ^=, ~=**

© JMA 2020. All rights reserved

Operadores

- Operador unario: **new**
 - Crea un nuevo objeto.
- Operador unario: **delete**
 - Elimina una propiedad de un objeto o quita un elemento de una matriz.
- Operador unario: **void**
 - Descarta el operando y devuelve no definido (undefined).
- Operador unario de tipo: **typeof**
 - Devuelve una cadena con el tipo de la variable, los valores devueltos son: "number", "string", "boolean", "object", "function" y "undefined".
- Operador binario de pertenencia a tipo: **instanceof**
 - Devuelve una true si es del tipo indicado.
- Operador binario de pertenencia: **in**
 - Comprueba la existencia de una propiedad en un objeto: property in object
- Operador condicional ternario:
 - <condición>?<expresión cuando verdadero>:<expresión cuando falso>

© JMA 2020. All rights reserved

Precedencia de operadores

Operador	Descripción
. [] ()	Acceso a campos, indexación de matrices y llamadas a funciones
++ -- ~ ! typeof new void delete	Operadores unarios, tipos de datos devueltos, creación de objetos, valores no definidos
* / %	Multiplicación, división, división módulo
+ - +	Adición, sustracción, concatenación de cadenas
<< >> >>>	Desplazamiento de bits
< <= > >=	Menor que, menor que o igual a, mayor que, mayor que o igual a
== != === !==	Igualdad, desigualdad, identidad, no identidad
&	AND de bits
^	XOR de bits
	OR de bits
&&	AND lógico
	OR lógico
?:	Condicional
= OP=	Asignación, asignación con operación
,	Evaluación múltiple

© JMA 2020. All rights reserved

Variables

- No es necesario declararlas de forma explícita, pero sí recomendable, aunque con la directiva de prologo 'use strict' el navegador generara error si no se declara.
- Se declaran sin tipo, van asumiendo el tipo del valor contenido.
- Declaración:
`var <Identificador> [= expresión], <Otro> [= expresión], ...;`
- Ámbito:
 - Local:
 - Función en la que está declarada
 - Accesible desde la propia función y todos sus bloques anidados.
 - Global:
 - Declarada fuera de cualquier función.
 - Las variables no declaradas son siempre globales.
 - Accesibles desde cualquier punto.
 - Las variables locales prevalecen sobre las globales

© JMA 2020. All rights reserved

Declaración de variables (ES6)

- **let:** ámbito de bloque, solo accesible en el bloque donde está declarada.

```
(function() {  
  if(true) {  
    let x = "variable local";  
  }  
  console.log(x); // error, "x" definida dentro del "if"  
})();
```
- **const:** constante, se asina valor al declararla y ya no puede cambiar de valor.

```
const PI = 3.15;  
PI = 3.14159; // error, es de sólo-lectura
```
- Las constantes numérica ahora se pueden expresar en binario y octal.

```
0b111110111 === 503  
0o767 === 503
```

© JMA 2020. All rights reserved

Destructuring (ES6)

- Asignar (repartir) los valores de un objeto o array en varias variables:

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

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

© JMA 2020. All rights reserved

INSTRUCCIONES DE CONTROL

© JMA 2020. All rights reserved

Bifurcación simple

```
if(<Condición>)  
  <Instrucción>; o {<Bloque de instrucciones>}  
[  
  else  
    <Instrucción>; o {<Bloque de instrucciones>}  
]
```

- Sin límite en los anidamientos.
- La parte **else** es opcional. Para evitar confusión es correcto crear bloques cuando existan anidamientos.

© JMA 2020. All rights reserved

Bifurcación múltiple

```
switch(<Expresión>) {  
  case <Valor 1> :  
    <Bloque de instrucciones>  
    [break;]  
  case <Valor 2> :  
    <Bloque de instrucciones>  
    [break;]  
  ...  
  [default:  
    <Bloque de instrucciones>]  
}
```

- Se ejecutan las instrucciones en cascada hasta el final o hasta que encuentra un **break**.
- Sin límite en los anidamientos.
- La parte **default** es opcional, se ejecuta cuando no se cumplen las condiciones anteriores.

© JMA 2020. All rights reserved

Bucles

```
while(<Condición>
    ; o <Instrucción>; o {<Bloque de instrucciones>}
```

```
do
    <Instrucción>; o <Bloque de instrucciones>
while (<Condición>;
```

- while: Se ejecuta de 0 a n veces.
- do while: Se ejecuta de 1 a n veces.

© JMA 2020. All rights reserved

Bucles

```
for([var] <Inicio>; <Final>; <Iteración>)
    ; o <Instrucción>; o {<Bloque de instrucciones>}
```

- Apartados (todos son opcionales):
 - <Inicio> : Se ejecutan antes de comenzar. Lista de expresiones separadas por comas. La palabra reservada **var** es opcional.
 - <Final> : Expresión condicional de finalización, se ejecuta mientras se cumpla la condición.
 - <Iteración> : Se ejecutan en cada iteración. Lista de expresiones separadas por comas.

```
for([var] <Variable> in <Objeto>)
    <Instrucción>; o {<Bloque de instrucciones>}
```

- A <Variable> se le asignan uno a uno todos los valores del índice o nombre de propiedad del <Objeto>.
- Para acceder a los valores contenido en el <Objeto>:

```
for(var cmp in obj)
    console.log(obj[cmp]);
```

(ES6)

```
for([var] <Variable> of <Objeto>)
    <Instrucción>; o {<Bloque de instrucciones>}
```

- A <Variable> se le asignan uno a uno todos los valores contenidos en <Objeto>.

© JMA 2020. All rights reserved

Control de flujo

- **break;**
 - Termina la instrucción asociada y pasa a la siguiente.
- **continue;**
 - Detiene el ciclo actual de un bucle y salta a evaluar la condición.
- **return [<Expresión>;]**
 - Termina la ejecución de la función y, opcionalmente, devuelve el resultado de la misma;

© JMA 2020. All rights reserved

Tratamiento de excepciones

```
try
    {<Bloque de instrucciones>}
catch (<Identificador>)
    {<Bloque de instrucciones>}
finally
    {<Bloque de instrucciones>}
```

- La sección catch solo se ejecuta en caso de error. El <Identificador> contiene un objeto de tipo Error con la información adicional sobre la excepción.
- La sección finally se ejecuta siempre, independientemente de si se produce o no la excepción.
- Las secciones catch y finally son opcionales pero al menos debe aparecer una.
- Se pueden anidar instrucciones try.
- El código de las secciones catch y finally no se encuentra vigilado por lo que es necesario envolverlo en una instrucción try si es susceptible de producir excepciones.
- Para lanzar excepciones se utiliza:
`throw new Error(<Mensaje>);`

© JMA 2020. All rights reserved

Otras instrucciones

- **with** (<objeto>)
 {<Bloque de instrucciones>}
 - Selector de objeto: No es necesario utilizar el punto para la selección de sus métodos y atributos.
- **debugger;**
 - Provoca un punto de interrupción cuando se ejecuta en un depurador, si un depurador no está presente o activo esta declaración no tiene ningún efecto observable.

© JMA 2020. All rights reserved

FUNCIONES

© JMA 2020. All rights reserved

Sintaxis

```
function Identificador ([<Lista de parámetros>]) {  
    ...  
    [return [<Expresión>];]  
}
```

- Una función es un objeto con conjunto de instrucciones que se agrupan para realizar una tarea concreta y que se pueden reutilizar fácilmente, permitiendo modularizar el código.
- Una función puede devolver un valor y participar en expresiones.
- No es obligatorio que las funciones tengan una instrucción de tipo `return` para devolver valores. Cuando una función no devuelve ningún valor o cuando en la instrucción `return` no se indica ningún valor, automáticamente se devuelve el valor `undefined`.
- Una función puede contener en su interior otras funciones anidadas.

© JMA 2020. All rights reserved

Lista de parámetros

- Una función puede recibir uno o varios valores para realizar sus operaciones.
- La lista de parámetros es una lista de identificadores separados por comas.
- Al ser el JavaScript un lenguaje "no tipado", no es posible asegurar que los parámetros que se pasan a una función sean del tipo adecuado para las operaciones que realiza la función.
- El número de argumentos que se pasa a una función debería ser el mismo que el número de argumentos que ha indicado la función. No obstante, JavaScript no muestra ningún error si se pasan más o menos argumentos de los necesarios.
- Si se pasan menos parámetros que los definidos en la función, al resto de parámetros hasta completar el número correcto se les asigna el valor `undefined`.
- Si a una función se le pasan más argumentos que los parámetros que ha definido, a los argumentos sobrantes no se les asignan nombres.
- El orden de los argumentos es fundamental, ya que el primer dato que se indica en la llamada, será el primer valor que espera la función; el segundo valor indicado en la llamada, es el segundo valor que espera la función y así sucesivamente.
- Se puede utilizar un número ilimitado de argumentos.

```
function fn(p1, p2, p3) { ... }  
rslt = fn('algo', var1, 3*pi);
```

© JMA 2020. All rights reserved

Lista de parámetros variables

- Una función se puede definir sin parámetros pero utilizar los parámetros pasados.
- La propiedad de la función `arguments` es un objeto de tipo `Arguments` que se puede tratar como si fuera un `Array`.

```
function avg() {  
  var rslt= 0;  
  for(var i=0; i < arguments.length; i++) {  
    rslt += arguments[i];  
  }  
  return arguments.length ? (rslt / arguments.length) : 0;  
}
```

```
var variable1 = avg(1, 3, 5, 8);  
var variable2 = avg(4, 6, 8, 1, 2, 3, 4, 5);
```

- La propiedad `callee` hace referencia a la función que se está ejecutando, accediendo a `arguments.callee.length` se puede obtener el número de parámetros con los que se ha definido la función.

© JMA 2020. All rights reserved

Parámetros de funciones (ES6)

- Operador de propagación: Convierte un array o cadena en una lista de parámetros.

```
var str = "foo"  
var chars = [ ...str ] // [ "f", "o", "o" ]
```

- Valores por defecto: Se pueden definir valores por defecto a los parámetros en las funciones.

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

- Resto de los parámetros: Convierte una lista de parámetros en un array.

```
function f (x, y, ...a) {  
  return (x + y) * a.length  
}
```

```
f(1, 2, "hello", true, 7) === 9
```

© JMA 2020. All rights reserved

Tipo Function

- Las funciones son objetos de tipo Function y como tal se pueden almacenar en variables.

```
function suma(a, b) {  
    return a + b;  
}  
var miFuncion = suma;  
rslt = miFuncion(2, 2);
```
- La asignación se debe realizar solo con el identificador sin paréntesis para evitar que se evalúe.
- De igual forma se pueden pasar una función como valor de un argumento de otra función.

```
function calcula(fn, a, b) {  
    return fn(a, b);  
}  
  
rslt = calcula(suma, 2, 2);
```

© JMA 2020. All rights reserved

Funciones anónimas

- La asignación permite definir una función con una expresión en la que el nombre de la función es opcional, se conocen como funciones anónimas.

```
var miFuncion = function(a, b) { return a + b; }
```
- De igual forma, una función puede devolver otra función:

```
return function() { return a + b; }
```
- Una función anónima autoejecutable consiste en crear una expresión de función e inmediatamente ejecutarla. Es muy útil para casos en que no se desea intervenir espacios de nombres globales, debido a que ninguna variable declarada dentro de la función es visible desde afuera.

```
(function(){  
    var local= 'Valor interno';  
})();
```

© JMA 2020. All rights reserved

Función Arrow (ES6)

- Funciones anónimas.

```
data.forEach(elem => {  
  console.log(elem);  
  // ...  
});  
var fn = (num1, num2) => num1 + num2;  
pairs = evens.map(v => ({ even: v, odd: v + 1 }));  
fn = () => {console.log("Error");};
```

- **this:** dentro de una función Arrow hace referencia al contenedor y no al contexto de la propia función.

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

- equivale a (ES5):

```
bar : function() {  
  document.addEventListener("click", function(e) {  
    this.foo();  
  }).bind(this));  
}
```

© JMA 2020. All rights reserved

OBJETOS Y CLASES

© JMA 2020. All rights reserved

Introducción

- El JavaScript utiliza objetos pero no es un lenguaje orientado a objetos, esta orientado a prototipos.
- No soporta clases, herencia, sobrecarga, ...
- Técnicamente, un objeto de JavaScript es un array asociativo formado por las propiedades y los métodos del objeto.
- Un array asociativo es aquel en el que cada elemento no está asociado a su posición numérica dentro del array, sino que está asociado a otro valor específico, pares nombre/valor. En los arrays normales, los valores se asocian a índices que siempre son numéricos, mientras que en los arrays asociativos se asocian a claves que siempre son cadenas de texto.
- Las propiedades son variables globales del objeto y los métodos son variables globales del objeto pero de tipo Function.
- Los miembros son accesibles externamente mediante la notación punto (objeto.miembro) o mediante la notación array (objeto["miembro"]). Los miembros son accesibles internamente mediante la notación punto y la palabra clave this (this.miembro) .

© JMA 2020. All rights reserved

Creación de Objetos

- Implementación directa

```
var elObjeto = new Object();
elObjeto.id = "99";
elObjeto.nombre = "Objeto de prueba";
elObjeto.muestraId = function() {
    alert("El ID del objeto es " + this.id);
}
elObjeto.ponNombre = function(nom) {
    this.nombre=nom.toUpperCase();
}
```
- Notación JSON

```
var elObjeto = {
    id : "99",
    nombre : "Objeto de prueba",
    muestraId : function() {
        alert("El ID del objeto es " + this.id);
    },
    ponNombre : function(nom) {
        this.nombre=nom.toUpperCase();
    }
}
```

© JMA 2020. All rights reserved

Funciones constructoras

- Al contrario que en los lenguajes orientados a objetos, en JavaScript no existe el concepto de constructor. Por lo tanto, al definir un objeto no se incluyen uno o varios constructores. En realidad, JavaScript emula el funcionamiento de los constructores mediante el uso de funciones constructoras.

```
function MiClase(elId, elNombre) {  
    this.id = elId;  
    this.nombre = elNombre;  
    this.muestraId = function() {  
        alert("El ID del objeto es " + this.id);  
    }  
    this.ponNombre = function(nom) {  
        this.nombre=nom.toUpperCase();  
    }  
}  
var elObjeto = new MiClase("99", "Objeto de prueba");
```

© JMA 2020. All rights reserved

Prototype

- Cada vez que se instancia un objeto con la función constructora, se definen tantas nuevas funciones como métodos incluya la función constructora.
- La penalización en el rendimiento y el consumo excesivo de recursos de esta técnica puede suponer un inconveniente en las aplicaciones profesionales realizadas con JavaScript.
- Todos los objetos de JavaScript incluyen una referencia interna a otro objeto llamado *prototype* o "prototipo". Cualquier propiedad o método que contenga el objeto prototipo, está presente de forma automática en el objeto original.
- Es como si cualquier objeto de JavaScript heredara de forma automática todas las propiedades y métodos de otro objeto llamado *prototype*. Cada tipo de objeto diferente hereda de un objeto *prototype* diferente. Dado que el *prototype* es el molde con el que se fabrica cada objeto de ese tipo. Si se modifica el molde o se le añaden nuevas características, todos los objetos fabricados con ese molde tendrán esas características.

© JMA 2020. All rights reserved

Prototype

- En el prototype de un objeto sólo se deben añadir aquellos elementos comunes para todos los objetos. Normalmente se añaden los métodos y las constantes (propiedades cuyo valor no varía durante la ejecución de la aplicación). Las propiedades del objeto permanecen en la función constructora para que cada objeto diferente pueda tener un valor distinto en esas propiedades.

```
function MiClase(elId, elNombre) {  
    this.id = elId;  
    this.nombre = elNombre;  
}  
MiClase.prototype.muestraId = function() {  
    alert("El ID del objeto es " + this.id);  
}  
MiClase.prototype.ponNombre = function(nom) {  
    this.nombre=nom.toUpperCase();  
}  
}
```

© JMA 2020. All rights reserved

Prototype

- La propiedad prototype también permite añadir y/o modificar las propiedades y métodos de los objetos predefinidos por JavaScript.
- Por lo tanto, es posible redefinir el comportamiento habitual de algunos métodos de los objetos nativos de JavaScript.
- Además, se pueden añadir propiedades o métodos completamente nuevos.
- Existen librerías de JavaScript formadas por un conjunto de utilidades que facilitan la programación de las aplicaciones y una de sus características habituales es el uso de la propiedad prototype para mejorar las funcionalidades básicas de JavaScript.

© JMA 2020. All rights reserved

Palabra clave this

- En JavaScript, así como en la mayoría de los lenguajes de programación orientados a objetos, **this** es una palabra clave especial que hace referencia al objeto en donde el método está siendo invocado.
- El valor de **this** se determina utilizando una serie de simples reglas:
 1. Si la función es invocada utilizando `Function.call` o `Function.apply`, **this** tendrá el valor del primer argumento pasado al método. Si el argumento es nulo (`null`) o indefinido (`undefined`), **this** hará referencia al objeto global (el objeto `window`);
 2. Si la función a invocar es creada utilizando `Function.bind`, **this** será el primer argumento que es pasado a la función en el momento en que se la crea;
 3. Si la función es invocada como un método de un objeto, **this** referenciará a dicho objeto;
 4. De lo contrario, si la función es invocada como una función independiente, no unida a algún objeto, **this** referenciará al objeto global.

© JMA 2020. All rights reserved

Métodos `apply()` y `call()`

- Los métodos del objeto `Function` `apply()` y `call()` permiten ejecutar una función como si fuera un método de otro objeto. La única diferencia entre los dos métodos es la forma en la que se pasan los argumentos a la función.
- El primer parámetro del método `call()` es el objeto sobre el que se va a ejecutar la función. El resto de parámetros del método `call()` son los parámetros que se pasan a la función.

```
function miFuncion(x) {  
    return this.numero + x;  
}  
var elObjeto = new Object();  
elObjeto.numero = 5;  
  
var resultado = miFuncion.call(elObjeto, 4);  
alert(resultado);
```
- El método `apply()` es idéntico al método `call()`, salvo que en este caso los parámetros se pasan como un array:

```
var resultado = miFuncion.apply(elObjeto, [4]);  
alert(resultado);
```

© JMA 2020. All rights reserved

Propiedades de objetos (ES6)

ES 2015

```
obj = { x, y }  
obj = {  
  foo (a, b) {  
    ...  
  },  
  bar (x, y) {  
    ...  
  },  
  *iter (x, y) {  
    ...  
  }  
}
```

Anteriormente:

```
obj = { x: x, y: y };  
obj = {  
  foo: function (a, b) {  
    ...  
  },  
  bar: function (x, y) {  
    ...  
  },  
  // iter: sin equivalencia  
  ...  
};
```

© JMA 2020. All rights reserved

Clases (ES6)

- Ahora JavaScript tendrá clases, muy parecidas las funciones constructoras de objetos que realizábamos en el estándar anterior, pero ahora bajo el paradigma de clases, con todo lo que eso conlleva, como por ejemplo, herencia.

```
class LibroTecnico extends Libro {  
  constructor(tematica, paginas) {  
    super(tematica, paginas);  
    this.capitulos = [];  
    this.precio = "";  
    // ...  
  }  
  metodo() {  
    // ...  
  }  
}
```

© JMA 2020. All rights reserved

Static Members (ES6)

```
class Rectangle extends Shape {
  ...
  static defaultRectangle () {
    return new Rectangle("default", 0, 0, 100, 100)
  }
}
class Circle extends Shape {
  ...
  static defaultCircle () {
    return new Circle("default", 0, 0, 100)
  }
}
var defRectangle = Rectangle.defaultRectangle()
var defCircle   = Circle.defaultCircle()
```

© JMA 2020. All rights reserved

Getter/Setter (ES6)

```
class Rectangle {
  constructor (width, height) {
    this._width = width;
    this._height = height;
  }
  set width (width) { this._width = width;      }
  get width ()    { return this._width;        }
  set height (height) { this._height = height;  }
  get height ()    { return this._height;      }
  get area ()      { return this._width * this._height; }
}
var r = new Rectangle(50, 20)
r.area === 1000
```

© JMA 2020. All rights reserved

mixin (ES6)

- Soporte para la herencia de estilo mixin mediante la ampliación de las expresiones que producen objetos de función.

```
var aggregation = (baseClass, ...mixins) => {  
  let base = class _Combined extends baseClass {  
    constructor (...args) {  
      super(...args)  
      mixins.forEach((mixin) => {  
        mixin.prototype.initializer.call(this)  
      })  
    }  
  }  
  let copyProps = (target, source) => {  
    Object.getOwnPropertyNames(source)  
      .concat(Object.getOwnPropertySymbols(source))  
      .forEach((prop) => {  
        if (prop.match(/^(?:constructor|prototype|arguments|caller|name|bind|call|apply|toString|length)$/))  
          return  
        Object.defineProperty(target, prop, Object.getOwnPropertyDescriptor(source, prop))  
      })  
  }  
  mixins.forEach((mixin) => {  
    copyProps(base.prototype, mixin.prototype)  
    copyProps(base, mixin)  
  })  
  return base  
}
```

© JMA 2020. All rights reserved

Módulos (ES6)

- Estructura el código en módulos similares a los espacios de nombres
- Los módulos siempre llevan 'use strict' de forma predeterminada.
- Cada fichero se comporta como un módulo y solo se carga la primera vez.
- Solo las partes marcadas como export pueden ser importadas en otros ficheros.
- Se puede llamar a las funciones desde los propios Scripts, sin tener que importarlos en el HTML (si usamos JavaScript en el navegador).

