

Javascript MVC, Angular y Node.js

FORMADORES { IT }

info@formadoresfreelance.es · www.formadoresit.es



Ayuntamiento de
FUENLABRADA

Índice

Creación de páginas web dinámicas	2
Ventaja	8
Desventajas	8
El lenguaje JavaScript	9
Pero, ¿qué es JavaScript?	10
Incluir scripts en páginas y archivos .js	11
Escribir JavaScript en el propio documento HTML	11
Definir JavaScript en un archivo externo a nuestro documento	12
Incluir JavaScript dentro de las propias etiquetas HTML	13
Sintaxis de lenguaje	13
Variables	15
Tipos de datos	17
Numéricos	17
Cadenas de texto	17
Arrays	20
Booleanos	21
Operadores	21
Asignación	21
Incremento y decremento	22
Lógicos	23
Negación	23
AND	23
OR	24
Matemáticos	24
Relacionales	24
Objetos del navegador	26
El Objeto window	27
El Objeto location	28
El Objeto screen	28

El Objeto document - La página en sí.....	29
El Objeto history	32
El Objeto navigator.....	32
Control de flujo.....	43
Estructura if.....	35
Estructura if...else.....	36
Estructura for	36
Estructura for...in.....	37
Funciones.....	38
Funciones útiles para cadenas de texto	38
Funciones útiles para arrays.....	40
Funciones útiles para números	41
El ámbito de las variables	42
Eventos y su manejo mediante funciones	44
Manejadores de eventos como atributos HTML	47
Manejadores de eventos y variable this.....	47
Manejadores de eventos como funciones externas	48
Manejadores de eventos semánticos.....	48
Objetos en Javascript y la especificación JSON	49
Particularidades de JSON sobre Javascript.....	49
Transformar JSON a String y String a JSON.....	51
El estándar DOM.....	52
Manipulación de un documento HTML con DOM y JavaScript.....	53
Selectores y DOM.....	56
Ajax	58
Funcionamiento y tecnologías implicadas.....	58
Tecnologías que forman AJAX	58
Funcionamiento de AJAX	59
Peticiones ajax y manejo de respuestas.....	60
Un ejemplo práctico	62
XML vs. JSON.....	63
La librería JQuery	65

Descarga y utilización	65
Ventajas al usar JQuery.....	66
Manera de desarrollar con JQuery	66
Selectores.....	67
Selector por ID	67
Selector por tipo.....	67
Selector por clase	68
Selectores anidados.....	68
Agrupación de selectores	69
Funciones principales.....	71
Manipulación de los elementos del DOM	72
Tratamiento de eventos con JQuery.....	73
Ajax con JQuery	75
Funciones para CSS	78
La librería AngularJS	79
Descarga de la librería	81
Principios de funcionamiento y elementos básicos.....	82
Directivas.....	82
ngApp (ng-app)	83
ngController (ng-controller).....	83
ngModel (ng-model).....	83
ngClick (ng-click)	84
ngInit (ng-init).....	84
ngRepeat (ng-repeat).....	84
ngChange (ng-change).....	85
ngShow (ng-show) ngHide (ng-hide).....	85
ngBind (ng-bind)	86
Creando nuestras propias directivas	86
Controllers (Controladores).....	87
Controladores en AngularJS	87
Propiedades del Controlador	88
Métodos del Controlador	89

Controladores en ficheros externos.....	89
Módulos.....	91
Ejemplo de Módulo en AngularJS.....	91
Los Controladores ensucian el ámbito Global de JavaScript	91
Definición de los Módulos.....	93
Ficheros de una Aplicación AngularJS	94
JavaScript en el lado del servidor: NodeJS.....	95
JavaScript del lado del servidor	95
Instalación y utilización.....	95
Instalación.....	95
Utilización.....	97
Creación de un servidor HTTP.....	97
¿Qué es el módulo express?.....	100
Instalación.....	100
Creando rutas	101
Recibiendo parámetros	101
Recibiendo POST.....	102
Usando expresiones regulares como ruta	102
Módulos principales.....	105
console	105
timers.....	105
module.....	105
buffer.....	106
util	106
events.....	106
stream	106
crypto	106
tls	106
string_decoder.....	106
fs	106
path.....	107
net	107