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

Y para importar en otro fichero:

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

```
<script type="module">  
import { app } from ".app.js";
```

© JMA 2020. All rights reserved

Módulos: Exportaciones (ES6)

- Se puede etiquetar cualquier variable, contante, función o clase como exportada colocando precediéndola con la clausula 'export'.
`export oneFunc() { ... }`
- La etiqueta 'export default' permite la exportación predeterminada de una función o clase (exportación única o principal).
`export default MyClass { ... }`
- La instrucción 'export' permite dar la lista de variables, contantes, funciones o clases exportadas (separadas por comas). La clausula as en la lista permite exportar con un nombre exterior diferente.
`export { MY_CONST, MyClass, oneFunc, count as contador, ... }`
- La reexportación permite importar e inmediatamente exportarlas, para crear los ficheros index.js que aíslen de la complejidad interna de un directorio y sus sub directorios:
`export ... from ...`

© JMA 2020. All rights reserved

Módulos: Importaciones (ES6)

- Con * se importan todo (exportaciones) pero hay que asignarle un espacio de nombres con as, el acceso se realiza a través de dicho nombre:
`import * as ns from ...`
- Se puede especificar la lista de lo que se desea importar (sin espacio de nombres):
`import {MyClass, oneFunc as contador, ... } from ...`
- La clausula as en la lista permite establecer nombres locales alternativos para evitar conflictos.
- La importación de la exportación predeterminada requiere un nombre local:
`import MyNameForDefault from ...`
- La clausula from indica la ubicación del fichero del módulo: es una ruta relativa al fichero sin extensión. Las rutas sin fichero utilizaran el fichero index.js de reexportación. Las rutas absolutas pueden estar asociadas a las bibliotecas o librerías en las herramientas de ensamblaje.
`import * as ns from './my-module'`

© JMA 2020. All rights reserved

Iteradores y Generadores (ES6)

- El patrón Iterador permite trabajar con colecciones por medio de abstracciones de alto nivel
- Un Iterador es un objeto que sabe como acceder a los elementos de una secuencia, uno cada vez, mientras que mantiene la referencia a su posición actual en la secuencia.
- En ES2015 las colecciones (arrays, maps, sets) son objetos iteradores.
- Los generadores permiten la implementación del patrón Iterador.
- Los Generadores son funciones que pueden ser detenidas y reanudadas en otro momento.
- Estas pausas en realidad ceden la ejecución al resto del programa, es decir no bloquean la ejecución.
- Los Generadores devuelven (generan) un objeto "Iterator" (iterador)

© JMA 2020. All rights reserved

Generadores (ES6)

- Para crear una función generadora

```
function* myGenerator() {  
  // ...  
  yield value;  
  // ...  
}
```
- La instrucción yield devuelve el valor y queda a la espera de continuar cuando se solicite el siguiente valor.
- El método next() ejecuta el generador hasta el siguiente yield dentro del mismo y devuelve un objeto con el valor.

```
var iter = myGenerator();  
// ...  
rslt = iter.next();  
// ...
```
- La nueva sintaxis del for permite recorrer el iterador completo:

```
for (let value of iter)
```

© JMA 2020. All rights reserved

<http://ecma-international.org/ecma-262/5.1/#sec-15>

STANDARD BUILT-IN ECMAScript OBJECTS

© JMA 2020. All rights reserved

Objeto Global

- El objeto Global es parte del entorno léxico del programa en ejecución.
- Se encarga de encapsular los elementos no sintácticos del lenguaje:
 - contantes globales (propiedades valor)
 - NaN, Infinity, undefined
 - funciones globales (propiedades función)
 - eval(x), parseInt(string , radix), parseFloat(string), isNaN(number), isFinite(number)
 - decodeURI(uri), decodeURIComponent(uri), encodeURI(uri), encodeURIComponent(uri)
 - objetos globales (propiedades referencia)
 - Math, JSON
 - tipos predefinidos (propiedades funciones constructoras).
 - Object, Function, Array, String, Boolean, Number, Date, RegExp
 - Error, EvalError, RangeError, ReferenceError, SyntaxError, TypeError, URIError

© JMA 2020. All rights reserved

Object

- **Object.create** (Función): Crea un objeto que tiene un prototipo especificado y contiene opcionalmente propiedades especificadas.
- **Object.defineProperties** (Función): Agrega una o varias propiedades a un objeto, o modifica atributos de propiedades existentes.
- **Object.defineProperty** (Función): Agrega una propiedad a un objeto o modifica atributos de una propiedad existente.
- **Object.seal** (Función): Impide la modificación de atributos de propiedades existentes e impide agregar nuevas propiedades.
- **Object.freeze** (Función): Impide la modificación de atributos y valores de propiedad existentes e impide agregar nuevas propiedades.
- **Object.preventExtensions** (Función): Impide la adición de nuevas propiedades a un objeto.
- **Object.isExtensible** (Función): Devuelve un valor que indica si se pueden agregar nuevas propiedades a un objeto.
- **Object.isFrozen** (Función): Devuelve true si no se pueden modificar atributos y valores de propiedad existentes en un objeto y no se pueden agregar nuevas propiedades al objeto.

© JMA 2020. All rights reserved

Object

- **Object.isSealed** (Función): Devuelve true si no se pueden modificar atributos de propiedad existentes en un objeto y no se pueden agregar nuevas propiedades al objeto.
- **Object.keys** (Función): Devuelve los nombres de las propiedades y los métodos enumerables de un objeto.
- **Object.getPrototypeOf** (Función): Devuelve el prototipo de un objeto.
- **Object.getOwnPropertyDescriptor** (Función): Devuelve la definición de una propiedad de datos o de una propiedad de descriptor de acceso.
- **Object.getOwnPropertyNames** (Función): Devuelve los nombres de las propiedades y métodos de un objeto.
- **prototype** (Propiedad): Devuelve una referencia al prototipo correspondiente a una clase de objetos.
- **toLocaleString** (Método): Devuelve un objeto convertido en una cadena basándose en la configuración regional actual.
- **toString** (Método): Devuelve una representación en forma de cadena de un objeto.
- **valueOf** (Método): Devuelve el valor primitivo del objeto especificado.

© JMA 2020. All rights reserved

Function

- **arguments** (Propiedad): Obtiene los argumentos del objeto Function que se está ejecutando actualmente.
- **caller** (Propiedad): Obtiene la función invocada por la función actual.
- **length** (Propiedad): Obtiene el número de argumentos definidos para una función.
- **apply** (Método): Llama a la función, sustituyendo el objeto especificado por el valor de `this` de la función, y la matriz especificada por los argumentos de la función.
- **bind** (Método): Para una función determinada, crea una función enlazada con el mismo cuerpo que la función original. En la función enlazada, el objeto `this` se resuelve en el objeto pasado. La función enlazada tiene los parámetros iniciales especificados.
- **call** (Método): Llama a un método de un objeto y sustituye el objeto actual por otro objeto.

© JMA 2020. All rights reserved

Error

- **name** (Propiedad): Devuelve el nombre de un error.
- **message** (Propiedad): Devuelve una cadena con un mensaje de error.
 - *RangeError*: Este error se produce cuando se proporciona a una función un argumento que ha superado su intervalo permitido.
 - *ReferenceError*: Este error tiene lugar cuando se detecta una referencia no válida.
 - *SyntaxError*: Este error se produce cuando se analiza el texto de origen y su sintaxis no es correcta.
 - *TypeError*: Este error se produce cuando el tipo real de un operando no coincide con el tipo esperado.
 - *URIError*: Este error tiene lugar cuando se detecta un identificador uniforme de recursos (URI) no válido.

© JMA 2020. All rights reserved

Array

- **length** (Propiedad): Devuelve un valor entero que supera en uno al elemento mayor definido en una matriz.
- **Array.isArray** (Función): : Devuelve un valor de tipo booleano que indica si un objeto es una matriz.
- **concat** (Método): Devuelve una matriz nueva que se compone de una combinación de dos matrices.
- **join** (Método): Devuelve un objeto String formado por todos los elementos de una matriz concatenados.
- **pop** (Método): Quita el último elemento de una matriz y lo devuelve.
- **push** (Método): Anexa nuevos elementos a una matriz y devuelve la nueva longitud de la matriz.
- **reverse** (Método): Devuelve un objeto Array con los elementos invertidos.
- **shift** (Método): Quita el primer elemento de una matriz y lo devuelve.
- **slice** (Método): Devuelve una sección de una matriz.
- **sort** (Método): Devuelve un objeto Array con los elementos ordenados.

© JMA 2020. All rights reserved

Array

- **splice** (Método): Quita elementos de una matriz, inserta nuevos elementos en su lugar si procede y devuelve los elementos eliminados.
- **unshift** (Método): Inserta nuevos elementos al principio de una matriz.
- **indexOf** (Método): Devuelve el índice de la primera aparición de un valor de una matriz.
- **lastIndexOf** (Método): Devuelve el índice de la última aparición de un valor especificado de una matriz.
- **every** (Método): Comprueba si una función de devolución de llamada definida devuelve true para todos los elementos de una matriz.
- **some** (Método): Comprueba si una función de devolución de llamada definida devuelve true para cualquier elemento de una matriz.
- **forEach** (Método): Llama a una función de devolución de llamada definida para cada elemento de una matriz.
- **map** (Método): Llama a una función de devolución de llamada definida para cada elemento de una matriz y devuelve una matriz que contiene los resultados.

© JMA 2020. All rights reserved

Array

- **filter** (Método): Llama a una función de devolución de llamada definida para cada elemento de una matriz y devuelve una matriz de aquellos valores para los que esa función devuelve true.
- **reduce** (Método): Acumula un solo resultado mediante la llamada a una función de devolución de llamada definida para todos los elementos de una matriz. El valor devuelto de la función de devolución de llamada es el resultado acumulado, y se proporciona como un argumento en la siguiente llamada a dicha función.
- **reduceRight** (Método): Acumula un solo resultado mediante la llamada a una función de devolución de llamada definida para todos los elementos de una matriz, en orden descendente. El valor devuelto de la función de devolución de llamada es el resultado acumulado, y se proporciona como un argumento en la siguiente llamada a dicha función.
- **toLocaleString** (Método): Devuelve una cadena con la configuración regional actual.
- **toString** (Método): Devuelve una representación de cadena de una matriz.
- **valueOf** (Método): Obtiene una referencia a la matriz.

© JMA 2020. All rights reserved

Date

- **Date.now** (Función): Devuelve el número de milisegundos que hay entre el 1 de enero de 1970 y la fecha y hora actuales.
- **Date.parse** (Función): Analiza una cadena que contiene una fecha y devuelve el número de milisegundos transcurridos entre esa fecha y la medianoche del 1 de enero de 1970.
- **Date.UTC** (Función): Devuelve el número de milisegundos transcurrido entre la medianoche del 1 de enero de 1970 en el horario universal coordinado (UTC) (o GMT) y la fecha proporcionada.
- **toString** (Método): Devuelve una representación en forma de cadena de un objeto.
- **toDateString** (Método): Devuelve una fecha como un valor de cadena.
- **toTimeString** (Método): Devuelve una hora como un valor de cadena.
- **toLocaleString** (Método): Devuelve un objeto convertido en cadena usando la configuración regional actual.
- **toLocaleDateString** (Método): Devuelve una fecha como un valor de cadena apropiado para la configuración regional actual del entorno host.
- **toLocaleTimeString** (Método): Devuelve una hora como un valor de cadena apropiado para la configuración regional actual del entorno host.
- **toISOString** (método): Devuelve una fecha como un valor alfanumérico en formato ISO.
- **toJSON** (método): Se utiliza para transformar datos de un tipo de objeto antes de la serialización JSON.
- **valueOf** (Método): Devuelve el valor primitivo del objeto especificado.

© JMA 2020. All rights reserved

Date

- **getTime** (Método): Devuelve el valor de tiempo en un objeto Date en milisegundos desde la medianoche del 1 de enero de 1970.
- **getFullYear, getUTCFullYear** (Método): Devuelve el valor de año usando la hora local o UTC.
- **getMonth, getUTCMonth** (Método): Devuelve el valor de mes usando la hora local o UTC.
- **getDate, getUTCDate** (Método): Devuelve el valor de día del mes usando la hora local o UTC.
- **getDay, getUTCDay** (Método): Devuelve el valor de día de la semana usando la hora local o UTC.
- **getHours, getUTCHours** (Método): Devuelve el valor de horas usando la hora local o UTC.
- **getMinutes, getUTCMinutes** (Método): Devuelve el valor de minutos usando la hora local o UTC.
- **getSeconds, getUTCSeconds** (Método): Devuelve el valor de segundos usando la hora local o UTC.
- **getMilliseconds, getUTCMilliseconds** (Método): Devuelve el valor de milisegundos usando la hora local o UTC.
- **getTimezoneOffset** (Método): Devuelve la diferencia en minutos entre la hora del equipo host y la hora universal coordinada (UTC).

© JMA 2020. All rights reserved

Date

- **setTime** (Método): Establece el valor de fecha y hora en el objeto Date.
- **setMilliseconds** (Método): Establece el valor de milisegundos usando la hora local.
- **setUTCMilliseconds** (Método): Establece el valor de milisegundos usando la hora UTC.
- **setSeconds** (Método): Establece el valor de segundos usando la hora local.
- **setUTCSeconds** (Método): Establece el valor de segundos usando la hora UTC.
- **setMinutes** (Método): Establece el valor de minutos usando la hora local.
- **setUTCMinutes** (Método): Establece el valor de minutos usando la hora UTC.
- **setHours** (Método): Establece el valor de horas usando la hora local.
- **setUTCHours** (Método): Establece el valor de horas usando la hora UTC.
- **setDate** (Método): Establece el día del mes numérico usando la hora local.
- **setUTCDate** (Método): Establece el día numérico del mes usando la hora UTC.
- **setMonth** (Método): Establece el valor de mes usando la hora local.
- **setUTCMonth** (Método): Establece el valor de mes usando la hora UTC.
- **setFullYear** (Método): Establece el valor de año usando la hora local.
- **setUTCFullYear** (Método): Establece el valor de año usando la hora UTC.

© JMA 2020. All rights reserved

Number

- **Number.MAX_VALUE**: El número más grande que se puede representar en JavaScript. Igual a aproximadamente 1,79E+308.
- **Number.MIN_VALUE**: El número más cercano a cero que se puede representar en JavaScript. Igual a aproximadamente 5,00E-324.
- **Number.NaN**: Un valor que no es un número.
- **Number.NEGATIVE_INFINITY**: Un valor inferior al número negativo más grande que se puede representar en JavaScript.
- **Number.POSITIVE_INFINITY**: Un valor superior al número más grande que se puede representar en JavaScript.
- **toExponential** (Método): Devuelve una cadena que contiene un número representado en notación exponencial.
- **toFixed** (Método): Devuelve una cadena que representa un número en notación de punto fijo.
- **toLocaleString** (Método): Devuelve un objeto convertido en una cadena basándose en la configuración regional actual.
- **toPrecision** (Método): Devuelve una cadena que contiene un número representado en notación exponencial o de punto fijo y que tiene un número especificado de dígitos.
- **toString** (Método): Devuelve una representación en forma de cadena de un objeto.
- **valueOf** (Método): Devuelve el valor primitivo del objeto especificado.

© JMA 2020. All rights reserved

String

- **length** (Propiedad): Devuelve la longitud de un objeto String.
- **String.fromCharCode** (Función): Devuelve una cadena a partir de varios valores de caracteres Unicode.
- **charAt** (Método): Devuelve el carácter que se encuentra en el índice especificado.
- **charCodeAt** (Método): Devuelve la codificación Unicode del carácter que se especifique.
- **concat** (Método): Devuelve una cadena que contiene la concatenación de las dos cadenas proporcionadas.
- **indexOf** (Método): Devuelve la posición del carácter donde tiene lugar la primera repetición de una subcadena dentro de una cadena.
- **lastIndexOf** (Método): Devuelve la última repetición de una subcadena dentro de una cadena.
- **localeCompare** (Método): Devuelve un valor que indica si dos cadenas son equivalentes en la configuración regional actual.
- **match** (Método): Busca una cadena mediante un objeto Regular Expression proporcionado y devuelve los resultados como una matriz.
- **replace** (Método): Usa una expresión regular para reemplazar texto en una cadena y devuelve el resultado.
- **search** (Método): Devuelve la posición de la primera coincidencia de subcadena en una búsqueda de expresión regular.
- **slice** (Método): Devuelve una sección de una cadena.

© JMA 2020. All rights reserved

String

- **split** (Método): Devuelve la matriz de cadenas resultante de la separación de una cadena en subcadenas.
- **substring** (Método): Devuelve la subcadena en la ubicación especificada dentro de un objeto String.
- **toLocaleLowerCase** (Método): Devuelve una cadena en la que todos los caracteres alfabéticos se convierten a minúsculas, según la configuración regional actual del entorno de host.
- **toLocaleString** (Método): Devuelve un objeto convertido en cadena usando la configuración regional actual.
- **toLocaleUpperCase** (Método): Devuelve una cadena en la que todos los caracteres alfabéticos se convierten a mayúsculas, según la configuración regional actual del entorno de host.
- **toLowerCase** (Método): Devuelve una cadena en la que todos los caracteres alfabéticos se convierten a minúsculas.
- **toString** (Método): Devuelve la cadena.
- **toUpperCase** (Método): Devuelve una cadena en la que todos los caracteres alfabéticos se convierten a mayúsculas.
- **trim** (Método): Devuelve una cadena donde se han quitado los caracteres de espacio en blanco iniciales y finales y los caracteres de terminador de línea.
- **valueOf** (Método): Devuelve la cadena.

© JMA 2020. All rights reserved

RegEx

- Crea un objeto 'expresión regular' para encontrar texto de acuerdo a un patrón.
- Las cadenas delimitadas por / generan un objeto RegEx con el patrón contenido en la cadena:

```
var re = new RegExp("^\\d{1,8}[A-Z]$");  
var re = /^\\d{1,8}[A-Z]$/;
```
- Permiten los modificadores:
 - g: búsqueda global (no para tras la primera coincidencia)
 - i: ignorar mayúsculas o minúsculas
 - m: tratar caracteres de inicio y fin (^ y \$) como múltiples líneas de texto
- Las expresiones regulares se utilizan con los métodos de RegEx (test y exec) y con los métodos de String (match, replace, search y split).

© JMA 2020. All rights reserved

RegEx: Escapes de carácter

Carácter de escape	Descripción	Modelo	Coincidencias
\a	Coincide con un carácter de campana, \u0007.	\a	"\u0007" en "Error!" + "\u0007"
\b	En una clase de caracteres, coincide con un retroceso, \u0008.	[b]{3,}	"b\b\b\b" en "b\b\b\b"
\t	Coincide con una tabulación, \u0009.	(\w+)\t	"artículo1\t", "artículo2\t" en "artículo1\tartículo2\t"
\r	Coincide con un retorno de carro, \u000D.(\r no es equivalente al carácter de nueva línea, \n.)	\r\n(\w+)	"\r\nEstas" en "\r\nEstas son\n\ndos líneas."
\v	Coincide con una tabulación vertical, \u000B.	[\v]{2,}	"\v\v\v" en "\v\v\v"
\f	Coincide con un avance de página, \u000C.	[\f]{2,}	"\f\f\f" en "\f\f\f"
\n	Coincide con una nueva línea, \u000A.	\r\n(\w+)	"\r\nEstas" en "\r\nEstas son\n\ndos líneas."
\e	Coincide con un escape, \u001B.	\e	"\x001B" en "\x001B"
\nnn	Usa la representación octal para especificar un carácter (nnn consta de tres dígitos como máximo).	\w[040]\w	"a b", "c d" en "a bc d"
\xnn	Usa la representación hexadecimal para especificar un carácter (nn consta de exactamente dos dígitos).	\w\x20\w	"a b", "c d" en "a bc d"
\cX	Se corresponde con el carácter de control ASCII especificado por X, donde X es la letra del carácter de control.	\cC	"\x0003" en "\x0003" (Ctrl-C)
\unnnn	Se corresponde con un carácter Unicode usando la representación hexadecimal (exactamente cuatro dígitos, representados aquí por nnnn).	\w\u0020\w	"a b", "c d" en "a bc d"
\	Cuando va seguido de un carácter sin significado, coincide con ese carácter	[d+[+-x*]]d+[d+[+-x*]]d+	"2+2" y "3*9" en "(2+2) * 3*9"

© JMA 2020. All rights reserved

RegEx: Clases de carácter

Clase de carácter	Descripción	Modelo	Coincidencias
[grupo_caracteres]	Coincide con cualquier carácter único de grupo_caracteres. De forma predeterminada, la coincidencia distingue entre mayúsculas y minúsculas.	[aeiou]	"a" en "casa" "a", "e" en "ave"
[^grupo_caracteres]	Negación: coincide con cualquier carácter individual que no esté en grupo_caracteres. De forma predeterminada, los caracteres de grupo_caracteres distinguen entre mayúsculas y minúsculas.	[^aei]	"r", "n", "o" en "reino"
[primero-último]	Intervalo de caracteres: coincide con cualquier carácter individual en el intervalo de primero a último.	[A-Z]	"A", "B" en "AB123"
.	Carácter comodín: coincide con cualquier carácter excepto con \n.	a.e	"ave" en "llave" "ate" en "yate"
\p{name}	Coincide con cualquier carácter único que pertenezca a la categoría general Unicode o al bloque con nombre especificado por nombre.	\p{Lu} \p{IsCyrillic}	"C", "L" en "City Lights" "Д", "Ж" in "ДЖем"
\P{name}	Coincide con cualquier carácter único que no pertenezca a la categoría general Unicode o al bloque con nombre especificado por nombre.	\P{Lu} \P{IsCyrillic}	"i", "t", "y" en "City" "e", "m" in "ДЖем"
\w	Coincide con cualquier carácter de una palabra.	\w	"I", "D", "A", "1", "3" en "ID A1.3"
\W	Coincide con cualquier carácter que no pertenezca a una palabra.	\W	" . . ." en "ID A1.3"
\s	Coincide con cualquier carácter que sea un espacio en blanco.	\w\s	"D " en "ID A1.3"
\S	Coincide con cualquier carácter que no sea un espacio en blanco.	\s\S	" . ." en "int_ctr"
\d	Coincide con cualquier dígito decimal.	\d	"4" en "4 = IV"
\D	Coincide con cualquier dígito que no sea decimal.	\D	" .", "-", " .", "I", "V" en "4 = IV"

© JMA 2020. All rights reserved

RegEx: Delimitadores

Aserción	Descripción	Modelo	Coincidencias
^	La coincidencia debe comenzar al principio de la cadena o de la línea.	<code>^d{3}</code>	"901-" en "901-333-"
\$	La coincidencia se debe producir al final de la cadena o antes de <code>\n</code> al final de la línea o de la cadena.	<code>-d{3}\$</code>	"-333" en "-901-333"
\A	La coincidencia se debe producir al principio de la cadena.	<code>\Ad{3}</code>	"901-" en "901-333-"
\Z	La coincidencia se debe producir al final de la cadena o antes de <code>\n</code> al final de la cadena.	<code>-d{3}\Z</code>	"-333" en "-901-333"
\z	La coincidencia se debe producir al final de la cadena.	<code>-d{3}\z</code>	"-333" en "-901-333"
\G	La coincidencia se debe producir en el punto en el que finalizó la coincidencia anterior.	<code>\G\d{3}</code>	"(1)", "(3)", "(5(" en "(1)(3)(5)(7)(9)"
\b	La coincidencia se debe producir en un límite entre un carácter <code>\w</code> (alfanumérico) y un carácter <code>\W</code> (no alfanumérico).	<code>\bw+ s w+ b</code>	"ellos ellos" en "ellos tema ellos ellos"
\B	La coincidencia no se debe producir en un límite <code>\b</code> .	<code>\Bend w* b</code>	"fin", "final" en "finalizar finalista finalizador finalizó"

© JMA 2020. All rights reserved

RegEx: Construcciones de agrupamiento

Construcción de agrupamiento	Descripción	Modelo	Coincidencias
(subexpresión)	Captura la subexpresión coincidente y le asigna un número ordinal de base cero.	<code>(\w)\1</code>	"aa" en "aaron"
(?<name>subexpresión)	Captura la subexpresión coincidente en un grupo con nombre.	<code>{<double>\w}\k<double></code>	"aa" en "aaron"
(?<nombre1-nombre2>subexpresión)	Define una definición de grupo de equilibrio.	<code>(((?<Open>\{)[^\{\}]*+(?<Close>Open>\})[^\{\}]*+)*(?(Open){})\$</code>	"((1-3)*(3-1))" en "3+2^((1-3)*(3-1))"
(?:subexpresión)	Define un grupo sin captura.	<code>Write(?:Line)?</code>	"WriteLine" en "Console.WriteLine()"
(?imnsx-imnsx:subexpresión)	Aplica o deshabilita las opciones especificadas dentro de subexpresión.	<code>A\d{2}(?:i w+) b</code>	"A12xl", "A12XL" en "A12xl A12XL a12xl"
(?=subexpresión)	Aserción de búsqueda anticipada positiva de ancho cero.	<code>\w+(?=.)</code>	"es", "corría" y "hermoso" en "Él es. El perro corría. El sol está hermoso."
(?!subexpresión)	Aserción de búsqueda anticipada negativa de ancho cero.	<code>\b(?:!un) w+ b</code>	"seguro", "usado" en "aseguro seguro unidad usado"
(?<=subexpresión)	Aserción de búsqueda tardía positiva de ancho cero.	<code>(?<=19)\d{2} b</code>	"99", "50", "05" en "1851 1999 1950 1905 2003"
(?< subexpresión)	Aserción de búsqueda tardía negativa de ancho cero.	<code>(?< 19)\d{2} b</code>	"51", "03" en "1851 1999 1950 1905 2003"
(?>subexpresión)	Subexpresión sin retroceso (o "expansiva").	<code>[13579](?>A+B+)</code>	"1ABB", "3ABB" y "5AB" en "1ABB 3ABBC 5AB 5AC"

© JMA 2020. All rights reserved

RegEx: Cuantificadores

Cuantificador	Descripción	Modelo	Coincidencias
*	Coincide con el elemento anterior cero o más veces.	\d*.\d	".0", "19.9", "219.9"
+	Coincide con el elemento anterior una o más veces.	"be+"	"ca!" en "caída", "be" en "bebé"
?	Coincide con el elemento anterior cero veces o una vez.	"rai?n"	"rata", "raicilla"
{n}	Coincide con el elemento anterior exactamente n veces.	".\d{3}"	".043" en "1,043.6", ",876", ",543", y ",210" en "9,876,543,210"
{n,}	Coincide con el elemento anterior al menos n veces.	".\d{2,}"	"166", "29", "1930"
{n,m}	Coincide con el elemento anterior al menos n veces, pero no más de m veces.	".\d{3,5}"	"166", "17668", "19302" en "193024"
?	Coincide con el elemento anterior cero o más veces, pero el menor número de veces que sea posible.	\d?.\d	".0", "19.9", "219.9"
++?	Coincide con el elemento anterior una o más veces, pero el menor número de veces que sea posible.	"be++?"	"ca!" en "caída", "be" en "bebé"
??	Coincide con el elemento anterior cero o una vez, pero el menor número de veces que sea posible.	"rai??n"	"rata", "raicilla"
{n}?	Coincide con el elemento anterior exactamente n veces.	".\d{3}?"	".043" en "1,043.6", ",876", ",543", y ",210" en "9,876,543,210"
{n,}?	Coincide con el elemento anterior al menos n veces, pero el menor número de veces posible.	".\d{2,}?"	"166", "29", "1930"
{n,m}?	Coincide con el elemento anterior entre n y m veces, pero el menor número de veces posible.	".\d{3,5}?"	"166", "17668", "19302" en "193024"

© JMA 2020. All rights reserved

RegEx: Construcciones de alternancia

Construcciones de alternancia	Descripción	Modelo	Coincidencias
	Coincide con cualquier elemento separado por el carácter de barra vertical ().	th(e is at)	"el", "este" en "este es el día."
(?(expresión)sí no)	Coincide con sí si expresión coincide; de lo contrario, coincide con la parte opcional no. expresión se interpreta como una aserción de ancho cero.	(?(A)\d{2}\b \b\d{3}\b)	"A10", "910" en "A10 C103 910"
(?(name)sí no)	Coincide con sí si la captura con nombre nombre tiene una coincidencia; de lo contrario, coincide con la parte opcional no.	(?<quoted>)?(? (quoted).+?" \S +\s)	Dogs.jpg, "Yiska playing.jpg" en "Dogs.jpg "Yiska playing.jpg"

© JMA 2020. All rights reserved

RegEx: Sustituciones

Carácter	Descripción	Modelo	Modelo de reemplazo	Cadena de entrada	Cadena de resultado
\$number	Sustituye la subcadena que coincide con el grupo número.	/b(/w+)(/s)(/w+)/b	\$3\$2\$1	"one two"	"two one"
\${name}	Sustituye la subcadena que coincide con el grupo nombre.	\b(?<word1>\w+)(\s){?<word2>\w+)\b	\${word2} \${word1}	"one two"	"two one"
\$\$	Sustituye un "\$" literal.	\b(\d+)\s?USD	\$\$1	"103 USD"	"\$103"
\$&	Sustituye una copia de toda la coincidencia.	(\\$(\d*(\.\d+)?)\{1})	**\$&	"\$1.30"	***\$1.30***
\$`	Sustituye todo el texto de la cadena de entrada delante de la coincidencia.	B+	\$`	"AABBCC"	"AAAACC"
\$'	Sustituye todo el texto de la cadena de entrada detrás de la coincidencia.	B+	\$'	"AABBCC"	"AACCCC"
\$+	Sustituye el último grupo capturado.	B+(C+)	\$+	"AABBCCDD"	AACCCDD
\$_	Sustituye toda la cadena de entrada.	B+	\$_	"AABBCC"	"AAAABBBCCC"

© JMA 2020. All rights reserved

JSON

- **JSON.parse** (Función): Convierte una cadena de la notación de objetos de JavaScript (JSON) en un objeto.
- **JSON.stringify** (Función): Convierte un valor de JavaScript en una cadena de la notación de objetos JavaScript (JSON).

© JMA 2020. All rights reserved

Math

- **Math.E**: Constante matemática e. Es el número de Euler, base de los logaritmos naturales.
- **Math.LN2**: Logaritmo natural de 2.
- **Math.LN10**: Logaritmo natural de 10.
- **Math.LOG2E**: Logaritmo de base 2 de e.
- **Math.LOG10E**: Logaritmo de base 10 de e.
- **Math.PI**: Pi. Es la proporción entre la circunferencia de un círculo y su diámetro.
- **Math.SQRT1_2**: Raíz cuadrada de 0,5, o, de forma equivalente, uno dividido por la raíz cuadrada de 2.
- **Math.SQRT2**: Raíz cuadrada de 2.
- **Math.abs** (Función): Devuelve el valor absoluto de un número.
- **Math.acos** (Función): Devuelve el arco coseno de un número.
- **Math.asin** (Función): Devuelve el arcoseno de un número.
- **Math.atan** (Función): Devuelve el arco tangente de un número.
- **Math.atan2** (Función): Devuelve el ángulo, en radianes, desde el eje X a un punto representado por las coordenadas x e y proporcionadas.
- **Math.cos** (Función): Devuelve el coseno de un número.

© JMA 2020. All rights reserved

Math

- **Math.ceil** (Función): Devuelve el entero más pequeño que sea mayor o igual que la expresión numérica proporcionada.
- **Math.exp** (Función): Devuelve e (base de los logaritmos naturales) elevado a una potencia.
- **Math.floor** (Función): Devuelve el entero más grande que sea menor o igual que la expresión numérica proporcionada.
- **Math.log** (Función): Devuelve el logaritmo natural de un número.
- **Math.max** (Función): Devuelve la mayor de dos expresiones numéricas proporcionadas.
- **Math.min** (Función): Devuelve el menor de dos números proporcionados.
- **Math.pow** (Función): Devuelve el valor de una expresión base elevada a una potencia especificada.
- **Math.random** (Función): Devuelve un número pseudoaleatorio entre 0 y 1.
- **Math.round** (Función): Devuelve una expresión numérica especificada redondeada al entero más cercano.
- **Math.sin** (Función): Devuelve el seno de un número.
- **Math.sqrt** (Función): Devuelve la raíz cuadrada de un número.
- **Math.tan** (Función): Devuelve la tangente de un número.

© JMA 2020. All rights reserved

Nuevos Objetos (ES6)

- **Symbol:** Permite crear un identificador único.
- **Promise:** Proporciona un mecanismo para programar el trabajo de modo que se lleve a cabo en un valor que todavía no se calculó.
- **Proxy:** Habilita el comportamiento personalizado de un objeto.
- **Reflect:** Proporciona métodos para su uso en las operaciones que se interceptan.
- **Intl.Collator:** Proporciona comparaciones de cadenas de configuración regional.
- **Intl.DateTimeFormat:** Proporciona formato de fecha y hora específico de la configuración regional.
- **Intl.NumberFormat:** Proporciona formato de número específico de la configuración regional.

- **Map:** Lista de pares clave-valor.
- **Set:** Colección de valores únicos que pueden ser de cualquier tipo.
- **WeakMap:** Colección de pares clave-valor en los que cada clave es una referencia de objeto.
- **WeakSet:** Colección de objetos únicos.

© JMA 2020. All rights reserved

Nuevos Objetos (ES6)

- **ArrayBuffer:** Representa un búfer sin formato de datos binarios, que se usa para almacenar datos de las diferentes matrices con tipo. No se puede leer directamente de ArrayBuffer ni escribir directamente en ArrayBuffer, pero se puede pasar a una matriz con tipo o un objeto DataView para interpretar el búfer sin formato según sea necesario.
- **DataView:** Se usa para leer y escribir diferentes tipos de datos binarios en cualquier ubicación de ArrayBuffer.
- **Float32Array:** Matriz con tipo de valores flotantes de 32 bits.
- **Float64Array:** Matriz con tipo de valores flotantes de 64 bits.
- **Int8Array:** Matriz con tipo de valores enteros de 8 bits.
- **Int16Array:** Matriz con tipo de valores enteros de 16 bits.
- **Int32Array:** Matriz con tipo de valores enteros de 32 bits.
- **Uint8Array:** Matriz con tipo de valores enteros sin signo de 8 bits.
- **Uint8ClampedArray:** Matriz con tipo de enteros sin signo de 8 bits con valores fijos.
- **Uint16Array:** Matriz con tipo de valores enteros sin signo de 16 bits.
- **Uint32Array:** Matriz con tipo de valores enteros sin signo de 32 bits.

© JMA 2020. All rights reserved

Symbol (ES6)

- Por especificación, las claves (Keys) de un objeto deben ser solamente del tipo String o Symbol. El valor de "Symbol" representa un identificador único.

```
let id = Symbol();  
let id = Symbol("id"); // es un symbol con la descripción "id"
```
- Se garantiza que los símbolos son únicos. Aunque declaremos varios Symbols con la misma descripción, éstos tendrán valores distintos. La descripción es solamente una etiqueta que no afecta nada más. Los Symbols no se auto convierten a String, hay que hacerlo explícitamente con `.description` o `.toString()`
- Los Symbols nos permiten crear claves "ocultas" en un objeto, a las cuales ninguna otra parte del código puede acceder ni sobrescribir.

```
let user = {  
  [id]: 123 // no "id": 123  
  name: "John",  
}  
user[id] = "Su id";
```

© JMA 2020. All rights reserved

Promise Pattern (ES6)

- El Promise Pattern es un patrón de organización de código que permite encadenar llamadas a métodos que se ejecutaran a la conclusión del anterior (flujos).
- Simplifica y soluciona los problemas comunes con el patrón Callback:
 - Llamadas anidadas
 - Complejidad de código

```
o.m(1, 2, f(m1(3, f1(4,5,ff(8)))) → o.m(1, 2).f().m1(3).f1(4, 5).ff(8)
```
- Aunque se utiliza extensamente para las operaciones asíncronas, no es exclusivo de las mismas.
- Las promesas se han incorporado a los objetos estándar de JavaScript en la versión 6.

© JMA 2020. All rights reserved

Objeto Promise (ES6)

- Una “promesa” es un objeto que actúa como proxy en los casos en los que no se puede utilizar el verdadero valor porque aún no se conoce (no se ha generado, llegado, ...) pero se debe continuar sin esperar a que este disponible (no se puede bloquear la función esperando a su obtención).
- Una “promesa” puede tener los siguientes estados:
 - Pendiente: Aún no se sabe si se podrá o no obtener el resultado.
 - Resuelta: Se ha podido obtener el resultado (Promise.resolve())
 - Rechazada: Ha habido algún tipo de error y no se ha podido obtener el resultado (Promise.reject())
- Los métodos del objeto promesa devuelven al propio objeto para permitir apilar llamadas sucesivas.
- Como objeto, la promesa se puede almacenar en una variable, pasar como parámetro o devolver desde una función, lo que permite aplicar los métodos en distintos puntos del código.

© JMA 2020. All rights reserved

Crear promesas (ES6)

- El objeto Promise gestiona la creación de la promesa y los cambios de estados de la misma.

```
list() {  
  return new Promise((resolve, reject) => {  
    this.http.get(this.baseUrl).subscribe(  
      data => resolve(data),  
      err => reject(err)  
    )  
  });  
}
```
- Para crear promesas ya concluidas:
 - Promise.resolve: Crea una promesa nueva como resuelta cuyo resultado es igual que su argumento.
 - Promise.reject: Crea una promesa nueva como rechazada cuyo resultado es igual que el argumento pasado.

© JMA 2020. All rights reserved

Invocar promesas (ES6)

- El objeto Promise creado expone los métodos:
 - `then(fnResuelta, fnRechazada)`: Recibe como parámetro la función a ejecutar cuando termine la anterior y, opcionalmente, la función a ejecutar en caso de que falle la anterior.
 - `catch(fnRechazada)`: Recibe como parámetro la función a ejecutar en caso de que falle. `list().then(calcular, ponError).then(guardar)`
 - `finally(fnResueltaOrRechazada)`: Recibe como parámetro la función a ejecutar tanto en caso de que se resuelva o se rechace (ES2018).
- Otras formas de crear e invocar promesas son:
 - `Promise.all`: Combina dos o más promesas y realiza la devolución solo cuando todas las promesas especificadas se completan o alguna sea rechaza.
 - `Promise.race`: Crea una nueva promesa que resolverá o rechazará con el mismo valor de resultado que la primera promesa que se va resolver o rechazar entre los argumentos pasados.

© JMA 2020. All rights reserved

DETECCIÓN Y CORRECCIÓN DE ERRORES

© JMA 2020. All rights reserved

Depuración

- La mayoría de los navegadores modernos ofrecen utilidades de desarrollo y depuración.
- Cada depurador ofrece:
 - Un editor multi-línea para experimentar con JavaScript;
 - Un inspector para revisar el código generado en la página;
 - Un visualizador de red o recursos, para examinar las peticiones que se realizan.
- Suelen suministrar un objeto console con los siguientes métodos:
 - `console.log()` para enviar y registrar mensajes generales.
 - `console.dir()` para registrar un objeto y visualizar sus propiedades.
 - `console.warn()` para registrar mensajes de alerta.
 - `console.error()` para registrar mensajes de error.
- Existen otros métodos para utilizar desde la consola, pero estos pueden variar según el navegador. La consola además provee la posibilidad de establecer puntos de interrupción y observar expresiones en el código con el fin de facilitar su depuración.

© JMA 2020. All rights reserved

NUEVAS VERSIONES ECMAScript

© JMA 2020. All rights reserved

Introducción

- A partir de la versión 6 cambio la política de versionado y sacan una revisión anual en Junio con los pocos cambios hasta ese momento: ECMAScript 2016 (ES7), ECMAScript 2017 (ES8), ECMAScript 2018 (ES9), ...
- Las nuevas versiones se van extendiendo progresivamente, los navegadores deben dar soporte pero los usuarios deben actualizar sus navegadores en caso de que sea posible, esto es un proceso lento que requiere su tiempo. En NodeJS es mas ágil.
- Las alternativas para empezar a utilizarlo son:
 - Transpiladores ("Transpiler": "Translator" y "Compiler"): Traducen o compilan un lenguaje de alto nivel a otro lenguaje de alto nivel, en este caso código ES6 a ES5. Los más conocidos y usados son Babel, TypeScript, Google Traceur, CoffeeScript.
 - Polyfill: Un fragmento de código o un plugin que permite tener las nuevas funcionalidades de HTML o ES en aquellos navegadores que nativamente no lo soportan. ([core-js](#), [polyfill.io](#), ...)

© JMA 2020. All rights reserved

EcmaScript 6

- <http://www.ecma-international.org/ecma-262/6.0/>
- <http://es6-features.org/>
- <https://kangax.github.io/compat-table/es2016plus/>
- <https://babeljs.io/>
- <https://github.com/google/traceur-compiler>
- <https://www.typescriptlang.org/>
- <https://github.com/zloirock/core-js>
- <https://es.javascript.info/>

© JMA 2020. All rights reserved

EcmaScript 2016 (ES7)

- El operador exponencial (**):
 - `x = Math.pow(3, 2);`
 - `x = 3 ** 2;`
 - `x = 3; x **= 2;`
- Método `Array.prototype.includes()`
 - `if(listado.indexOf(item) !== -1) // lo contiene`
 - `if(listado.includes(item)) // lo contiene`

© JMA 2020. All rights reserved

EcmaScript 2017 (ES8)

- `Object.values()`: devuelve un array con los valores correspondientes a las propiedades enumerables de un objeto.
- `Object.entries()`: devuelve una matriz de arrays `[key, value]` del objeto dado.
- `Object.getOwnPropertyDescriptors()`: devuelve todos los descriptores de propiedad propios de un objeto dado para su clonado.
 - `value`: El valor asociado con la propiedad (solo descriptores de datos).
 - `writable`: true si y solo si el valor asociado con la propiedad puede ser cambiado (solo descriptores de datos).
 - `get`: Una función que sirve como un getter para la propiedad, o undefined si no hay getter (solo descriptores de acceso).
 - `set`: Una función que sirve como un setter para la propiedad, o undefined si no hay setter (solo descriptores de acceso).
 - `configurable`: true si y solo si el tipo de este descriptor de propiedad puede ser cambiado y si la propiedad puede ser borrada de el objeto correspondiente.
 - `enumerable`: true si y solo si esta propiedad aparece durante la enumeración de las propiedad en el objeto correspondiente.
- `str.padStart()` y `str.padEnd()` rellenan la cadena actual, por el principio o por el final, con la cadena dada (repitiéndola si es necesaria) para que la cadena resultante alcance una longitud dada, si aun no la tiene.
- Ahora permite tener comas finales después del último parámetro de función.

© JMA 2020. All rights reserved

async/await (ES8)

- La declaración de función `async` define una función asíncrona, que devuelve un objeto `AsyncFunction`. Una función asíncrona es una función que opera asincrónicamente a través del bucle de eventos, utilizando una promesa implícita para devolver su resultado. Pero la sintaxis y la estructura de su código usando funciones asíncronas se parece mucho más a las funciones síncronas estándar.
- El operador `await` se usa para esperar a una `Promise` y sólo dentro de una `async function`.

```
function resolveAfter2Seconds(x) {  
  return new Promise(resolve => { setTimeout(() => { resolve(x); }, 2000); });  
}  
async function f1() {  
  var x = await resolveAfter2Seconds(10);  
  console.log(x); // 10  
}
```

© JMA 2020. All rights reserved

Memoria compartida con acceso atómico (ES8)

- Cuando dos threads tienen acceso a un espacio de memoria compartido, es importante que las operaciones de escritura o lectura sean atómicas, es decir, que se hagan en bloque y aisladas de otros threads, para garantizar la validez de los datos.
- Esta funcionalidad está pensada para web workers, ya que es el único caso en el se puede implementar multithreading en Javascript, y de momento está orientada a muy bajo nivel: `ArrayBuffers` (arrays de datos binarios).
- El mecanismo consta de:
 - `SharedArrayBuffer`: Un nuevo constructor para crear `ArrayBuffers` de memoria compartida.
 - La clase `Atomic`, con métodos estáticos para acceder/manipular ese `SharedArrayBuffer` de forma atómica.

© JMA 2020. All rights reserved

EcmaScript 2018 (ES9)

- ES6 incorporaba el spread operator y los parámetros rest para hacer asignaciones por destructuring con Arrays.

```
const primes = [2, 3, 5, 7, 11];  
const [first, second, ...rest] = primes;  
console.log(rest); // [5, 7, 11]  
const primes2 = [first, second, ...rest];  
console.log(primes2); // [2, 3, 5, 7, 11]
```
- ES9 permite hacer asignaciones por destructuring con objetos.

```
const person = { firstName: 'Peter', lastName: 'Parker', age: 26, }  
const { age, ...name } = person;  
console.log(name); // {firstName: "Peter", lastName: "Parker"}  
const person2 = { age, ...name };  
console.log(person2); // {age: 26, firstName: "Peter", lastName: "Parker"}
```
- `promesa.finally`: permite llamar siempre a una función de callback sin parámetros al completarse (resuelta o rechazada).
- For asíncrono: `for await` (`const line of readLines(filePath)`) {

© JMA 2020. All rights reserved

EcmaScript 2018 (ES9)

- Mejoras en las expresiones regulares:
 - Named capture groups

```
let re = /(?!<year>\d{4})-(?!<month>\d{2})-(?!<day>\d{2})/u;  
let result = re.exec('2015-01-02');  
// result.groups.year === '2015';  
// result.groups.month === '01';  
// result.groups.day === '02';
```
 - Comprobar si la expresión esta precedida de un determinado patrón sin incluirla en el resultado:

```
const regex = /(?!<=€)([0-9])+/;  
let result = regex.test('€2000'); // result === 2000
```
 - El indicado `/s` evita los problemas con las secuencias de escape del `.` como comodín:

```
/hola.mundo/.test('hola\tmlundo'); //false  
/hola.mundo/s.test('hola\tmlundo'); //true
```

© JMA 2020. All rights reserved

EcmaScript 2019 (ES10)

- `Object.fromEntries()`: transforma una lista de arrays [clave-valor] en un objeto, proceso contrario al método `entries()`

```
const arr = [{"firstName", "Peter"}, {"lastName", "Parker"}, {"age", 26}];  
Object.fromEntries(arr) // = { firstName: 'Peter', lastName: 'Parker', age: 26 }
```
- `Array.prototype.flat()`: permite aplanar un array multidimensional por niveles.

```
const arr = ['1', '2', ['3.1', '3.2'], '4', ['5.1', ['5.2.1', '5.2.2']], '6'];  
arr.flat();    // ["1", "2", "3.1", "3.2", "4", "5.1", Array(2), "6"]  
arr.flat(2);   // ["1", "2", "3.1", "3.2", "4", "5.1", "5.2.1", "5.2.2", "6"]
```
- `Array.prototype.flatMap()`: combina el método `map(callback)` con el `flat()` para transformar y aplanar un array de forma eficiente.

```
const arr = [[1, 2], [3, 4], [5, 6]];  
arr.map(([a, b]) => [a, a * b]) // [[1, 2], [3, 12], [5, 30]]  
    .flat() // [1, 2, 3, 12, 5, 30]  
arr.flatMap(([a, b]) => [a, a * b]) // [1, 2, 3, 12, 5, 30]
```

© JMA 2020. All rights reserved

EcmaScript 2019 (ES10)

- `String.trimStart()`, `String.trimEnd()`: eliminan los espacios en blanco al principio o del final de un string. Por consistencia con su proceso contrario `padStart` y `padEnd`, se ha mantenido la nomenclatura pero se añaden también como alias `trimLeft` y `trimRight` por compatibilidad con otros.
- `Symbol.description`: Obtiene la cadena asociada al símbolo.
- `Function.prototype.toString()` ahora devuelve la cadena con el código fuente sin eliminar espacios en blanco, comentarios y saltos de línea.
- La captura del error en una referencia al declarar el `catch` ahora es opcional:

```
try {  
  :  
} catch { // catch(e)  
  :  
}
```
- `JSON.stringify()`, independiente del formato de entrada, debe devolver una cadena UTF-8 bien formada y `JSON.parse()` debe aceptar los símbolos de separador de línea (`\u2028`) y separador de párrafo (`\u2029`).