dgram.....	107
dns	107
http	107
https	107
url.....	107
querystrings	107
readline.....	107
repl.....	108
vm	108
child_process.....	108
assert.....	108
tty.....	108
zlib.....	108
os.....	108
_debugger.....	108
cluster.....	108
punycode.....	109
domain.....	109
_linklist.....	109
buffer_ieee754.....	109
constants.....	110
freelist.....	110
sys	110
Combinación de tecnologías en la creación de una página web	111
Diferencia entre "frontend" y "backend"	111
Arquitectura de la aplicación web SPA + API REST	111
Estructura de archivos.....	112
Implementando nuestro API REST	113
Creación del modelo de datos	114
Definición de rutas o Endpoints del API.....	114
Desarrollo de la parte Frontend con AngularJS	117

Creación de páginas web dinámicas

Una página web dinámica es una página que se actualiza conforme el usuario va haciendo peticiones, navegando por la página o actualizando su contenido.

Suelen venir cargadas de alto contenido visual, opciones para discapacitados o aprendizaje de las elecciones que ha ido tomando el usuario.

En contra a lo que ocurre con las páginas estáticas, en las que su contenido e información se encuentra predeterminado, en las páginas dinámicas la información va apareciendo según el ciclo de vida del usuario en la aplicación.

Esto se hace posible porque una página dinámica tiene asociada una aplicación web o una Base de Datos desde la que se permite visualizar el contenido.

Para la creación de este tipo de páginas deberán utilizarse etiquetas HTML y algún lenguaje de programación que se ejecute tanto del “lado servidor” como del “lado cliente”.

Los lenguajes utilizados para la creación de este tipo de páginas son principalmente: ASP, PHP, JSP, pero, sobre todo, mucho Javascript (JS).

Actualmente, Javascript (JS) ha experimentado un avance sorprendente, debido a la aparición de numerosos frameworks de desarrollo web basados en su lenguaje. En este manual trataremos algunos de ellos, como JQuery o AngularJS.



Ilustración 1: Esquema de funcionamiento de una página web dinámica

Ventajas

- El proceso de actualización y creación es sumamente sencillo, sin necesidad de entrar en el servidor.
- Gran número de funcionalidades y desarrollos tales como bases de datos, foros, contenido dinámico, etc.
- Facilitan tener actualizada diariamente toda la información.
- Diferentes áreas de diferentes empresas pueden participar en la creación y el mantenimiento.
- Dominación total sobre la administración de todos los contenidos.
- Contenidos fácilmente reutilizables.
- Una mayor interactividad con el usuario.
- Presentación de contenidos en diversos dispositivos y formatos, como los terminales móviles.
- Los autores del contenido no requieren conocimientos técnicos.

Desventajas

- Mayores requerimientos técnicos para su alojamiento en Servidores de pago y, por tanto, costes de alojamiento mayores.
- En algunos casos, un mayor coste de desarrollo que implican mayor cantidad de recursos en el apartado visual de la aplicación.

El lenguaje JavaScript

JavaScript, que no debe confundirse con Java, fue creado en 10 días en mayo de 1995 por Brendan Eich, que entonces trabajaba en Netscape y ahora de Mozilla.

JavaScript no siempre fue conocido como JavaScript: el nombre original era Mocha, un nombre elegido por Marc Andreessen, fundador de Netscape.

En septiembre de 1995 el nombre fue cambiado a LiveScript, a continuación, en diciembre del mismo año, al recibir una licencia de marca de Sun, se adoptó el nombre de JavaScript. Esto fue un movimiento de marketing en ese momento, con Java que era muy popular en todo entonces.