© JMA 2020. All rights reserved

ECMAScript 2020 (ES11)

- Operadores:
 - Encadenamiento opcional, cortocircuitado (?.):
 - $v?.p \rightarrow v === \text{null} ? \text{null} : v.p$
 - Selección de valores (||):
 - $v || \text{"default!"} \rightarrow v ? v : \text{"default!"}$
 - Coalescencia nula (??):
 - $v ?? \text{"default!"} \rightarrow (v === \text{null} || v === \text{undefined}) ? \text{"default!"} : v$
- El objeto `globalThis` establece una forma universal de acceder al objeto global (`this`) en JavaScript que anteriormente dependía del entorno, usando `window` o `self` en el navegador, o `global` en NodeJs.
- Ahora el orden de las propiedades está garantizado en el bucle `for in` o `JSON.stringify`.

© JMA 2020. All rights reserved

ECMAScript 2020 (ES11)

- Aparece el `BigInt` para superar algunas limitaciones de (2^{53}) de ciertas APIs cuando los ids numéricos son muy grandes (los ids usados por Twitter), y cuyo valor superaba el valor de `Number.MAX_SAFE_INTEGER`, obligando a tener que tratarlos como strings. Se pueden crear números enteros más grandes añadiendo una "n" al final del entero o mediante la función `BigInt`.
- `Strings.matchAll()`: genera un iterador con todos los objetos coincidentes generados por una expresión regular global.
- `Promise.allSettled`: Ejecuta una colección de promesas y devuelve el resultado de cada una de ellas para su tratamiento como resuelta o rechazada.

© JMA 2020. All rights reserved

ECMAScript 2020 (ES11)

- Importación dinámica de módulos como promesas:

```
import(`./my-modules/${ myModuleName }.js`)  
  .then(module => { module.doStuff(); })  
  .catch(err => console.log(err));  
(async () => {  
  const module = await import (`./my-modules/${myModuleName }.js`);  
  module2.doStuff();  
})();
```
- El objeto `import.meta` expone el contenido específico con los metadatos de módulo JavaScript.

```
<script type="module" src="my-module.js"></script>  
console.log(import.meta); // { url: "file:///home/user/my-module.js" }
```
- Propagación (importación y exportación):

```
export * as ns from 'module'
```

© JMA 2020. All rights reserved

ECMAScript 2021 (ES12)

- Separador numérico: permite separar los números con guiones bajo (`_`) sin afectar al valor (azúcar sintáctica): `100_000_000 === 100000000`
- Operador de asignación lógica: se incorporan los operadores de asignación lógica `&&=` `||=` y `??=`
- Métodos, getters y setters privados en las clases: con el prefijo `#` en su nombre verán su alcance limitado a la clase donde están definidos.

```
#private(val) { ... }  
set #width (width) { this.#width = width; }  
get width () { return this.#width ; }
```
- `String.replaceAll()` reemplaza todas las ocurrencias de una cadena por otra cadena (`replace()` solo reemplazaba la primera ocurrencia).
- `Promise.any()`, cuyo argumento es un array de promesas, será gestionado capturando la respuesta de la primera que sea resuelta de forma satisfactoria, o un array de excepciones en caso de que todas sean rechazadas

```
Promise.any([promise1, promise2, promise3])
```

© JMA 2020. All rights reserved

ECMAScript 2022 (ES13)

- **Await a nivel superior:** permite usar el `await` sin necesidad de estar contenido dentro de una función asíncrona (marcada con `async`).
- **Object.hasOwn():** devuelve `true` si el objeto especificado tiene la propiedad indicada como propiedad propia. Si la propiedad es heredada, o no existe, el método devuelve `false`.
- **Array.at():** recibe un valor numérico entero y devuelve el elemento en esa posición, permitiendo valores positivos y negativos. Los valores negativos contarán desde el último elemento del array.
`[1,2,3,4,5].at(-1) // devuelve 5`
- **error.cause:** permite especificar que causó un error.

```
try {
  connectToDatabase();
} catch (err) {
  throw new Error("Connecting to database failed.", { cause: err });
}
```

© JMA 2020. All rights reserved


ECMAScript 2022 (ES13)

- **Propiedades y métodos privados:** Se pueden crear atributos y métodos privados en las clases usando el prefijo `#` en el nombre.


```
class Usuario{
  #nombre;
  constructor(nombre) {
    this.#nombre = nombre;
  }
  #apellidos(apellido) {
    return this.#nombre + ' ' + apellido;
  }
  mostrar(apellido) {
    console.log(this.#apellidos(apellido));
  }
}

const usuario = new Usuario("Pepito");
console.log(usuario.#nombre);           // Error
console.log(usuario.#apellidos("Grillo")); // Error
console.log(usuario.mostrar("Grillo"));  // Pepito Grillo
```

© JMA 2020. All rights reserved



DOM (Document Object Model)



BOM (Browser Object Model)

© JMA 2020. All rights reserved

DOM

© JMA 2020. All rights reserved

Introducción a DOM

- El Modelo de Objetos del Documento es una interfaz independiente de la plataforma y del lenguaje que permite a programas y scripts acceder y actualizar dinámicamente los contenidos, la estructura y el estilo de los documentos.
- El Nivel 1 del Modelo de Objetos del Documento proporciona un conjunto estándar de objetos para representar documentos HTML y XML, un modelo estándar de cómo pueden combinarse estos objetos y una interfaz estándar para acceder a ellos y manipularlos.
- El Modelo de Objetos del Documento Nivel 2 Hojas de Estilo y Hojas de Estilo en Cascada (CSS) define una interfaz neutral para la plataforma y el lenguaje que permite a los programas y scripts acceder dinámicamente y actualizar el contenido de documentos de hojas de estilo.
- El Modelo de Objetos del Documento de nivel 3 especifica una plataforma y un interfaz de lenguaje neutral que permiten a programas y scripts acceder dinámicamente y actualizar el contenido, estructura y estilos del documento.
- Se encuentra normalizado a través de la recomendación del W3C:
 - http://www.w3.org/TR/#tr_DOM
- Es responsabilidad de los navegadores dar soporte a los diferentes niveles de las recomendaciones.

© JMA 2020. All rights reserved

Soporte del DOM

- La primera especificación del DOM (Document Object Model Level 1) se definió en 1998 y permitió homogeneizar la implementación del DHTML o HTML dinámico en los diferentes navegadores, ya que permitía modificar el contenido de las páginas web sin necesidad de recargar la página entera.
- Desafortunadamente, las posibilidades teóricas de DOM son mucho más avanzadas de las que se pueden utilizar en la práctica para desarrollar aplicaciones web.
- El motivo es que el uso de DOM siempre está limitado por las posibilidades que ofrece cada navegador.
- Mientras que algunos navegadores como Firefox y Safari implementan DOM de nivel 1 y 2 (y parte del 3), otros navegadores como Internet Explorer (versión 7 y anteriores) ni siquiera son capaces de ofrecer una implementación completa de DOM nivel 1.
- Con la aparición del HTML 5, esta situación se corregirá al unificar las recomendaciones del DOM con las del propio HTML, pero persistirá en las implementaciones anteriores de los navegadores.

© JMA 2020. All rights reserved

Árbol de nodos

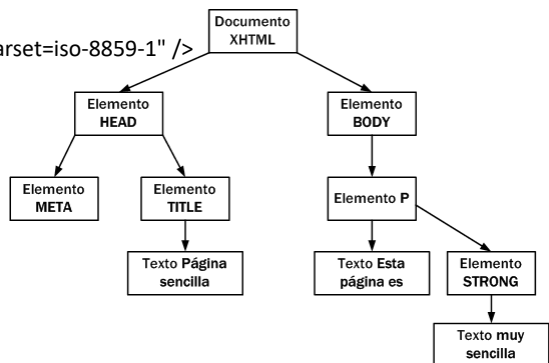
- Antes de poder utilizar sus funciones, DOM transforma internamente el archivo XML o HTML original en una estructura más fácil de manejar formada por una jerarquía de nodos.
- DOM transforma el código XML en una serie de nodos interconectados en forma de árbol.
- El árbol generado no sólo representa los contenidos del archivo original (mediante los nodos del árbol) sino que también representa sus relaciones (mediante las ramas del árbol que conectan los nodos).
- Aunque en ocasiones DOM se asocia con la programación web y con JavaScript, la API de DOM es independiente de cualquier lenguaje de programación.
- DOM está disponible en la mayoría de lenguajes de programación comúnmente empleados.

© JMA 2020. All rights reserved

Árbol de nodos

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Página sencilla</title>
</head>

<body>
<p>Esta página es <strong>muy sencilla</strong></p>
</body>
</html>
```



© JMA 2020. All rights reserved

Tipos de nodos

- Los documentos XML y HTML tratados por DOM se convierten en una jerarquía de nodos.
- Los nodos que representan los documentos pueden ser de diferentes tipos.
- Los tipos más importantes son:
 - Document: es el nodo raíz de todos los documentos HTML y XML. Todos los demás nodos derivan de él.
 - DocumentType: es el nodo que contiene la representación del DTD empleado en la página (indicado mediante el DOCTYPE).
 - Element: representa el contenido definido por un par de etiquetas de apertura y cierre (<etiqueta>...</etiqueta>) o de una etiqueta abreviada que se abre y se cierra a la vez (<etiqueta/>). Es el único nodo que puede tener tanto nodos hijos como atributos.
 - Attr: representa el par nombre-de-atributo/valor.
 - Text: almacena el contenido del texto que se encuentra entre una etiqueta de apertura y una de cierre. También almacena el contenido de una sección de tipo CDATA.
 - CDATASection: es el nodo que representa una sección de tipo <![CDATA[]]>.
 - Comment: representa un comentario de XML.
- Los otros tipos de nodos son: DocumentFragment, Entity, EntityReference, ProcessingInstruction y Notation.

© JMA 2020. All rights reserved

Interfaz Node: Constantes

- Node.ELEMENT_NODE = 1
- Node.ATTRIBUTE_NODE = 2
- Node.TEXT_NODE = 3
- Node.CDATA_SECTION_NODE = 4
- Node.ENTITY_REFERENCE_NODE = 5
- Node.ENTITY_NODE = 6
- Node.PROCESSING_INSTRUCTION_NODE = 7
- Node.COMMENT_NODE = 8
- Node.DOCUMENT_NODE = 9
- Node.DOCUMENT_TYPE_NODE = 10
- Node.DOCUMENT_FRAGMENT_NODE = 11
- Node.NOTATION_NODE = 12

© JMA 2020. All rights reserved

Interfaz Node: Propiedades

Propiedad	Valor devuelto	Descripción
nodeName	String	El nombre del nodo (no definido en todos tipos de nodo)
nodeValue	String	El valor del nodo (no definido en todos los tipos de nodo)
nodeType	Number	Una de las 12 constantes definidas anteriormente
ownerDocument	Document	Referencia del documento al que pertenece el nodo
firstChild	Node	Referencia del primer nodo de la lista childNodes
lastChild	Node	Referencia del último nodo de la lista childNodes
childNodes	NodeList	Lista de todos los nodos hijo del nodo actual
previousSibling	Node	Referencia del nodo hermano anterior o null si este nodo es el primer hermano
nextSibling	Node	Referencia del nodo hermano siguiente o null si este nodo es el último hermano
attributes	NamedNodeMap	Se emplea con nodos de tipo Element. Contiene objetos de tipo Attr que definen todos los atributos del elemento

© JMA 2020. All rights reserved

Interfaz Node: Métodos

Método	Valor devuelto	Descripción
hasChildNodes()	Boolean	Devuelve true si el nodo actual tiene uno o más nodos hijo
appendChild(nodo)	Node	Añade un nuevo nodo al final de la lista childNodes
insertBefore(nuevoNodo, anteriorNodo)	Node	Inserta el nodo nuevoNodo antes que la posición del nodo anteriorNodo dentro de la lista childNodes
removeChild(nodo)	Node	Elimina un nodo de la lista childNodes
replaceChild(nuevoNodo, anteriorNodo)	Node	Reemplaza el nodo anteriorNodo por el nodo nuevoNodo

© JMA 2020. All rights reserved

Acceso a los nodos

- DOM proporciona dos métodos alternativos para acceder a un nodo específico: acceso a través de sus nodos padre y acceso directo.
- Las propiedades del interfaz Node permiten acceder al nodo raíz de la página y navegar a través de la jerarquía de los nodos padre a sus nodos hijos hasta alcanzar el nodo buscado.
- DOM proporciona una serie de métodos para acceder de forma directa a los nodos deseados:
 - **getElementsByTagName(eti)**: obtiene todos los elementos de la página cuya etiqueta sea igual que el parámetro que se le pasa a la función.
 - **getElementsByTagName(name)**: obtiene todos los elementos de la página cuyo atributo name coincida con el parámetro que se le pasa a la función.
 - **getElementById(id)**: obtiene el elemento de la página cuyo atributo id coincida con el parámetro que se le pasa a la función. Se trata de la función preferida para acceder directamente a un nodo.

© JMA 2020. All rights reserved

Selectors API (HTML5)

- **querySelectorAll(css)**
 - Es el método más versátil para seleccionar etiquetas: devuelve todos los elementos dentro de un elemento que coinciden con el selector CSS dado.

```
for (let elem of document.querySelectorAll('ul > li:last-child')) {  
    alert(elem.innerHTML); // "prueba", "pasado"  
}
```
- **querySelector(css)**
 - La llamada a `elem.querySelector(css)` devuelve el primer elemento para el selector CSS dado, equivale a `elem.querySelectorAll(css)[0]`.

```
let element = document.querySelector('#elementId');
```
- Aparte de los métodos de las consultas de selección, están disponibles:
 - `elem.matches(css)` para comprobar si `elem` coincide con el selector CSS dado.
 - `elem.contains(subelem)` devuelve `true` si `subelem` está dentro de `elem` (un descendiente de `elem`) o cuando `elem==subelem`.
 - `elem.closest(css)` para buscar el ancestro más cercano que coincida con el selector CSS dado. El propio `elem` también se comprueba.

© JMA 2020. All rights reserved

Atributos

- Los nodos de tipo Element contienen la propiedad attributes, que permite acceder a todos los atributos de cada elemento.
- DOM proporciona diversos métodos para tratar con los atributos:
 - getItem(nombre), devuelve el nodo cuya propiedad nodeName contenga el valor nombre.
 - setItem(nodo), añade el nodo a la lista attributes, indexándolo según su propiedad nodeName.
 - removeItem(nombre), elimina el nodo cuya propiedad nodeName coincida con el valor nombre.
 - item(posicion), devuelve el nodo que se encuentra en la posición indicada por el valor numérico del parámetro.
- Estos métodos devuelven un nodo de tipo Attr, para acceder a su valor para consultarlo o modificarlo es necesario utilizar su propiedad nodeValue.
- DOM proporciona métodos de acceso directo:
 - getAttribute(nombre), es equivalente a attributes.getItem(nombre).
 - setAttribute(nombre, valor) equivalente a attributes.getItem(nombre).value = valor.
 - removeAttribute(nombre), equivalente a attributes.removeItem(nombre).

© JMA 2020. All rights reserved

Creación y eliminación de nodos

- Los métodos DOM disponibles para el mantenimiento del árbol de nodos son los siguientes:
 - createElement(eti): Crea un elemento del tipo indicado en el parámetro.
 - createAttribute(atrib): Crea un nodo de tipo atributo con el nombre indicado.
 - createTextNode(texto): Crea un nodo de tipo texto con el valor indicado como parámetro.
 - appendChild(nodo): Añade un nodo al final de la lista childNodes de otro nodo. Se debe invocar sobre el nodo que va a ser nodo padre del nodo añadido.
 - replaceChild(new, old): intercambia un nodo por otro. Se debe invocar sobre el nodo padre que contiene el nodo que se va a cambiar.
 - removeChild(nodo): elimina un nodo. Se debe invocar sobre el nodo padre del nodo que se va a eliminar.

© JMA 2020. All rights reserved

Pasos para crear elementos

1. Crear un nodo de tipo elemento
2. Cualificar el nuevo elemento con atributos. Por cada atributo:
 1. Crear un nodo de tipo atributo
 2. Asociar el nodo de atributo al elemento
3. Dar contenido al nuevo elemento.
 1. Crear un nodo de tipo texto
 2. Asociar el nodo de texto al elemento
4. Modificar al elemento padre que lo va a contener en la página original.
 - Añadir el nodo del nuevo elemento.
 - Buscar el nodo original y sustituirlo por el nodo del nuevo elemento.

© JMA 2020. All rights reserved

HTML DOM

- El Modelo de Objetos del Documento HTML amplía el interfaz Node exponiendo ciertos métodos y propiedades de conveniencia que son consistentes con los modelos existentes y que son más apropiados para los autores de scripts.
- Incluye las siguientes especializaciones para HTML:
 - Una Interfaz HTMLDocument, derivada de la interfaz Document del núcleo. HTMLDocument especifica las operaciones y consultas que pueden realizarse en un documento HTML.
 - Una Interfaz HTMLElement, derivada de la interfaz Element del núcleo. HTMLElement especifica las operaciones y consultas que pueden realizarse en cualquier elemento HTML. Entre los métodos de HTMLElement se incluyen aquellos que permiten leer y modificar los atributos que se aplican a todos los elementos HTML.
 - Especializaciones para todos los elementos HTML que tengan atributos que vayan más allá de los especificados en la Interfaz HTMLElement. La interfaz derivada para el elemento contiene métodos explícitos para establecer y obtener los valores para todos estos atributos. Así mismo expone los diferentes eventos soportados por cada elemento.

© JMA 2020. All rights reserved

Interfaz HTMLDocument

- Un HTMLDocument es la raíz de la jerarquía HTML y almacena todos los contenidos. Además de proporcionar acceso a la jerarquía, también proporciona algunos métodos de conveniencia para acceder a ciertos conjuntos de información del documento.
- Métodos:
 - open: Abre un flujo de documento para escribir.
 - close: Cierra un flujo de documento abierto por open() y fuerza la representación.
 - write: Escribe una cadena de texto en un flujo de documento abierto por open().
 - writeln: Escribe una cadena de texto seguida por un carácter de nueva línea en un flujo de documento abierto por open().
 - getElementById: Devuelve el elemento cuyo id está dado por elementId.
 - getElementsByName: Devuelve el conjunto (posiblemente vacío) de elementos cuyo valor name está dado por elementName.
 - querySelectorAll: Devuelve todos los elementos secundarios que coinciden con un selector CSS.

© JMA 2020. All rights reserved

HTMLDocument

- Atributos:
 - title: El título de un documento según se especifica en el elemento TITLE de la cabecera del documento.
 - referrer: Devuelve el URI de la página que referenció a esta página. Si el usuario navegó hasta esta página directamente el valor es una cadena vacía.
 - domain: El nombre de dominio del servidor que sirvió el documento, o una cadena vacía si el servidor no puede ser identificado por un nombre de dominio.
 - URL: El URI completo del documento.
 - body: El elemento que contiene el contenido del documento.
 - images: Un conjunto de todos los elementos IMG de un documento. Por motivos de compatibilidad, el comportamiento se limita a elementos IMG.
 - applets: Un conjunto de todos los elementos OBJECT que incluyan aplicaciones y elementos APPLET (desaprobados) de un documento.
 - links: Un conjunto de todos los elementos AREA y elementos ancla (A) de un documento con un valor para el atributo href.
 - forms: Un conjunto de todos los formularios de un documento.
 - anchors: Un conjunto de todos los elementos ancla (A) de un documento con un valor para el atributo name.
 - cookie: Las cookies asociadas a este documento.

© JMA 2020. All rights reserved

Interfaz HTMLElement

- Todas las interfaces de elementos HTML derivan de esta interfaz.
- Atributos:
 - id: El identificador del elemento.
 - title: El título consultivo del elemento.
 - lang: Código de idioma definido en RFC 1766.
 - dir: Especifica la dirección base de la direccionalidad del texto neutral y la de direccionalidad de tablas.
 - className: El atributo de class del elemento.
 - innerHTML: Contenido de un elemento.
 - textContent: Contenido textual de un nodo y sus descendientes
 - style: Estilo de un elemento

© JMA 2020. All rights reserved

Métodos HTMLElement

Método	Descripción
click()	Simula un clic del ratón sobre un elemento
focus(), blur()	Da y quita el foco a un elemento
addEventListener()	Conecta un controlador de eventos para el elemento especificado
removeEventListener()	Elimina un controlador de eventos que se han asociado con addEventListener()
getElementsByClassName(), getElementsByTagName()	Devuelve una colección de todos los elementos secundarios con el nombre de clase especificado o el nombre de la etiqueta especificada
querySelector()	Devuelve el primer elemento hijo que coincide con un selector CSS
querySelectorAll()	Devuelve todos los elementos secundarios que coinciden con un selector CSS
setAttribute()	Establece o cambia el atributo especificado con el valor especificado
removeAttribute()	Elimina un atributo especificado de un elemento
appendChild()	Añade un nuevo nodo hijo a un elemento como el último nodo hijo
replaceChild()	Reemplaza un nodo secundario en un elemento
removeChild()	Elimina un nodo hijo de un elemento
cloneNode()	Clones un elemento

© JMA 2020. All rights reserved

Interfaces HTMLElement

HTMLHtmlElement	HTMLInputElement	HTMLDivElement	HTMLParamElement
HTMLHeadElement	HTMLTextAreaElement	HTMLParagraphElement	HTMLAppletElement
HTMLLinkElement	HTMLButtonElement	HTMLHeadingElement	HTMLMapElement
HTMLTitleElement	HTMLLabelElement	HTMLQuoteElement	HTMLAreaElement
HTMLMetaElement	HTMLFieldSetElement	HTMLPreElement	HTMLScriptElement
HTMLBaseElement	HTMLLegendElement	HTMLBRElement	HTMLTableElement
HTMLIsIndexElement	HTMLUListElement	HTMLBaseFontElement	HTMLTableCaptionElement
HTMLStyleElement	HTMLOListElement	HTMLFontElement	HTMLTableColElement
HTMLBodyElement	HTMLDListElement	HTMLHRElement	HTMLTableSectionElement
HTMLFormElement	HTMLDirectoryElement	HTMLModElement	HTMLTableRowElement
HTMLSelectElement	HTMLMenuElement	HTMLAnchorElement	HTMLTableCellElement
HTMLOptGroupElement	HTMLLIElement	HTMLImageElement	HTMLFrameSetElement
HTMLOptionElement	HTMLBlockquoteElement	HTMLObjectElement	HTMLFrameElement
			HTMLIFrameElement

© JMA 2020. All rights reserved

Programación basada en eventos

- Frente a la programación tradicional, donde las aplicaciones se ejecutaban secuencialmente de principio a fin, en la actualidad el modelo predominante es el de la programación asíncrona basada en eventos.
- Los scripts y programas inician el entorno o página y quedan a la espera, sin realizar ninguna tarea, hasta que se produzca un evento, situación ante la cual les interesa reaccionar.
- Una vez producido, para realizar el tratamiento del evento, ejecutan alguna tarea asociada a la aparición de ese evento. Al concluir el tratamiento, el script o programa vuelve al estado de espera de siguiente evento.
- El tratamiento del evento suele encapsularse en una función, conocidas como “controladores de eventos” o “manejadores de eventos”.
- JavaScript permite asignar una función a cada uno de los eventos. De esta forma, cuando se produce cualquier evento, JavaScript ejecuta su función asociada.
- Si la función devuelve el valor false se cancela la ejecución del evento.

© JMA 2020. All rights reserved

Eventos en DOM

- El nivel 1 de DOM no incluye especificaciones relativas a los eventos JavaScript. El nivel 2 de DOM incluye ciertos aspectos relacionados con los eventos y el nivel 3 de DOM incluye la especificación completa de los eventos de JavaScript.
- La incompatibilidad más importante entre navegadores y versiones de los mismos se da en el modelo de eventos del navegador.
- Así, existen hasta tres modelos diferentes para manejar los eventos dependiendo del navegador en el que se ejecute la aplicación:
 - **Modelo básico de eventos:** Este modelo simple de eventos se introdujo para la versión 4 del estándar HTML y se considera parte del nivel más básico de DOM. Aunque sus características son limitadas, es el único modelo que es compatible en todos los navegadores y por tanto, el único que permite crear aplicaciones que funcionan de la misma manera en todos los navegadores.
 - **Modelo de eventos estándar:** Las versiones más avanzadas del estándar DOM (DOM nivel 2) definen un modelo de eventos completamente nuevo y mucho más poderoso que el original. Todos los navegadores modernos lo incluyen, salvo Internet Explorer.
 - **Modelo de eventos de Internet Explorer:** Internet Explorer utiliza su propio modelo de eventos, que es similar pero incompatible con el modelo estándar. Creado para Internet Explorer 4, Microsoft decidió seguir utilizándolo en el resto de versiones, a pesar de haber participado en la creación del estándar de DOM que define el modelo de eventos estándar.

© JMA 2020. All rights reserved

Modelo básico de eventos

- Cada elemento o etiqueta define su propia lista de posibles eventos que se le pueden asignar.
- Un mismo tipo de evento puede estar definido para varios elementos diferentes y un elemento puede tener asociados varios eventos diferentes.
- El nombre de cada evento se construye mediante el prefijo on, seguido del nombre en inglés de la acción asociada al evento.
- Algunos de los eventos mas comunes son:
 - onclick: Pinchar y soltar el ratón
 - onchange: Deseleccionar un elemento que se ha modificado
 - onfocus: Seleccionar un elemento
 - onblur: Deseleccionar el elemento
 - onselect: Seleccionar un texto
 - onsubmit: Enviar el formulario
 - onload: La página se ha cargado completamente
 - onunload: Se abandona la página

© JMA 2020. All rights reserved

Manejadores de eventos

- El HTML expone, en su recomendación, la lista de posibles eventos de una etiqueta como si fueran atributos de la etiqueta.
- Existen varias formas para enlazar el código de control del evento al evento:
 - Manejadores como atributos de los elementos.
 - En la etiqueta, el valor del atributo evento es el código JavaScript del controlador.
 - Manejadores como funciones externas.
 - En la etiqueta, el valor del atributo evento es la invocación de una función reutilizable de JavaScript que es el controlador.
 - Manejadores “semánticos” o “no intrusivos”.
 - Para separar los contenidos del comportamiento, una vez definido el contenido en HTML, mediante un script se localizan las etiquetas y se les asocia a sus eventos los controladores de eventos.

© JMA 2020. All rights reserved

Modelo de eventos estándar

- La especificación DOM define otros dos métodos:
 - `addEventListener()`: asocia manejadores de eventos a eventos.
 - `removeEventListener()`: desasociar manejadores de eventos previamente asociados al evento.
- Ambos métodos requieren tres mismos parámetros:
 - el nombre del “event listener”: el nombre del evento sin el prefijo `on`.
 - una referencia a la función encargada de procesar el evento
 - el tipo de flujo de eventos al que se aplica: `true` cuando el manejador se emplea en la fase de capture y `false` cuando el manejador se asocia a la fase de bubbling.
- El modelo DOM permiten asociar mas de un manejador de evento al mismo evento de un objeto, por lo que para desasociar el evento los parámetros deben ser exactamente los mismos que en la asociación.
- También se pueden asociar asignándoselos a los atributos con prefijo `on`:
 - `button.setAttribute('onclick', "alert('NOT CALLED')");`
 - `button.onclick = function () { alert('TWO'); };`

© JMA 2020. All rights reserved

Principales Eventos

Clipboard	onCopy onCut onPaste
Composition	onCompositionEnd onCompositionStart onCompositionUpdate
Keyboard	onKeyDown onKeyPress onKeyUp
Focus	onFocus onBlur
Mouse	onClick onContextMenu onDoubleClick onDrag onDragEnd onDragEnter onDragExit onDragLeave onDragOver onDragStart onDrop onMouseDown onMouseEnter onMouseLeave onMouseMove onMouseOut onMouseOver onMouseUp onWheel
Selection	onSelect
Touch	onTouchCancel onTouchEnd onTouchMove onTouchStart
UI	onScroll
Media	onAbort onCanPlay onCanPlayThrough onDurationChange onEmptied onEncrypted onEnded onError onLoadedData onLoadedMetadata onLoadStart onPause onPlay onPlaying onProgress onRateChange onSeeked onSeeking onStalled onSuspend onTimeUpdate onVolumeChange onWaiting
Image	onLoad onError
Animation	onAnimationStart onAnimationEnd onAnimationIteration
Transition	onTransitionEnd
Other	onToggle

© JMA 2020. All rights reserved

El flujo de eventos

- El flujo de eventos permite que varios elementos diferentes puedan responder a un mismo evento, propagación del evento siguiendo las relaciones de contenedores y contenidos.
- El orden en el que se ejecutan los eventos asignados a cada elemento de la página es lo que constituye el flujo de eventos, pero existen muchas diferencias en el flujo de eventos de cada navegador:
 - Event bubbling: En este modelo, el orden que se sigue es ascendente desde el elemento más específico (contenido) hasta el elemento menos específico (continente).
 - Event capturing: el flujo de eventos se define descendentemente desde el elemento menos específico hasta el elemento más específico.
 - Eventos DOM: El flujo de eventos definido en la especificación DOM soporta tanto el bubbling como el capturing, pero el "event capturing" se ejecuta en primer lugar. Los dos flujos de eventos recorren todos los objetos DOM desde el objeto document hasta el elemento más específico y viceversa. Además, la mayoría de navegadores que implementan los estándares, continúan el flujo hasta el objeto window.

© JMA 2020. All rights reserved

El objeto event

- Cuando se produce un evento, no es suficiente con asignarle una función responsable de procesar ese evento.
- El objeto event es el mecanismo definido por los navegadores para proporcionar toda esa información.
- Se trata de un objeto que se crea automáticamente cuando se produce un evento y que se destruye de forma automática cuando se han ejecutado todas las funciones asignadas al evento.
- El estándar DOM especifica que el objeto event es el único parámetro que se debe pasar a las funciones encargadas de procesar los eventos.
 - `var ev= arguments[0];`
- Internet Explorer permite el acceso al objeto event a través del objeto window.
 - `var ev= window.event;`
- El objeto event presenta unas propiedades y métodos muy diferentes en función del tipo de navegador en el que se ejecuta la aplicación JavaScript.

© JMA 2020. All rights reserved

event (DOM)

Propiedad/Método	Devuelve	Descripción
altKey	Boolean	Devuelve true si se ha pulsado la tecla ALT y false en otro caso
bubbles	Boolean	Indica si el evento pertenece al flujo de eventos de bubbling
button	Entero	El botón del ratón que ha sido pulsado.
cancelable	Boolean	Indica si el evento se puede cancelar
cancelBubble	Boolean	Indica si se ha detenido el flujo de eventos de tipo bubbling
charCode	Entero	El código unicode del carácter correspondiente a la tecla pulsada
clientX	Entero	Coordenada X de la posición del ratón respecto del área visible de la ventana
clientY	Entero	Coordenada Y de la posición del ratón respecto del área visible de la ventana
ctrlKey	Boolean	Devuelve true si se ha pulsado la tecla CTRL y false en otro caso
currentTarget	Element	El elemento que es el objetivo del evento
detail	Entero	El número de veces que se han pulsado los botones del ratón
eventPhase	Entero	La fase a la que pertenece el evento: 0 – Fase capturing 1 – En el elemento destino 2 – Fase bubbling

© JMA 2020. All rights reserved

event (DOM)

Propiedad/ Método	Devuelve	Descripción
isChar	Boolean	Indica si la tecla pulsada corresponde a un carácter
keyCode	Entero	Indica el código numérico de la tecla pulsada
metaKey	Entero	Devuelve true si se ha pulsado la tecla META y false en otro caso
pageX	Entero	Coordenada X de la posición del ratón respecto de la página
pageY	Entero	Coordenada Y de la posición del ratón respecto de la página
preventDefault()	Función	Se emplea para cancelar la acción predefinida del evento
relatedTarget	Element	El elemento que es el objetivo secundario del evento (relacionado con los eventos de ratón)
screenX	Entero	Coordenada X de la posición del ratón respecto de la pantalla completa
screenY	Entero	Coordenada Y de la posición del ratón respecto de la pantalla completa
shiftKey	Boolean	Devuelve true si se ha pulsado la tecla SHIFT y false en otro caso
stopPropagation()	Función	Se emplea para detener el flujo de eventos de tipo bubbling
target	Element	El elemento que origina el evento
timeStamp	Número	La fecha y hora en la que se ha producido el evento
type	Cadena	El nombre del evento

© JMA 2020. All rights reserved

JavaScript no intrusivo

- Todas las aplicaciones que se desarrollen deberían cumplir la normativa de accesibilidad vigente.
- Una de las normas indica que no se debe introducir código JavaScript dentro de los elementos HTML.
- Los validadores lo detectan y avisan que no se podrá interactuar con determinados elementos que el usuario que accede desconoce, que no puede visualizar correctamente, etc.
- Alternativa:
 - Buscar y asignar dinámicamente los controladores de eventos al final del cuerpo (body):

```
(function(){  
    document.querySelector('#btnVer').addEventListener('click', function () { ... });  
})();
```
 - Usar atributos personalizados (no estándar) dentro de la etiqueta HTML que serán sustituidos al cargar el documento por el código JavaScript correspondiente (Ej: jquery.unobtrusive):
 - Validaciones: data-val-number, data-val-required, data-valmsg-for, data-valmsg-replace, ...

© JMA 2020. All rights reserved

data-* (HTML5)

- Un atributo de datos personalizado es un atributo que no está en ningún espacio de nombres, cuyo nombre comienza con la cadena "data-", tiene al menos un carácter después del guion, es compatible con XML y **no contiene letras ASCII en mayúsculas**.

```
<div class="spaceship" data-ship-id="92432"  
    data-weapons="laser 2" data-shields="50%"  
    data-x="30" data-y="10" data-z="90">
```

- Los atributos de datos personalizados están destinados a almacenar datos personalizados privados en la página o aplicación, para la cual no hay atributos o elementos más apropiados.
- La propiedad dataset de las etiquetas expone los datos personalizados como propiedades, cuyo nombre es el nombre del atributo sin el prefijo "data-" y, en caso de contener -, se elimina y el siguiente carácter se pasa a mayúscula:

```
var x = miDiv.dataset.shipId;
```

© JMA 2020. All rights reserved

Formularios

- Los formularios permiten las entradas de usuario, una de las principales razones por las que se creó el lenguaje de programación JavaScript fue la necesidad de validar los datos de los formularios directamente en el navegador del usuario.
- document.forms es el array que contiene todos los formularios de la página. Se crea automáticamente un array llamado elements por cada uno de los formularios de la página que contiene la referencia a todos los elementos (cuadros de texto, botones, listas desplegables, etc.) del formulario.
- Cada elemento cuenta con las siguientes propiedades:
 - type: indica el tipo de elemento que se trata. Para los elementos de tipo <input> (text, button, checkbox, etc.) coincide con el valor de su atributo type. Para las listas desplegables normales (elemento <select>) su valor es select-one, lo que permite diferenciarlas de las listas que permiten seleccionar varios elementos a la vez y cuyo tipo es select-multiple. Por último, en los elementos de tipo <textarea>, el valor de type es textarea.
 - form: es una referencia directa al formulario al que pertenece el elemento.
 - name: obtiene el valor del atributo name de XHTML.
 - value: permite leer y modificar el valor del atributo value. Para los campos de texto (<input type="text"> y <textarea>) obtiene el texto que ha escrito el usuario. Para los botones obtiene el texto que se muestra en el botón.

© JMA 2020. All rights reserved

Formularios

- Los eventos más utilizados en el manejo de los formularios son los siguientes:
 - onclick: evento que se produce cuando se pincha con el ratón sobre un elemento. Normalmente se utiliza con cualquiera de los tipos de botones (`<input type="button">`, `<input type="submit">`, `<input type="image">`).
 - onchange: evento que se produce cuando el usuario cambia el valor de un elemento de texto (`<input type="text">` o `<textarea>`). También se produce cuando el usuario selecciona una opción en una lista desplegable (`<select>`). Sin embargo, el evento sólo se produce si después de realizar el cambio, el usuario pasa al siguiente campo del formulario, lo que técnicamente se conoce como que "el otro campo de formulario ha perdido el foco".
 - onfocus: evento que se produce cuando el usuario selecciona un elemento del formulario.
 - onblur: evento complementario de onfocus, ya que se produce cuando el usuario ha deseleccionado un elemento por haber seleccionado otro elemento del formulario. Técnicamente, se dice que el elemento anterior "ha perdido el foco".

© JMA 2020. All rights reserved

Validación

- La validación de formularios en cliente evita la sobrecarga que supone las innecesarias idas y venidas al servidor, pero no sustituye la validación de servidor, con lo que mejora la experiencia de usuario y ayuda a reducir el tráfico de red y la carga de procesamiento en el servidor.
- Normalmente, la validación de un formulario consiste en llamar a una función de validación cuando el usuario pulsa sobre el botón de envío del formulario.
- El formulario dispone del evento onsubmit que se dispara justo antes de enviar el formulario. La función de validación se puede asociar a dicho evento.
- Si el evento onsubmit devuelve el valor true, el formulario se envía normalmente pero, si es false, se cancela el envío del formulario.
- El formulario se envía cuando el usuario pulsa sobre el botón `<input type="submit">` o cuando se invoca el método `form.submit()`.

© JMA 2020. All rights reserved

Validación restringida (HTML5)

- El HTML5 brinda sintaxis y elementos de API para posibilitar la validación de formularios del lado del cliente.
- Aunque esta funcionalidad no reemplaza la validación del lado del servidor, que todavía es necesaria por seguridad e integridad de la información, la validación del lado del cliente puede brindar una experiencia de usuario mejor al darle al usuario una respuesta inmediata acerca de la información ingresada.
- Se puede evitar la validación restringida especificando el atributo `novalidate` en el elemento `<form>`, o el atributo `formnovalidate` en el elemento `<button>` y en el elemento `<input>` (cuando `type` es `submit` o `image`). Estos atributos indican que el formulario no será validado cuando se envíe.

© JMA 2020. All rights reserved

Validaciones soportadas (HTML5)

- Se han incorporado una serie de atributos a la etiqueta `INPUT` que permiten validar automáticamente el formulario antes de enviarlo.
 - `Type`: `number`, `range`, `url`, `email`, `date`, `month`, `week`, `tel`, `time`, `color`
 - `Required`: es obligatorio dar valor.
 - `Multiple`: indica si al usuario se le permite especificar más de un valor (`email`, `url`, `file`, ...).
 - `Pattern`: especifica una expresión regular que debe cumplir el valor del control (o los valores si `múltiple` está activado)
 - `Min` y `Max`: indican el rango permitido de valores para el elemento.
 - `Step`: indica la granularidad que se espera (y requiere) del valor, mediante la limitación de los valores permitidos.

© JMA 2020. All rights reserved

API de validación restringida (HTML5)

- En objetos `HTMLFormElement` el método `checkValidity()`, que devuelve verdadero si todos los elementos asociados del formulario que necesitan validación satisfacen las restricciones.
- En elementos asociados al formulario:
 - la propiedad `willValidate`, es falso si el elemento no satisface las restricciones.
 - la propiedad `validity`, es un objeto `ValidityState` que representa los estados de validación en que está el elemento (p. ej., condiciones de restricción que han fallado o exitosas).
 - la propiedad `validationMessage`, es un mensaje que contiene todas las fallas o errores en las restricciones que pertenecen a ese elemento.
 - el método `checkValidity()`, devuelve falso si el elemento no logra satisfacer alguna de las restricciones, o verdadero si pasa lo contrario.
 - el método `setCustomValidity()`, establece un mensaje de validación personalizado, permitiendo imponer y validar restricciones más allá de las que están predefinidas.

© JMA 2020. All rights reserved

Propiedad `validity` (HTML5)

```
interface ValidityState {  
  readonly attribute boolean valueMissing;  
  readonly attribute boolean typeMismatch;  
  readonly attribute boolean patternMismatch;  
  readonly attribute boolean tooLong;  
  readonly attribute boolean tooShort;  
  readonly attribute boolean rangeUnderflow;  
  readonly attribute boolean rangeOverflow;  
  readonly attribute boolean stepMismatch;  
  readonly attribute boolean badInput;  
  readonly attribute boolean customError;  
  readonly attribute boolean valid;  
};
```

© JMA 2020. All rights reserved

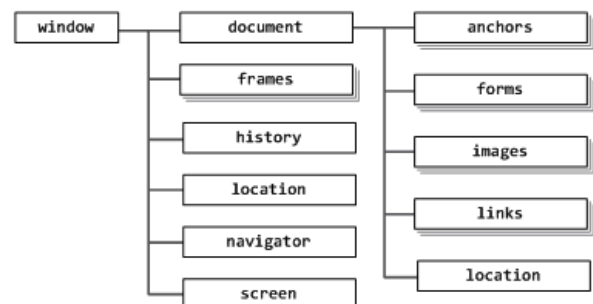
Validaciones

```
document.querySelector("#campo").addEventListener('change', function(ev) {
  if (ev.target.value.startsWith('H')) {
    ev.target.setCustomValidity('No puede empezar por H');
  } else {
    ev.target.setCustomValidity("");
  }
});
document.querySelector("#btnManual").addEventListener('click', function(ev) {
  if (document.forms[0].checkValidity()) {
    document.querySelector("#sumario").textContent = "";
    document.forms[0].submit();
  } else {
    let msg = "";
    for(let item of document.forms[0].elements) {
      if(!item.checkValidity()) msg += `<li>${item.dataset.label}: ${item.validationMessage}</li>`;
    }
    document.querySelector("#sumario").innerHTML = `<h3>Sumario</h3><ul>${msg}</ul>`;
  }
});
document.querySelector("form").addEventListener('submit', function (e) {
  // ...
  if (!document.forms[0].checkValidity()) { e.preventDefault(); }
});
```

© JMA 2020. All rights reserved

BOM (Browser Object Model)

- Las versiones 3.0 de los navegadores Internet Explorer y Netscape Navigator introdujeron el concepto de Browser Object Model o BOM, que permite acceder y modificar las propiedades de las ventanas del propio navegador.
- El mayor inconveniente de BOM es que, al contrario de lo que sucede con DOM, ninguna entidad se encarga de estandarizarlo o definir unos mínimos de interoperabilidad entre navegadores.



© JMA 2020. All rights reserved

window

- El objeto window representa la ventana completa del navegador. Mediante este objeto, es posible mover, redimensionar y manipular la ventana actual del navegador. Incluso es posible abrir y cerrar nuevas ventanas de navegador.
- La propiedades mas habituales de la ventana son:
 - defaultStatus: Establece o devuelve el texto por defecto en la barra de estado de una ventana
 - name: Establece o devuelve el nombre de una ventana
 - status: Establece o devuelve el texto en la barra de estado de una ventana
 - opener: Devuelve una referencia a la ventana que creó la ventana
 - parent: Devuelve la ventana principal de la ventana actual
 - closed: Devuelve un valor booleano que indica si una ventana se ha cerrado o no
 - innerHeight: Devuelve la altura interior del área de contenido de una ventana
 - innerWidth: Devuelve la anchura interior del área de contenido de una ventana
 - document: Devuelve el objeto de documento de la ventana (Ver objeto Document)
 - history: Devuelve el objeto Historia de la ventana (Ver objeto History)
 - location: Devuelve el objeto de localización de la ventana (Ver objeto Location)
 - navigator: Devuelve el objeto Navigator para la ventana (Ver objeto Navigator)

© JMA 2020. All rights reserved

window

Método	Descripción
alert()	Muestra un cuadro de alerta con un mensaje y un botón Aceptar
prompt()	Muestra un cuadro de diálogo que solicita una entrada de texto
confirm()	Muestra un cuadro de diálogo con un mensaje con Aceptar/Cancelar
stop()	Detiene la ventana de carga
open()	Abre una nueva ventana del navegador
close()	Cierra la ventana actual
createPopup()	Crea una ventana emergente
print()	Imprime el contenido de la ventana actual
focus()	Pone el foco en la ventana actual
blur()	Quita el foco de la ventana actual
atob()	Decodifica una cadena codificada en base 64
btoa()	Codifica una cadena en base 64

© JMA 2020. All rights reserved

window

Método	Descripción
setTimeout()	Llama a una función o evalúa una expresión después de un número especificado de milisegundos
clearTimeout()	Borra un temporizador programado con setTimeout ()
setInterval()	Llama a una función o una expresión evalúa a intervalos especificados (en milisegundos)
clearInterval()	Borra un temporizador programado con setInterval ()
moveTo()	Mueve una ventana a la posición especificada
moveBy()	Mueve una ventana con relación a su posición actual
resizeBy()	Cambia el tamaño de la ventana por los píxeles especificados
resizeTo()	Cambia el tamaño de la ventana a la anchura y altura especificadas
scrollBy()	Desplaza el documento por el número de píxeles especificado
scrollTo()	Desplaza el documento a las coordenadas especificadas

© JMA 2020. All rights reserved

location

- El objeto location es uno de los objetos más útiles del BOM. Debido a la falta de estandarización, location es una propiedad tanto del objeto window como del objeto document.
- El objeto location representa la URL de la página HTML que se muestra en la ventana del navegador y proporciona varias propiedades y métodos útiles para el manejo de la URL:
 - hash: El contenido de la URL que se encuentra después del signo # (ancla)
 - href: La URL completa de la página actual
 - search: Todo el contenido que se encuentra tras el símbolo ?, es decir, la consulta o "query string"
 - assign(newURL): Equivalente a location.href = newURL
 - replace(newURL): Similar a assign(), salvo que se borra la página actual del array history del navegador
 - reload(srv): Recarga la página. Si el argumento es true, se carga la página desde el servidor, o cuando es false desde la cache del navegador.

© JMA 2020. All rights reserved

history

- El objeto `history` proporciona acceso al historial del navegador. Esto expone métodos útiles y propiedades permiten avanzar y retroceder a través del historial del usuario:
 - `length`: Para obtener el número de páginas en el historial de la pila.
 - `back()`: Para moverse hacia atrás, página anterior, actuará exactamente como si el usuario hiciera clic en el botón atrás en la barra de herramientas del navegador.
 - `forward()`: Para moverse hacia adelante, página siguiente, actuará exactamente como si el usuario hiciera clic en el botón siguiente en la barra de herramientas del navegador.
 - `go(np)`: para cargar una página desde el historial de la sesión, identificada por su posición relativa a la página actual (Iniciando con la página actual, relativa al índice 0, los valores negativos representan páginas anteriores y los positivos páginas posteriores).

© JMA 2020. All rights reserved

navigator

- El objeto `navigator` es uno de los primeros objetos que incluyó el BOM y permite obtener información sobre el propio navegador.
- Aunque es uno de los objetos menos estandarizados, algunas de sus propiedades son comunes en casi todos los navegadores:
 - `appName`: Cadena que representa el nombre oficial del navegador
 - `appVersion`: Cadena que representa la versión del navegador
 - `browserLanguage`: Cadena que representa el idioma del navegador
 - `cookieEnabled`: Boolean que indica si las cookies están habilitadas
 - `javaEnabled`: Boolean que indica si Java está habilitado
 - `language`: Cadena que representa el idioma del navegador
 - `platform`: Cadena que representa la plataforma sobre la que se ejecuta el navegador
 - `plugins`: Array con la lista de plugins instalados en el navegador
 - `userAgent`: Cadena que representa la cadena que el navegador emplea para identificarse en los servidores

© JMA 2020. All rights reserved

screen

- El objeto screen se utiliza para obtener información sobre la pantalla del usuario.
- Uno de los datos más importantes que proporciona el objeto screen es la resolución del monitor en el que se están visualizando las páginas.
- Los diseñadores de páginas web necesitan conocer las resoluciones más utilizadas por los usuarios para adaptar sus diseños a esas resoluciones.
- Las siguientes propiedades están disponibles en el objeto screen:
 - availHeight: Altura de pantalla disponible para las ventanas
 - availWidth: Anchura de pantalla disponible para las ventanas
 - colorDepth: Profundidad de color de la pantalla (32 bits normalmente)
 - height: Altura total de la pantalla en píxel
 - width: Anchura total de la pantalla en píxel

© JMA 2020. All rights reserved

Nuevos APIs (HTML5)

- Geolocation
- LocalStorage
- Native Drag and Drop events
- Multitasking (Worker processes)
- Application Cache (offline access)
- Sockets (real-time server communication: chat, games, etc.)
- Server-Sent Events

© JMA 2020. All rights reserved

Geolocalización

- Nos permite averiguar la posición geográfica del usuario (lat, lon)
 - Hay métodos más precisos (GPS) y menos (a partir de la dirección IP o usando la red GSM)
 - El método exacto por el que se está calculando la posición es transparente al desarrollador Javascript
 - Lo único que nos da el API son las coordenadas. Necesitaremos algún servicio adicional dsi queremos dibujar un mapa con la posición, etc. (p.ej. Google Maps)
- Este API no funciona en Explorer 8 y anteriores. Se pueden usar librerías alternativas, como Google Gears (funciona, pero el API es distinto)

© JMA 2020. All rights reserved

Geolocalización

```
<script>
var x = document.getElementById("demo");

function getLocation() {
  if (navigator.geolocation) {
    navigator.geolocation.getCurrentPosition(showPosition);
  } else {
    x.innerHTML = "Geolocation is not supported by this browser.";
  }
}

function showPosition(position) {
  x.innerHTML = "Latitude: " + position.coords.latitude +
    "<br>Longitude: " + position.coords.longitude;
}
</script>
```

© JMA 2020. All rights reserved

Web Storage

- El almacenamiento local es una muy buena forma de guardar datos en el cliente sin tener que utilizar cookies.
- A diferencia de las cookies, el límite de almacenamiento es mucho mayor (al menos 5 MB) y la información nunca se transfiere al servidor.
- En localStorage, los datos que se guardan son de tipo/valor (key/value), así que si se desea guardar datos más complejos, podemos hacerlo guardando JSON en forma de string (JSON.stringify(obj)) que con JSON.parse(str) recuperaremos la estructura guardada.
- El almacenamiento local esta vinculado al origen (por dominio y protocolo). Todas las páginas, de un mismo origen, se pueden almacenar y acceder a los mismos datos.
- HTML define dos objetos (arrays asociativos de JavaScript) para almacenar datos en el cliente:
 - window.localStorage: almacenes de datos sin fecha de caducidad
 - window.sessionStorage: almacena los datos para una sola sesión (los datos se pierden cuando la pestaña del navegador se cierra)

© JMA 2020. All rights reserved

Web Storage

- Para comprobar el soporte del navegador:

```
if (typeof(Storage) !== "undefined") {  
    // Con soporte para almacenamiento.  
} else {  
    // Con soporte para almacenamiento.  
}
```
- Con localStorage.setItem("clave", "valor") si no existe previamente la clave: se crea la clave y se le asigna el valor, o si existe: se actualiza el valor.
- Con localStorage.getItem("clave") se recupera el valor asociado a la clave, si existe, "undefined" o si no existe.
- Con localStorage.removeItem("clave") se elimina del almacenamiento de forma permanente.

© JMA 2020. All rights reserved

Drag and Drop

- Arrastrar y soltar es una característica muy común. Que es cuando se "agarra" un objeto y se arrastra a una ubicación diferente.
- En HTML5, arrastrar y soltar es parte de la norma: Cualquier elemento puede ser arrastrable.
- Con el atributo `draggable="true"` se indica las etiquetas que permitimos arrastrar.
- Se utilizan una serie de eventos que se ejecutan durante las diversas etapas de la operación de arrastre y colocación.

© JMA 2020. All rights reserved

Drag and Drop

- `ondragstart`: especifica lo que debe suceder cuando se arrastra el elemento, cachea en el argumento del evento la información a arrastrar:
 - `ev.dataTransfer.setData("text", ev.target.id);`
- `ondragover`: cada elemento debe especificar a través del evento si permite recibir el arrastre (soltar) y en casos. Para autorizar el soltar:
 - `ev.preventDefault();`
- `ondrop`: define las operaciones a realizar cuando se suelta dentro del elemento, recupera la información arrastrada del argumento del evento, que debe ser del mismo tipo que la cacheada:
 - `var data = ev.dataTransfer.getData("text");`

© JMA 2020. All rights reserved

Web Worker

- Los navegadores ejecutan las aplicaciones en un único thread, lo que significa que si JavaScript está ejecutando una tarea muy complicada, que se traduce en tiempo de procesado, el rendimiento del navegador se ve afectado.
- Los Web workers se introdujeron con la idea de simplificar la ejecución de threads en el navegador.
- Un worker permite crear un entorno en el que un bloque de código JavaScript puede ejecutarse de manera paralela sin afectar al thread principal del navegador.
- Los Web workers utilizan un protocolo de paso de mensajes similar a los utilizados en programación paralela.
- El usuario puede seguir haciendo lo que quiere: hacer clic, la selección de las cosas, etc., mientras que el trabajador web se ejecuta en segundo plano.
- Dado que los trabajadores web están en archivos externos, no tienen acceso a los siguientes objetos JavaScript (hilo principal del JavaScript): DOM, window, document y parent.
- Esta limitación se puede solventar mediante el paso de mensajes con el metodo `postMessage` y con el evento `onmessage` del objeto trabajador web, los datos del trabajador web se traspasan en el `event.data`.

© JMA 2020. All rights reserved

Crear un Web Worker

- Se crea un fichero .js con la implementación del Worker, pero a diferencia de la ejecución un script en el documento principal, la visibilidad de un Worker es mucho más reducida.
- La palabra reservada `this` no hace referencia al objeto Global o Window, sino a la instancia del Worker que se está ejecutando.
- Para enviar mensajes al hilo principal:
`this.postMessage(datos);`
- Para recibir mensajes del hilo principal:
`this.addEventListener('message', function(e) {
 var datos = e.data;
 // ...
}, false);`
- El Worker termina cuando acaba el código JavaScript o cuando se invoca a `this.close()`.

© JMA 2020. All rights reserved

Usar un Web Worker

- Comprobar la disponibilidad de la característica:
`if(typeof(Worker) !== "undefined") {`
- Crear el Worker y asociar las funciones que procesan los mensajes recibidos y los errores.
`var worker = new Worker('workerWithError.js');
worker.addEventListener('message', function(e) {
 var datos = e.data;
 // ...
}, false);
worker.addEventListener('error', function(e) {...}, false);`
- Para enviar mensajes al Worker:
`worker.postMessage(datos);`

© JMA 2020. All rights reserved

Application Cache

- Cada vez es más importante poder acceder a las aplicaciones web sin conexión.
- HTML5 permite resolver algunas de las molestias asociadas al trabajo sin conexión mediante la interfaz `ApplicationCache`.
- Las tres ventajas que conlleva el uso de la interfaz de caché para una aplicación.
 - Navegación sin conexión: los usuarios pueden explorar todo el sitio web sin conexión.
 - Velocidad: los recursos almacenados en caché son locales y, por tanto, se cargan más rápido.
 - Reducción de carga del servidor: el navegador solo descarga recursos del servidor que han cambiado.
- La caché de aplicación (o `AppCache`) permite que el desarrollador especifique los archivos que el navegador debe almacenar en caché y poner a disposición de los usuarios que trabajen sin conexión.
- La aplicación se cargará y funcionará correctamente, incluso si el usuario pulsa el botón de actualización mientras trabaja sin conexión.

© JMA 2020. All rights reserved

Manifiesto de caché

- El archivo de manifiesto de caché es un sencillo archivo de texto que contiene los recursos que debe almacenar en caché el navegador para el acceso sin conexión.
<html manifest="http://www.example.com/example.mf">
- El atributo manifest debe estar incluido en todas las páginas de la aplicación web que se quiera almacenar en caché, el navegador no almacenará en caché ninguna página que no contenga el atributo manifest (a menos que esa página aparezca explícitamente en el propio archivo de manifiesto).
- Un archivo de manifiesto puede incluir tres secciones:
 - **CACHE:** Los archivos incluidos en esta sección (o inmediatamente después de CACHE MANIFEST) se almacenarán en caché explícitamente después de descargarse por primera vez (sección predeterminada).
 - **NETWORK:** Los archivos incluidos en esta sección no se cachean, siempre se solicitan al servidor incluso si el usuario está trabajando sin conexión.
 - **FALLBACK:** se especifican páginas alternativas en caso de no poder acceder a un recurso. La primera URI corresponde al recurso y la segunda, a la página alternativa.

© JMA 2020. All rights reserved

Manifiesto de caché

CACHE MANIFEST

/favicon.ico

CACHE:

index.html

stylesheet.css

images/logo.png

scripts/main.js

NETWORK:

login.php

FALLBACK:

*.html /offline.html

© JMA 2020. All rights reserved

API Application Cache

- El objeto `window.applicationCache` permite acceder (mediante programación) a la caché de aplicación del navegador.
- Su propiedad `status` permite comprobar el estado de la memoria caché.
- Para actualizar la caché mediante programación, primero se debe hacer una llamada a `applicationCache.update()`, cuando el estado de `applicationCache.status` sea `UPDATEREADY`, al llamar a `applicationCache.swapCache()`, se sustituirá la antigua caché por la nueva.
- El navegador activa eventos para una serie de acciones como el progreso de las descargas, la actualización de la caché de las aplicaciones y los estados de error.
- Los navegadores suelen limitar a 5 MB de datos almacenados en caché por cada sitio.

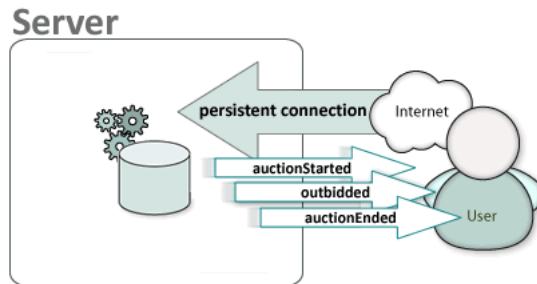
© JMA 2020. All rights reserved

Server-Sent Events

- La notificación del servidor al cliente es inmediata.
- Evita sondeos reiterados e innecesarios al servidor.
- Envía datos arbitrarios desde el servidor para el cliente, destinados a ser procesado por un script.
- Permite la actualización del contenido sin recargar la página.
- Útil para:
 - Chat en tiempo real o de juego
 - Correo electrónico
 - Actualizaciones en vivo

© JMA 2020. All rights reserved

Server-Sent Events



© JMA 2020. All rights reserved

Server-Sent Events

```
var sse = new EventSource('/sse/service');
sse.onopen = function () {
  initData();
};
sse.onmessage = function (event) {
  var data = JSON.parse(event.data);
  recibeData(data);
};
sse.onerror= function (event) {
  diplayError(event);
};
```

© JMA 2020. All rights reserved

WebSockets

<http://websocket.org/>

- La especificación WebSocket define un API que establece conexiones "socket" entre un navegador web y un servidor.
- Permite una conexión persistente entre el cliente y el servidor, y ambas partes pueden empezar a enviar datos en cualquier momento.
 - `var connection = new WebSocket('ws://echo.websocket.org/');`
- Cuando se establezca una conexión con el servidor (cuando el evento `open` se active), se puede empezar a enviar datos al servidor con el método `send('your message')` en el objeto de conexión.
- El servidor puede enviar mensajes en cualquier momento, que activa el evento `onmessage` que recibe un objeto "event" para acceder al mensaje actual mediante la propiedad `data`.
- WebSocket sigue siendo una tecnología joven y no está implementada completamente en todos los navegadores.
- Presentan problemas de compatibilidad con los servidores proxy que median las conexiones HTTP en la mayoría de las redes corporativas, con la petición HTTP de cambio de HTTP a WebSocket (normalmente utilizado para HTTP/SSL).

© JMA 2020. All rights reserved

JavaScript Object Notation
<http://tools.ietf.org/html/rfc4627>

JSON

© JMA 2020. All rights reserved

Introducción

- JSON (JavaScript Object Notation) es un formato sencillo para el intercambio de información.
- El formato JSON permite representar estructuras de datos (arrays) y objetos (arrays asociativos) en forma de texto.
- La notación de objetos mediante JSON es una de las características principales de JavaScript y es un mecanismo definido en los fundamentos básicos del lenguaje.
- En los últimos años, JSON se ha convertido en una alternativa al formato XML, ya que es más fácil de leer y escribir, además de ser mucho más conciso.
- No obstante, XML es superior técnicamente porque es un lenguaje de marcado, mientras que JSON es simplemente un formato para intercambiar datos.
- La especificación completa del JSON es la RFC 4627, su tipo MIME oficial es application/json y la extensión recomendada es .json.

© JMA 2020. All rights reserved

Estructuras

- JSON está constituido por dos estructuras:
 - Una colección de pares de nombre/valor. En los lenguajes esto es conocido como un objeto, registro, estructura, diccionario, tabla hash, lista de claves o un arreglo asociativo.
 - Una lista ordenada de valores. En la mayoría de los lenguajes, esto se implementa como tablas, arreglos, vectores, listas o secuencias.
- Estas son estructuras universales; virtualmente todos los lenguajes de programación las soportan de una forma u otra. Es razonable que un formato de intercambio de datos que es independiente del lenguaje de programación se base en estas estructuras.

© JMA 2020. All rights reserved

Sintaxis

- Un array es un conjunto de valores separados por comas (,) que se encierran entre corchetes [...]
- Un objeto es un conjunto de pares nombre:valor separados por comas (,) que se acotan entre llaves { ... }
- Los nombres son cadenas, entre comillas dobles (").
- El separador entre el nombre y el valor son los dos puntos (:)
- El valor debe ser un objeto, un array, un número, una cadena o uno de los tres nombres literales siguientes (en minúsculas):
 - true, false o null
- Se codifica en Unicode, la codificación predeterminada es UTF-8.

© JMA 2020. All rights reserved

Valores numéricos

- La representación de números es similar a la utilizada en la mayoría de los lenguajes de programación.
- Un número contiene una parte entera que puede ser prefijada con un signo menos opcional, que puede ser seguida por una parte fraccionaria y / o una parte exponencial.
- La parte fraccionaria comienza con un punto (como separador decimal) seguido de uno o más dígitos.
- La parte exponencial comienza con la letra E en mayúsculas o minúsculas, lo que puede ser seguido por un signo más o menos, y son seguidas por uno o más dígitos.
- Los formatos octales y hexadecimales no están permitidos. Los ceros iniciales no están permitidos.
- No se permiten valores numéricos que no se puedan representar como secuencias de dígitos (como infinito y NaN).

© JMA 2020. All rights reserved

Valores cadena

- La representación de las cadenas es similar a las convenciones utilizadas en la familia C de lenguajes de programación.
- Una cadena comienza y termina con comillas (").
- Se pueden utilizar todos los caracteres Unicode dentro de las comillas con excepción de los caracteres que se deben escapar: los caracteres de control (U + 0000 a U + 001F) y los caracteres con significado.
- Cuando un carácter se encuentra fuera del plano multilingüe básico (U + 0000 a U + FFFF), puede ser representado por su correspondiente valor hexadecimal. Las letras hexadecimales A-F puede ir en mayúsculas o en minúsculas.
- Secuencias de escape:
 - `\\, \/, \", \n, \r, \b, \f, \t`
 - `\u[0-9A-Fa-f]{4}`

© JMA 2020. All rights reserved

Objeto con anidamientos

```
{
  "Image": {
    "Width": 800,
    "Height": 600,
    "Title": "View from 15th Floor",
    "Thumbnail": {
      "Url": "/image/481989943",
      "Height": 125,
      "Width": "100"
    },
    "IDs": [116, 943, 234, 38793]
  }
}
```

© JMA 2020. All rights reserved

Array de objetos

```
[
  {
    "precision": "zip",
    "Latitude": 37.7668,
    "Longitude": -122.3959,
    "City": "SAN FRANCISCO",
    "State": "CA",
    "Zip": "94107"
  },
  {
    "precision": "zip",
    "Latitude": 37.371991,
    "Longitude": -122.026020,
    "City": "SUNNYVALE",
    "State": "CA",
    "Zip": "94085"
  }
]
```

© JMA 2020. All rights reserved

JSON en JavaScript

- El Standard Built-in ECMAScript Objects define que todo interprete de JavaScript debe contar con un objeto JSON como miembro del objeto Global.
- El objeto debe contener, al menos, los siguientes miembros:
 - **JSON.parse** (Función): Convierte una cadena de la notación de objetos de JavaScript (JSON) en un objeto de JavaScript.
 - **JSON.stringify** (Función): Convierte un valor de JavaScript en una cadena de la notación de objetos JavaScript (JSON).

© JMA 2020. All rights reserved



AJAX



© JMA 2020. All rights reserved

Introducción

- AJAX, acrónimo de Asynchronous JavaScript And XML (JavaScript asíncrono y XML), es una técnica de desarrollo web para crear aplicaciones 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, mejorando la interactividad, velocidad y usabilidad en las aplicaciones.
- Ajax es una tecnología asíncrona, en el sentido de que los datos adicionales se solicitan al servidor y se cargan en segundo plano sin interferir con la visualización ni el comportamiento de la página. Ajax no constituye una tecnología en sí, sino que es un término que engloba a un grupo de éstas que trabajan conjuntamente.
- JavaScript es el lenguaje interpretado (scripting language) en el que normalmente se efectúan las funciones de llamada de Ajax mientras que el acceso a los datos se realiza mediante XMLHttpRequest, objeto disponible en los navegadores actuales. En cualquier caso, no es necesario que el contenido asíncrono esté formateado en XML.
- Ajax es una técnica válida para múltiples plataformas y utilizable en muchos sistemas operativos y navegadores dado que está basado en estándares abiertos como JavaScript y Document Object Model (DOM).

© JMA 2020. All rights reserved

XMLHttpRequest

- XMLHttpRequest (XHR), también conocido como XMLHttpRequest (Extensible Markup Language / Hypertext Transfer Protocol), es una interfaz empleada para realizar peticiones HTTP y HTTPS a servidores Web.
- Para los datos transferidos se usa cualquier codificación basada en texto, incluyendo: texto plano, XML, JSON, HTML y codificaciones particulares específicas.
- El navegador implementa la interfaz como una clase de la que una aplicación cliente puede generar tantas instancias como necesite y permita el navegador para manejar el diálogo con el servidor.
- La primera versión de la interfaz XMLHttpRequest fue desarrollada por Microsoft que la introdujo en la versión 5.0 de Internet Explorer utilizando un objeto ActiveX. A partir de la versión 7 la interfaz se ofrece de manera integrada.
- El proyecto Mozilla incorporó la primera implementación integrada en la versión 1.0 de la Suite Mozilla en 2002. Esta implementación sería seguida por Apple a partir de Safari 1.2, Opera Software a partir del Opera 8.0 e iCab desde la versión 3.0b352.
- El World Wide Web Consortium presentó el 27 de septiembre de 2006 el primer borrador de una especificación estándar de la interfaz.

© JMA 2020. All rights reserved

Propiedades

Propiedad	Descripción	
readyState	0	No inicializado (objeto creado, pero no se ha invocado el método open)
	1	Cargando (objeto creado, pero no se ha invocado el método send)
	2	Cargado (se ha invocado el método send, pero el servidor aún no ha respondido)
	3	Interactivo (descargando, se han recibido algunos datos, aunque no se puede emplear la propiedad responseText)
	4	Completo (se han recibido todos los datos de la respuesta del servidor)
responseText	El contenido de la respuesta del servidor en forma de cadena de texto	
responseXML	El contenido de la respuesta del servidor en formato XML. El objeto devuelto se puede procesar como un objeto DOM	
status	El código de estado HTTP devuelto por el servidor (200 para una respuesta correcta, 404 para "No encontrado", 500 para un error de servidor, etc.)	
statusText	El código de estado HTTP devuelto por el servidor en forma de cadena de texto: "OK", "Not Found", "Internal Server Error", etc.	

© JMA 2020. All rights reserved

Métodos y eventos

Método	Descripción
abort()	Detiene la petición actual.
getAllResponseHeaders()	Devuelve una cadena de texto con todas las cabeceras de la respuesta del servidor.
getResponseHeader("cabecera")	Devuelve una cadena de texto con el contenido de la cabecera solicitada.
open("metodo", "url")	Establece los parámetros de la petición que se realiza al servidor. Los parámetros necesarios son el método HTTP empleado y la URL destino.
send(contenido)	Realiza la petición HTTP al servidor
setRequestHeader("cabecera", "valor")	Permite establecer cabeceras personalizadas en la petición HTTP. Se debe invocar entre el open() y el send().
onreadystatechange	Evento. Se invoca cada vez que se produce un cambio en el estado de la petición HTTP.

© JMA 2020. All rights reserved

Códigos HTTP (status)

status	statusText	Descripción
100	Continue	Una parte de la petición (normalmente la primera) se ha recibido sin problemas y se puede enviar el resto de la petición
101	Switching protocols	El servidor va a cambiar el protocolo con el que se envía la información de la respuesta. En la cabecera Upgrade indica el nuevo protocolo
200	OK	La petición se ha recibido correctamente y se está enviando la respuesta. Este código es con mucha diferencia el que mas devuelven los servidores
201	Created	Se ha creado un nuevo recurso (por ejemplo una página web o un archivo) como parte de la respuesta
202	Accepted	La petición se ha recibido correctamente y se va a responder, pero no de forma inmediata
203	Non-Authoritative Information	La respuesta que se envía la ha generado un servidor externo. A efectos prácticos, es muy parecido al código 200
204	No Content	La petición se ha recibido de forma correcta pero no es necesaria una respuesta
205	Reset Content	El servidor solicita al navegador que inicialice el documento desde el que se realizó la petición, como por ejemplo un formulario
206	Partial Content	La respuesta contiene sólo la parte concreta del documento que se ha solicitado en la petición

© JMA 2020. All rights reserved

Códigos de redirección

status	statusText	Descripción
300	Multiple Choices	El contenido original ha cambiado de sitio y se devuelve una lista con varias direcciones alternativas en las que se puede encontrar el contenido
301	Moved Permanently	El contenido original ha cambiado de sitio y el servidor devuelve la nueva URL del contenido. La próxima vez que solicite el contenido, el navegador utiliza la nueva URL
302	Found	El contenido original ha cambiado de sitio de forma temporal. El servidor devuelve la nueva URL, pero el navegador debe seguir utilizando la URL original en las próximas peticiones
303	See Other	El contenido solicitado se puede obtener en la URL alternativa devuelta por el servidor. Este código no implica que el contenido original ha cambiado de sitio
304	Not Modified	Normalmente, el navegador guarda en su caché los contenidos accedidos frecuentemente. Cuando el navegador solicita esos contenidos, incluye la condición de que no hayan cambiado desde la última vez que los recibió. Si el contenido no ha cambiado, el servidor devuelve este código para indicar que la respuesta sería la misma que la última vez
305	Use Proxy	El recurso solicitado sólo se puede obtener a través de un proxy, cuyos datos se incluyen en la respuesta
307	Temporary Redirect	Se trata de un código muy similar al 302, ya que indica que el recurso solicitado se encuentra de forma temporal en otra URL

© JMA 2020. All rights reserved

Códigos de error en la petición

status	statusText	Descripción
400	Bad Request	El servidor no entiende la petición porque no ha sido creada de forma correcta
401	Unauthorized	El recurso solicitado requiere autorización previa
402	Payment Required	Código reservado para su uso futuro
403	Forbidden	No se puede acceder al recurso solicitado por falta de permisos o porque el usuario y contraseña indicados no son correctos
404	Not Found	El recurso solicitado no se encuentra en la URL indicada. Se trata de uno de los códigos más utilizados y responsable de los típicos errores de <i>Página no encontrada</i>
405	Method Not Allowed	El servidor no permite el uso del método utilizado por la petición, por ejemplo por utilizar el método GET cuando el servidor sólo permite el método POST
406	Not Acceptable	El tipo de contenido solicitado por el navegador no se encuentra entre la lista de tipos de contenidos que admite, por lo que no se envía en la respuesta
407	Proxy Authentication Required	Similar al código 401, indica que el navegador debe obtener autorización del proxy antes de que se le pueda enviar el contenido solicitado
408	Request Timeout	El navegador ha tardado demasiado tiempo en realizar la petición, por lo que el servidor la descarta

© JMA 2020. All rights reserved

Códigos de error en la petición

status	statusText	Descripción
409	Conflict	El navegador no puede procesar la petición, ya que implica realizar una operación no permitida (como por ejemplo crear, modificar o borrar un archivo)
410	Gone	Similar al código 404. Indica que el recurso solicitado ha cambiado para siempre su localización, pero no se proporciona su nueva URL
411	Length Required	El servidor no procesa la petición porque no se ha indicado de forma explícita el tamaño del contenido de la petición
412	Precondition Failed	No se cumple una de las condiciones bajo las que se realizó la petición
413	Request Entity Too Large	La petición incluye más datos de los que el servidor es capaz de procesar. Normalmente este error se produce cuando se adjunta en la petición un archivo con un tamaño demasiado grande
414	Request-URI Too Long	La URL de la petición es demasiado grande, como cuando se incluyen más de 512 bytes en una petición realizada con el método GET
415	Unsupported Media Type	Al menos una parte de la petición incluye un formato que el servidor no es capaz procesar
416	Requested Range Not Suitable	El trozo de documento solicitado no está disponible, como por ejemplo cuando se solicitan bytes que están por encima del tamaño total del contenido
417	Expectation Failed	El servidor no puede procesar la petición porque al menos uno de los valores incluidos en la cabecera Expect no se pueden cumplir

© JMA 2020. All rights reserved

Códigos de error del servidor

status	statusText	Descripción
500	Internal Server Error	Se ha producido algún error en el servidor que impide procesar la petición
501	Not Implemented	Procesar la respuesta requiere ciertas características no soportadas por el servidor
502	Bad Gateway	El servidor está actuando de proxy entre el navegador y un servidor externo del que ha obtenido una respuesta no válida
503	Service Unavailable	El servidor está sobrecargado de peticiones y no puede procesar la petición realizada
504	Gateway Timeout	El servidor está actuando de proxy entre el navegador y un servidor externo que ha tardado demasiado tiempo en responder
505	HTTP Version Not Supported	El servidor no es capaz de procesar la versión HTTP utilizada en la petición. La respuesta indica las versiones de HTTP que soporta el servidor

© JMA 2020. All rights reserved

Pasos a seguir

1. Obtener XMLHttpRequest
2. Crear y asignar el controlador del evento onreadystatechange.
3. Abrir conexión: open(method, url, *async*):
4. Opcional. Añadir cabeceras.
5. Opcional. Serializar datos a enviar vía POST.
6. Enviar petición: send()

© JMA 2020. All rights reserved

Obtener XMLHttpRequest

- Los navegadores que siguen los estándares (Firefox, Safari, Opera, Internet Explorer 7+) implementan el objeto XMLHttpRequest de forma nativa, por lo que se puede obtener a través del objeto window. Los navegadores obsoletos (Internet Explorer 5 y 6) implementan el objeto XMLHttpRequest como un objeto de tipo ActiveX.

```
var xmlhttp;
if (window.XMLHttpRequest) {
    //El explorador implementa la interfaz de forma nativa
    xmlhttp = new XMLHttpRequest();
} else if (window.ActiveXObject) {
    //El explorador permite crear objetos ActiveX
    try {
        xmlhttp = new ActiveXObject("MSXML2.XMLHTTP");
    } catch (e) {
        try {
            xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
        } catch (e) {}
    }
}
if (!xmlhttp) {
    alert("No ha sido posible crear una instancia de XMLHttpRequest");
}
```

© JMA 2020. All rights reserved

Controlador del evento onreadystatechange

- readyState: cuando vale 4 (Completo: se han recibido todos los datos de la respuesta del servidor)
- status: cuando vale 200 (OK: La petición se ha recibido correctamente y se está enviando la respuesta)
- responseText: El contenido de la respuesta del servidor en forma de cadena de texto
- responseXML: El contenido de la respuesta del servidor en formato XML.

```
xmlhttp.onreadystatechange = function () {  
    if (xmlhttp.readyState == 4)  
        if (xmlhttp.status == 200) {  
            xmlDoc = xmlhttp.responseXML;  
            // Tratamiento de los datos recibidos  
        } else {  
            // Tratamiento de excepción  
        }  
};
```

© JMA 2020. All rights reserved

Enviar petición

- Abrir conexión: open(method, url, async):
 - method: Verbo HTTP (GET, POST, ...)
 - async: Opcional, marcar con false para comportamiento síncrono

```
xmlhttp.open("GET", "demo_get.asp?nocache=" + Math.random());  
xmlhttp.open("POST", "demo_post.asp");
```
- Opcional. Añadir cabeceras.

```
xmlhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
```
- Opcional. Serializar datos a enviar vía POST.
 - Reunir los datos a enviar en una cadena formada por pares "nombre=valor" concatenados con ampersand (&).

```
var nombre= document.getElementById("fldNombre");  
var datos="nombre="+encodeURIComponent(nombre)  
       +"&apellidos="+encodeURIComponent(apellidos);
```
- Enviar petición: send()

```
xmlhttp.send(datos);
```

© JMA 2020. All rights reserved

Texto plano, HTML y JavaScript

- Texto plano

```
var rslt = http_request.responseText;  
document.getElementById("myDiv").innerHTML=rslt;
```

- HTML

```
var rslt = http_request.responseText;  
document.getElementById("myDiv").innerHTML=rslt;
```

- JavaScript

```
var respuesta = http_request.responseText;  
eval(respuesta);
```

© JMA 2020. All rights reserved

XML

- Serializar:

```
var datos="<Datos>";  
datos+="<Nombre>"+nombre+"</Nombre>";  
datos+="<Apellidos>"+apellidos+"</Apellidos>";  
datos+="</Datos>";
```

- Recibir:

```
xmlDoc=xmlhttp.responseXML;  
txt="";  
x=xmlDoc.getElementsByTagName("Provincias");  
for (i=0;i<x.length;i++) {  
    txt=txt + x[i].childNodes[0].nodeValue + "<br>";  
}  
document.getElementById("myDiv").innerHTML=txt;
```

© JMA 2020. All rights reserved

JSON

- Serializar:

```
var datos= JSON.stringify(objeto_json);
var datos='{ "Datos": {';
datos+=' "Nombre":"' +nombre+' "';
datos+=', "Apellidos":"' +apellidos+' "';
datos+='}}';
```

- Recibir:

```
var respuesta = http_request.responseText;
var objeto_json= JSON.parse(respuesta).Provincias;
//var objeto_json = eval("(" +respuesta+")").Provincias;
txt="";
for (i=0;i<objeto_json.length;i++) {
    txt=txt + objeto_json[i].Nombre + "<br>";
}
document.getElementById("myDiv").innerHTML=txt;
```

© JMA 2020. All rights reserved

Detener las peticiones

```
...
// Fijar un temporizador que aborte la petición
var temporizador = setTimeout(function() {
    xmlhttp.abort();
    alert(...);
}, 18000); // 2 minutos
xmlhttp.onreadystatechange = function () {
    if (xmlhttp.readyState == 4) {
        clearTimeout(temporizador);
        if (xmlhttp.status == 200) {
            // Eliminar el temporizador, innecesario
        }
    }
}
...
```

© JMA 2020. All rights reserved

Indicador de descarga

```
...
var temporizador = setTimeout(function() {
    document.getElementById("trabajandoAJAX").style.display="none";
    xmlhttp.abort();
    alert(...);
}, 18000); // 30 segundos
// Muestra el indicador hasta ahora oculto
document.getElementById("trabajandoAJAX").style.display = "block";
xmlhttp.onreadystatechange = function () {
    if (xmlhttp.readyState == 4)
        try {
            if (xmlhttp.status == 200) {
                ...
            } finally {
                // Oculta el indicador
                document.getElementById("trabajandoAJAX").style.display="none";
            }
        }
    ...
}
```

© JMA 2020. All rights reserved

Seguridad

- La ejecución de aplicaciones JavaScript puede suponer un riesgo para el usuario que permite su ejecución.
- Por este motivo, los navegadores restringen la ejecución de todo código JavaScript a un entorno de ejecución limitado.
- Las aplicaciones JavaScript no pueden establecer conexiones de red con dominios distintos al dominio en el que se aloja la aplicación JavaScript.
- Los navegadores emplean un método estricto para diferenciar entre dos dominios ya que no permiten ni subdominios ni otros protocolos ni otros puertos.
- Si el código JavaScript se descarga desde la siguiente URL: <http://www.ejemplo.com>
- Las funciones y métodos incluidos en ese código no pueden acceder a:
 - <https://www.ejemplo.com/scripts/codigo2.js>
 - <http://www.ejemplo.com:8080/scripts/codigo2.js>
 - <http://scripts.ejemplo.com/codigo2.js>
 - <http://192.168.0.1/scripts/codigo2.js>
- La propiedad `document.domain` se emplea para permitir el acceso entre subdominios del dominio principal de la aplicación.

© JMA 2020. All rights reserved

JSONP (JSON con padding)

- JSONP es una técnica de comunicación utilizada en los programas JavaScript para realizar llamadas asíncronas a dominios diferentes. JSONP es un método concebido para suplir la limitación de AJAX entre dominios por razones de seguridad. Esta restricción no se aplica a la etiqueta `<script>` de HTML, para la cual se puede especificar en su atributo `src` la URL de un script alojado en un servidor remoto.
- En esta técnica se devuelve un objeto JSON envuelto en la llamada de una función (debe ser código JavaScript válido), la función ya debe estar definida en el entorno de JavaScript y se encarga de manipular los datos JSON.
`miJsonCallback ({ "Nombre": "Carmelo", "Apellidos": "Cotón" });`
- Por convención, el nombre de la función de retorno se suele especificar mediante un parámetro de la consulta, normalmente, utilizando `jsonp` o `callback` como nombre del campo en la solicitud al servidor.

```
<script type="text/javascript"
src="http://otrodominio.com/datos.json?callback=
miJsonCallback"></script>
```

© JMA 2020. All rights reserved

Insertar elemento script

- El uso de JSONP sólo tiene sentido si se quiere realizar una llamada asíncrona a otro dominio, por ello es necesario manipular el DOM para insertar un elemento `<script>` en la cabecera de la página, ya que una vez cargado el documento, no es posible escribir en él.

```
function loadScript (id, src, callback) {
    // Crear elemento
    var script = document.createElement("script");
    // Atributos del script
    script.setAttribute("type", "text/javascript");
    script.setAttribute("src", src + "?callback=" + callback);
    script.setAttribute("id", id);
    // Insertar script en la cabecera
    document.getElementsByTagName("head")[0] .appendChild(script);
}
```

© JMA 2020. All rights reserved

CORS

- Un recurso hace una solicitud HTTP de origen cruzado cuando solicita otro recurso de un dominio distinto al que pertenece.
- Por razones de seguridad, los exploradores restringen las solicitudes HTTP de origen cruzado iniciadas dentro de un script. Por ejemplo, XMLHttpRequest sigue la política de mismo-origen. Por lo que, una aplicación usando XMLHttpRequest solo puede hacer solicitudes HTTP a su propio dominio. Para mejorar las aplicaciones web, los desarrolladores pidieron a los proveedores de navegadores que permitieran a XMLHttpRequest realizar solicitudes de dominio cruzado.
- El Grupo de Trabajo de Aplicaciones Web del W3C recomienda el nuevo mecanismo de Intercambio de Recursos de Origen Cruzado (CORS, Cross-origin resource sharing). Los servidores deben indicar al navegador mediante cabeceras si aceptan peticiones cruzadas y con que características:
 - "Access-Control-Allow-Origin", "*"
 - "Access-Control-Allow-Headers", "Origin, X-Requested-With, Content-Type, Accept"
 - "Access-Control-Allow-Methods", "GET, POST, PUT, DELETE"

© JMA 2020. All rights reserved

Desactivar la seguridad de Chrome

- Pasos para Windows:
 - Localizar el acceso directo al navegador (icono) y crear una copia como "Chrome Desarrollo".
 - Botón derecho -> Propiedades -> Destino
 - Editar el destino añadiendo el parámetro al final. ej: "C:\Program Files (x86)\Google\Chrome\Application\chrome.exe" --disable-web-security
 - Aceptar el cambio y lanzar Chrome
- Para desactivar parcialmente la seguridad:
 - allow-file-access
 - allow-file-access-from-files
 - allow-cross-origin-auth-prompt
- Referencia a otros parametros:
 - <http://peter.sh/experiments/chromium-command-line-switches/>

© JMA 2020. All rights reserved

Fetch

- La API Fetch proporciona una interfaz JavaScript para acceder y manipular partes del canal HTTP, tales como peticiones y respuestas. También provee un método global `fetch()` que proporciona una forma fácil y lógica de obtener recursos de forma asíncrona por la red.
- Fetch proporciona una alternativa mejor al uso de `XMLHttpRequest` y puede ser empleada fácilmente por otras tecnologías como `Service Workers`. Fetch también aporta un único lugar lógico en el que definir otros conceptos relacionados con HTTP como CORS y extensiones para HTTP.

```
fetch('https://httpbin.org/get')  
  .then(response => response.json())  
  .then(body => console.log(body));
```

- Hay que tener en cuenta:
 - La promesa será resuelta si el servidor responde aunque sea con un estado HTTP de error 4xx o 5xx (con la propiedad `ok` a `false`), solo será rechazada ante un fallo de red o si algo impidió completar la solicitud.
 - Por defecto, fetch no enviará ni recibirá cookies del servidor.

© JMA 2020. All rights reserved

Response

- La interfaz `Response` representa la respuesta a una petición.
- Propiedades
 - `body`: Expone un `ReadableStream` con los contenidos del body.
 - `bodyUsed`: Almacena un `Boolean` en el cuál declara si el body ya fue enviado como respuesta anteriormente.
 - `headers`: Contiene el objeto `Headers` asociado con la respuesta.
 - `ok`: Contiene un estado indicando si la respuesta fue correcta (estado en el rango 200-299) o no.
 - `redirected`: Indica si la respuesta es o no el resultado de una redirección; eso es, su lista de URL tiene más de una entrada.
 - `status`: Contiene el código de estado de la respuesta (ej: 200 si fue OK).
 - `statusText`: Contiene el mensaje de estado correspondiente al código de estado (ej: OK para el 200).
 - `type`: Contiene el tipo de respuesta (ej: basic, cors).
 - `url`: Contiene la URL de respuesta.
 - `useFinalURL`: Contiene un valor booleano indicando si ésta es la URL final de respuesta.
- Define los métodos `arrayBuffer()`, `blob()`, `json()`, `text()` y `formData()` para extraer el body. Todos devuelven una promesa que resuelve una vez se completa el flujo del Body.

© JMA 2020. All rights reserved

Peticiones

- Para configurar la petición se utiliza el segundo parámetro del fetch() mediante un objeto con las opciones.

```
fetch(url, {
  method: 'POST', // *GET, POST, PUT, DELETE, etc.
  mode: 'cors', // no-cors, *cors, same-origin
  cache: 'no-cache', // *default, no-cache, reload, force-cache, only-if-cached
  credentials: 'same-origin', // include, *same-origin, omit
  headers: {
    'Content-Type': 'application/json' // 'application/x-www-form-urlencoded'
  },
  redirect: 'follow', // manual, *follow, error
  referrerPolicy: 'no-referrer', // no-referrer, *no-referrer-when-downgrade, origin, origin-when-cross-origin, same-origin, strict-origin, strict-origin-when-cross-origin, unsafe-url
  body: JSON.stringify(data) // requiere la cabecera "Content-Type"
}).then(
```

© JMA 2020. All rights reserved

Enviar datos

```
var formData = new FormData();
var fileField = document.querySelector("input[type='file']");

formData.append('username', 'abc123');
formData.append('avatar', fileField.files[0]);

fetch('https://example.com/profile/avatar', {
  method: 'PUT',
  headers: {
    'Content-Type': 'application/x-www-form-urlencoded',
  },
  body: formData
})
.then(response => response.json())
.catch(error => console.error('Error:', error))
.then(response => console.log('Success:', response));
```

© JMA 2020. All rights reserved

Tratar respuestas

- Una promise `fetch()` será rechazada con `TypeError` cuando se encuentre un error de red, aunque esto normalmente significa problemas de permisos o similares, y no cuando el servidor responde con un error (códigos estados HTTP 4xx o 5xx).

```
fetch('https://httpbin.org/image/png').then(function (response) {  
  if (response.ok) {  
    response.blob().then(function (miBlob) {  
      var milimagen = document.querySelector('img');  
      var objectURL = URL.createObjectURL(miBlob);  
      milimagen.src = objectURL;  
    }).catch(function (error) {  
      console.error('Error en los datos recibidos: ' + error.message);  
    });  
  } else {  
    console.error('Error ' + response.status + ': ' + response.statusText);  
  }  
}).catch(function (error) {  
  console.error('Hubo un problema con la petición Fetch:' + error.message);  
});
```

© JMA 2020. All rights reserved

REpresentational State Transfer

SERVICIOS RESTFUL

© JMA 2020. All rights reserved

REST (REpresentational State Transfer)

- Un **estilo de arquitectura** para desarrollar aplicaciones web distribuidas que se basa en el uso del protocolo HTTP e Hypermedia.
- Definido en el 2000 por Roy Fielding, para no reinventar la rueda, se basa en aprovechar lo que ya estaba definido en el HTTP pero que no se utilizaba.
- El HTTP ya define 8 métodos (algunas veces referidos como "verbos") que indica la acción que desea que se efectúe sobre el recurso identificado:
 - HEAD, GET, POST, PUT, DELETE, TRACE, OPTIONS, CONNECT
- El HTTP permite en el encabezado transmitir la información de comportamiento:
 - Accept, Content-type, Response (códigos de estado), Authorization, Cache-control, ...

© JMA 2020. All rights reserved

Objetivos de los servicios REST

- Desacoplar el cliente del backend
- Mayor escalabilidad
 - Sin estado en el backend.
- Separación de problemas
- División de responsabilidades
- API uniforme para todos los clientes
 - Disponer de una interfaz uniforme (basada en URIs)

© JMA 2020. All rights reserved

Uso de la cabecera

- **Request:** Método /uri?parámetros
 - GET: Recupera el recurso
 - Todos: Sin parámetros
 - Uno: Con parámetros
 - POST: Crea un nuevo recurso
 - PUT: Edita el recurso
 - DELETE: Elimina el recurso
- **Accept:** Indica al servidor el formato o posibles formatos esperados, utilizando MIME.
- **Content-type:** Indica en que formato está codificado el cuerpo, utilizando MIME
- **Response:** Código de estado con el que el servidor informa del resultado de la petición.

© JMA 2020. All rights reserved

Peticiones

- Request: GET /users
 - Response: 200
 - content-type:application/json
- Request: GET /users/11
 - Response: 200
 - content-type:application/json
- Request: POST /users
 - Response: 201 Created
 - content-type:application/json
 - body
- Request: PUT /users/11
 - Response: 200
 - content-type:application/json
 - body
- Request: DELETE /users/11
 - Response: 204 No Content

Fake Online REST API for Testing and Prototyping
<https://jsonplaceholder.typicode.com/>

© JMA 2020. All rights reserved

Richardson Maturity Model

<http://www.crummy.com/writing/speaking/2008-QCon/act3.html>

- # Nivel 1 (Pobre): Se usan URLs para identificar recursos:
 - Se debe identificar un recurso
`/invoices/?page=2` → `/invoices/page/2`
 - Se construyen con nombres nunca con verbos
`/getUser/{id}` → `/users/{id}/`
`/users/{id}/edit/login` → `users/{id}/access-token`
 - Deberían tener una estructura jerárquica
`/invoices/user/{id}` → `/user/{id}/invoices`
- # Nivel 2 (Medio): Se usa el protocolo HTTP adecuadamente
- # Nivel 3 (Óptimo): Se implementa hypermedia.

© JMA 2020. All rights reserved

Hypermedia

- Se basa en la idea de enlazar recursos: propiedades que son enlaces a otros recursos.
- Para que sea útil, el cliente debe saber que en la respuesta hay contenido hypermedia.
- En content-type es clave para esto
 - Un tipo genérico no aporta nada:
`Content-Type: text/xml`
 - Se pueden crear tipos propios
`Content-Type: application/servicio+xml`

© JMA 2020. All rights reserved

JSON Hypertext Application Language

- RFC4627 <http://tools.ietf.org/html/draft-kelly-json-hal-00>
- Content-Type: application/hal+json

```
{
  "_links": {
    "self": {"href": "/orders/523" },
    "warehouse": {"href": "/warehouse/56" },
    "invoice": {"href": "/invoices/873"}
  },
  "currency": "USD"
  , "status": "shipped"
  , "total": 10.20
}
```

© JMA 2020. All rights reserved

Patrón Agregado (Aggregate)

- Una Agregación es un grupo de objetos asociados que deben tratarse como una unidad a la hora de manipular sus datos.
- El patrón Agregado es ampliamente utilizado en los modelos de datos basados en Diseños Orientados al Dominio (DDD).
- Proporciona un forma de encapsular nuestras entidades y los accesos y relaciones que se establecen entre las mismas de manera que se simplifique la complejidad del sistema en la medida de lo posible.
- Cada Agregación cuenta con una Entidad Raíz (root) y una Frontera (boundary):
 - La Entidad Raíz es una Entidad contenida en la Agregación de la que colgarán el resto de entidades del agregado y será el único punto de entrada a la Agregación.
 - La Frontera define qué está dentro de la Agregación y qué no.
- La Agregación es la unidad de persistencia, se recupera toda y se almacena toda.

© JMA 2020. All rights reserved

REpresentational State Transfer

SERVICIOS RESTFUL

© JMA 2020. All rights reserved

Objetivos

-
- Desacoplar el cliente del backend
 - Mayor escalabilidad
 - Sin estado en el backend.
 - Separación de problemas
 - División de especialidades
 - API uniforme para todos los clientes
 - Disponer de una interfaz uniforme (basada en URIs)
-

© JMA 2020. All rights reserved

REST (REpresentational State Transfer)

- Un **estilo de arquitectura** para desarrollar aplicaciones web distribuidas que se basa en el uso del protocolo HTTP e Hypermedia.
- Definido en el 2000 por Roy Fielding, para no reinventar la rueda, se basa en aprovechar lo que ya estaba definido en el HTTP pero que no se utilizaba.
- El HTTP ya define 8 métodos (algunas veces referido como "verbos") que indica la acción que desea que se efectúe sobre el recurso identificado:
 - HEAD, GET, POST, PUT, DELETE, TRACE, OPTIONS, CONNECT
- El HTTP permite en el encabezado transmitir la información de comportamiento:
 - Accept, Content-type, Response (códigos de estado), Authorization, Cache-control, ...

© JMA 2020. All rights reserved

Uso de la cabecera

- Request: Método /uri?parámetros
 - GET: Recupera el recurso
 - Todos: Sin parámetros
 - Uno: Con parámetros
 - POST: Crea un nuevo recurso
 - PUT: Edita el recurso
 - DELETE: Elimina el recurso
- Accept: Indica al servidor el formato o posibles formatos esperados, utilizando MIME.
- Content-type: Indica en que formato está codificado el cuerpo, utilizando MIME
- Response: Código de estado con el que el servidor informa del resultado de la petición.

© JMA 2020. All rights reserved

Peticiones

Request: GET /users
Response: 200
content-type:application/json
Request: GET /users/11
Response: 200
content-type:application/json
Request: POST /users
Response: 201
content-type:application/json
Request: PUT /users/11
Response: 200
content-type:application/json
Request: DELETE /users/11
Response: 204 no content

© JMA 2020. All rights reserved

Encabezado HTTP Cache-Control

- El encabezado HTTP Cache-Control especifica directivas (instrucciones) para almacenar temporalmente (caching) tanto en peticiones como en respuestas. Una directiva dada en una petición no significa que la misma directiva estar en la respuesta.
- Los valores estándar que pueden ser usados por el servidor en una respuesta HTTP son:
 - public: La respuesta puede estar almacenada en cualquier memoria cache.
 - private: La respuesta puede estar almacenada sólo por el cache de un navegador.
 - no-cache: La respuesta puede estar almacenada en cualquier memoria cache pero DEBE pasar siempre por una validación con el servidor de origen antes de utilizarse.
 - no-store: La respuesta puede no ser almacenada en cualquier cache.
 - max-age=<seconds>: La cantidad máxima de tiempo un recurso es considerado reciente.
 - s-maxage=<seconds>: Anula el encabezado max-age o el Expires, pero solo para caches compartidos (e.g., proxies).
 - must-revalidate: Indica que una vez un recurso se vuelve obsoleto, el cache no debe usar su copia obsoleta sin validar correctamente en el servidor de origen.
 - proxy-revalidate: Similar a must-revalidate, pero solo para caches compartidos (es decir, proxies). Ignorado por caches privados.
 - no-transform: No deberían hacerse transformaciones o conversiones al recurso.

© JMA 2020. All rights reserved

Encabezados HTTP ETag, If-Match y If-None-Match

- El encabezado de respuesta de HTTP ETag es un identificador (resumen hash) para una versión específica de un recurso y los encabezados If-Match e If-None-Match de la solicitud HTTP hace que la solicitud sea condicional.
- Para los métodos GET y HEAD con If-None-Match: si el ETag no coincide con los datos, el servidor devolverá el recurso solicitado con un estado 200, si coincide el servidor debe devolver el código de estado HTTP 304 (No modificado) y DEBE generar cualquiera de los siguientes campos de encabezado que se habrían enviado en una respuesta 200 (OK) a la misma solicitud: Cache-Control, Content-Location, Date, ETag, Expires y Vary.
- Para los métodos PUT y DELETE con If-Match: si el ETag coincide con los datos, se realiza la actualización o borrado y se devuelve un estado HTTP 204 (sin contenido) incluyendo el Cache-Control y el ETag de la versión actualizada del recurso en el PUT. Si no coinciden, se ha producido un error de concurrencia, la versión del servidor ha sido modificada desde que la recibió el cliente, debe devolver una respuesta HTTP con un cuerpo de mensaje vacío y un código de estado 412 (Precondición fallida).
- Si los datos solicitados ya no existen, el servidor debe devolver una respuesta HTTP con el código de estado 404 (no encontrado).

© JMA 2020. All rights reserved

Richardson Maturity Model

<http://www.crummy.com/writing/speaking/2008-QCon/act3.html>

- Nivel 0: Definir un URI y todas las operaciones son solicitudes POST a este URI.
- Nivel 1 (Pobre): Crear distintos URI para recursos individuales pero utilizan solo un método.
 - Se debe identificar un recurso
/entities/?invoices=2 → entities/invoices/2
 - Se construyen con nombres nunca con verbos
/getUser/{id} → /users/{id}/
/users/{id}/edit/login → users/{id}/access-token
 - Deberían tener una estructura jerárquica
/invoices/user/{id} → /user/{id}/invoices
- Nivel 2 (Medio): Usar métodos HTTP para definir operaciones en los recursos.
- Nivel 3 (Óptimo): Usar hipermedia (HATEOAS, se describe a continuación).

© JMA 2020. All rights reserved

Hypermedia

- Se basa en la idea de enlazar recursos: propiedades que son enlaces a otros recursos.
- Para que sea útil, el cliente debe saber que en la respuesta hay contenido hypermedia.
- En content-type es clave para esto
 - Un tipo genérico no aporta nada:
Content-Type: text/xml
 - Se pueden crear tipos propios
Content-Type: application/servicio+xml

© JMA 2020. All rights reserved

JSON Hypertext Application Language

- RFC4627 <http://tools.ietf.org/html/draft-kelly-json-hal-00>
- Content-Type: application/hal+json

```
{
  "_links": {
    "self": {"href": "/orders/523" },
    "warehouse": {"href": "/warehouse/56" },
    "invoice": {"href": "/invoices/873"}
  },
  "currency": "USD"
  , "status": "shipped"
  , "total": 10.20
}
```

© JMA 2020. All rights reserved

JAVASCRIPT FRAMEWORK

© JMA 2020. All rights reserved

JavaScript Framework

- En el ámbito del desarrollo del software, el término “JavaScript Framework” significa una biblioteca que proporciona a los desarrolladores plantillas preconstruidas y código JavaScript preescrito para tareas de programación estándar. Millones de desarrolladores usan este tipo de frameworks para acelerar el flujo de trabajo de desarrollo y aplicar las mejores prácticas de una manera fluida y fácil.
- Debido a su gran variedad, su uso depende de los objetivos principales, la funcionalidad general de la plataforma, los requisitos del proyecto y cómo se puede implementar dentro de cada escenario específico.
- Al usar este tipo de marcos desarrollo para JavaScript, se puede ahorrar una gran cantidad de tiempo y esfuerzo en el desarrollo de sitios web y aplicaciones basados en este lenguaje. Simplifica todo el procedimiento y permite a los desarrolladores crear aplicaciones web a gran escala de manera eficiente.
- Los frameworks de JavaScript son una parte esencial del desarrollo web front-end moderno, los cuales proveen a los desarrolladores herramientas probadas y testeadas para la creación de aplicaciones web interactivas y escalables. Muchas empresas modernas utilizan frameworks como parte estándar de sus herramientas, por lo que muchos trabajos de desarrollo front-end en la actualidad requieren experiencia en frameworks.

© JMA 2020. All rights reserved

Clasificación

- Por Tipos:
 - Estructurales
 - Específicos
- Por Aspectos:
 - Presentación
 - Plantillas
 - Enlazado
 - Gráficos, animaciones, ...
 - Componentes
 - Comunicaciones
 - Enrutado
 - Arquitectónicas
 - Mantenimiento de Estado
 - Inyección de dependencias
 - Modelos de programación: reactiva, funcional, modularidad, ...
 - Testing y QA

<https://github.com/sorrycc/awesome-javascript>

© JMA 2020. All rights reserved

Generación del esqueleto de aplicación

- Configurar un nuevo proyecto web puede ser un proceso complicado y tedioso, con tareas como:
 - Crear la estructura básica de archivos y bootstrap
 - Configurar SystemJS o WebPack para transpilar el código
 - Crear scripts para ejecutar el servidor de desarrollo, tester, publicación, ...
- Disponemos de diferentes opciones de asistencia:
 - Proyectos semilla (seed) disponibles en github
 - Generadores basados en Yeoman
 - Herramientas oficiales de los Framework (CLI):
 - Los *Command Line Interface* (CLI) permiten generar proyectos y elementos de código desde consola, así como ejecutar un servidor de desarrollo o lanzar los tests de la aplicación.

© JMA 2020. All rights reserved

Agrupar y minificar recursos

- La agrupación y la minificación (minimización) son dos técnicas para mejorar el tiempo de carga de la solicitud. La agrupación y minificación mejora el tiempo de carga al reducir el número de solicitudes al servidor y reducir el tamaño de los activos solicitados (como CSS y JavaScript).
- La mayoría de los exploradores principales actuales limitan el número de conexiones simultáneas por cada nombre de host. Esto significa que, mientras se procesan las primeras solicitudes, el explorador pondrá en cola las solicitudes adicionales, a la espera de que vayan recibiendo las respuestas. La agrupación permite combinar varios archivos JavaScript (.js) o varios archivos de hoja de estilos en cascada (.css) para que se puedan descargar como una unidad, en lugar de individualmente.
- Se entiende por "minificación" el proceso mediante el cual se eliminan datos innecesarios o redundantes de un recurso sin que se vea afectada la forma en que los navegadores lo procesan. Por ejemplo, eliminar comentarios y formato innecesario, retirar código que no se usa, acortar los nombres de variables y funciones a un carácter, ...
- Es aconsejable minimizar los recursos de HTML, CSS y JavaScript.

© JMA 2020. All rights reserved

Vainilla (Vanilla JS) vs biblioteca

- Los componentes web pueden resultar difíciles de escribir desde cero. Hay mucho en qué pensar y escribir, un componente puede requerir una gran cantidad de código repetitivo. Afortunadamente, existen algunas bibliotecas excelentes que pueden hacer que la creación de elementos personalizados sea más sencilla y ahorrarle mucho tiempo y esfuerzo.
- Es importante tener en cuenta que no es necesario utilizar una biblioteca para crear y compartir un elemento personalizado. Los elementos personalizados sin procesar son excelentes si su tarea se limita a uno o unos pocos elementos, ya que permiten una implementación más optimizada y evitan el bloqueo de la biblioteca. [Vanilla.JS](#) es una broma creada por Eric Wastl que promueve la vuelta a las esencias.
- Sin embargo, si se está escribiendo muchísimos elementos personalizados, el uso de una biblioteca puede hacer que su código sea más simple y limpio, y que su flujo de trabajo sea más eficiente.
- Una buena biblioteca de componentes web debería producir un componente web que "simplemente funcionen" como cualquier otro elemento HTML. Las buenas bibliotecas también tienen una alta relación valor-carga útil, es decir, proporcionan mucho valor con el mínimo tamaño de su descarga.

© JMA 2020. All rights reserved

Vanilla JS

- Vanilla JS actualmente es el framework más usado en Internet, siendo usado por Google, Amazon, Twitter y muchas otras webs.
- A la hora de acceder al DOM por IDs es el doble de rápido que Dojo y 100 veces más rápido que Prototype JS cuando accedemos a los elementos por el nombre de la etiqueta, siendo estos unos de los framework más eficiente en cuanto a acceder al DOM, solo superado por Vanilla JS.
- Vanilla JavaScript es como se conoce al lenguaje JavaScript cuando se utiliza sin ninguna librería o framework. La traducción más castellana sería JavaScript a pelo.
- Vanilla JS es una iniciativa, en forma de framework que intenta enseñar las grandes ventajas de no usar frameworks y potenciar nuestras aplicaciones sin necesidad de añadir grandes archivos extra.
- No dicen que los framework sean malos, simplemente que hay que descargarlos, interpretarlos y reducen el rendimiento (mejoran la productividad) aumentando los consumos, en muchas ocasiones se abusa de estos frameworks para cosas que no son necesarias y que se pueden llevar a cabo sin ellos.

© JMA 2020. All rights reserved

JavaScript Framework

- **MooTools** (My object oriented tools) es un Framework web orientado a objetos para JavaScript, de código abierto, compacto y modular. Aporta una manera de desarrollar JavaScript sin importar en qué navegador se ejecute de una manera elegante.
- **Prototype** es un framework escrito en JavaScript que se orienta al desarrollo sencillo y dinámico de aplicaciones web. Es una herramienta que implementa las técnicas AJAX y su potencial es aprovechado al máximo cuando se desarrolla con Ruby On Rails.
- **jQuery** es una biblioteca de JavaScript, permite simplificar la manera de interactuar con los documentos HTML, manipular el árbol DOM, manejar eventos, desarrollar animaciones y agregar interacción con la técnica AJAX a páginas web.
- **jQuery UI** es una biblioteca de componentes para el framework jQuery que le añaden un conjunto de plug-ins, widgets y efectos visuales para la creación de aplicaciones web. Cada componente o módulo se desarrolla de acuerdo a la filosofía de jQuery5 (find something, manipulate it: encuentra algo, manipúlalo).
- **Dojo** es un framework que contiene APIs y widgets (controles) para facilitar el desarrollo de aplicaciones Web que utilicen tecnología AJAX. Contiene un sistema de empaquetado inteligente, los efectos de UI, drag and drop APIs, widget APIs, abstracción de eventos, almacenamiento de APIs en el cliente, e interacción de APIs con AJAX.

© JMA 2020. All rights reserved

JavaScript Framework

- **Twitter Bootstrap** es un framework enfocado en el diseño adaptativo que permite construir sitios web rápidamente con estilos, plantillas y funciones predefinidas puestas a disposición del desarrollador.
- **AngularJS** es un framework de JavaScript de código abierto, que ayuda con la gestión de lo que se conoce como aplicaciones de una sola página. Su objetivo es aumentar las aplicaciones basadas en navegador con capacidad de Modelo Vista Controlador (MVC), en un esfuerzo para hacer que el desarrollo y las pruebas sean más fáciles.
- **Backbone** es una herramienta de desarrollo/API para el lenguaje de programación Javascript con un interfaz RESTful por JSON, basada en el paradigma de diseño de aplicaciones Modelo Vista Controlador. Está diseñada para desarrollar aplicaciones de una única página y para mantener las diferentes partes de las aplicaciones web sincronizadas.
- **Ext JS** es una biblioteca de JavaScript para el desarrollo de aplicaciones web interactivas usando tecnologías como AJAX, DHTML y DOM.
- **Node.js** es un entorno de programación en la capa del servidor (pero no limitándose a ello) basado en el lenguaje de programación ECMAScript, asíncrono, con I/O de datos en una arquitectura orientada a eventos y basado en el motor V8 de Google. Fue creado con el enfoque de ser útil en la creación de programas de red altamente escalables, como por ejemplo, servidores web.

© JMA 2020. All rights reserved

Agrupar y minificar recursos

- La agrupación y la minificación (minimización) son dos técnicas para mejorar el tiempo de carga de la solicitud. La agrupación y minificación mejora el tiempo de carga al reducir el número de solicitudes al servidor y reducir el tamaño de los activos solicitados (como CSS y JavaScript).
- La mayoría de los exploradores principales actuales limitan el número de conexiones simultáneas por cada nombre de host. Esto significa que, mientras se procesan las primeras solicitudes, el explorador pondrá en cola las solicitudes adicionales, a la espera de que vayan recibiendo las respuestas. La agrupación permite combinar varios archivos JavaScript (.js) o varios archivos de hoja de estilos en cascada (.css) para que se puedan descargar como una unidad, en lugar de individualmente.
- Se entiende por "minificación" el proceso mediante el cual se eliminan datos innecesarios o redundantes de un recurso sin que se vea afectada la forma en que los navegadores lo procesan. Por ejemplo, eliminar comentarios y formato innecesario, retirar código que no se usa, acortar los nombres de variables y funciones a un carácter, ...
- Es aconsejable minimizar los recursos de HTML, CSS y JavaScript.

© JMA 2020. All rights reserved

Bibliotecas para construir componentes web

- [Hybrids](#) es una biblioteca de interfaz de usuario para crear componentes web con API simple y funcional.
- [Polymer](#) proporciona un conjunto de funciones para crear elementos personalizados.
- [LitElement](#) usa [lit-html](#) para representar en el Shadow DOM del elemento y agrega API para ayudar a administrar las propiedades y atributos del elemento.
- [Slim.js](#) es una biblioteca de componentes web liviana de código abierto que proporciona enlace de datos y capacidades extendidas para componentes, utilizando la herencia de clases nativa de es6.
- [Stencil](#) es un compilador de código abierto que genera componentes web que cumplen con los estándares.
- Los [Angular Elements](#) son componentes Angular empaquetados como Web Components.
- Implementaciones con diferentes frameworks
 - <https://github.com/shprink/web-components-todo>
 - <https://github.com/thinktecture-labs/web-components-chat>

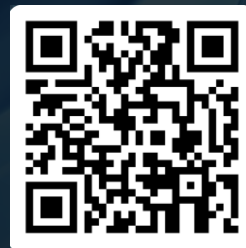
© JMA 2020. All rights reserved

Icono Training Consulting

JavaScript



Completa nuestra encuesta
de satisfacción a través del QR



GRACIAS



2024