En 1996 - 1997 JavaScript fue llevado a ECMA para labrarse una especificación estándar, que otros proveedores de navegadores entonces podrían implementar basado en el trabajo realizado en Netscape. El trabajo realizado en este período de tiempo finalmente llevó a la liberación oficial de ECMA-262 Ed.1: ECMAScript es el nombre de la norma oficial, siendo JavaScript la más conocida de las implementaciones. ActionScript 3 es otra aplicación bien conocida de ECMAScript.

El proceso de las normas continuó en ciclos, con los lanzamientos de ECMAScript 2 en 1998 y ECMAScript 3 en 1999, que es la línea base para el moderno JavaScript. El trabajo dirigido por Waldemar Horwat (entonces de Netscape, ahora en Google) se inició en 2000 y en un primer momento, Microsoft parecía a participar (e incluso implementar) algunas de las propuestas en su idioma JScript.net.

El próximo evento importante fue en 2005, con dos grandes acontecimientos en la historia de JavaScript. En primer lugar, Brendan Eich y Mozilla reincorporaron Ecma como miembro sin fines de lucro y el trabajo comenzó en E4X, ECMA-357, que venía de ex empleados de Microsoft en BEA (originalmente adquirido como Crossgain). Esto llevó a trabajar en forma conjunta con Macromedia, que estaban implementando E4X en ActionScript 3.

Así, junto con Macromedia (posteriormente adquirida por Adobe), el trabajo se reinicia en ECMAScript 4 con el objetivo de estandarizar lo que había en AS3 y aplicarlo en SpiderMonkey. Con este fin, Adobe lanzó el "AVM2", cuyo nombre en código Tamarin, como un proyecto de código abierto. Pero Tamarin y AS3

eran demasiado diferentes de Web JavaScript para converger, como se dieron cuenta las partes en 2007 y 2008.

Por desgracia, todavía había confusión entre los diferentes actores; Doug Crockford - luego a Yahoo! - se unió a Microsoft en 2007 para oponerse a ECMAScript 4, lo que le llevó al 3,1 refuerzo de ECMAScript.

Mientras todo esto sucedía las comunidades de origen y desarrolladores abiertas se pusieron a trabajar para revolucionar lo que podría hacerse con JavaScript. Este esfuerzo de la comunidad se desató en 2005, cuando Jesse James Garrett publicó un libro blanco en el que acuñó el término Ajax, y describió un conjunto de tecnologías, de las cuales JavaScript era la columna vertebral, que se utiliza para crear aplicaciones web en las que los datos pueden ser cargados en el fondo, evitando la necesidad de cargar la página completa y como resultado: aplicaciones más dinámicas. Esto dio lugar a un periodo de renacimiento del uso de JavaScript encabezada por bibliotecas de código abierto y las comunidades que se formaron a su alrededor, con las bibliotecas como Prototype, JQuery, Dojo, Mootools y otros.

En julio de 2008 las partes dispares de ambos lados se reunieron en Oslo. Esto llevó a la eventual acuerdo a principios de 2009 para cambiar el nombre de ECMAScript 3.1 a ECMAScript 5 e impulsar el lenguaje.

Todo esto entonces nos trae a la actualidad, con JavaScript, que ha entrado en un nuevo y emocionante ciclo de evolución, la innovación y la normalización, con nuevos desarrollos como la plataforma nodejs, que nos permite utilizar JavaScript en el lado del servidor, y APIs de HTML5 para controlar los medios de comunicación de los usuarios, se abren sockets web para la comunicación, obtener datos sobre las características de ubicación y disposición geográfica, y más.

En el año 2014 y principios del 2015 se considera HTML5 y el nuevo ECMAScript 5 como un estándar para el desarrollo de aplicaciones web.

Es un momento emocionante para aprender JavaScript.

Pero, ¿qué es JavaScript?

Como ya hemos comentado en capítulos anteriores una página web dinámica es aquella que incorpora efectos como texto que aparece y desaparece, animaciones, acciones que se activan al pulsar botones y ventanas con mensajes de aviso al usuario.

Básicamente, JavaScript es un lenguaje de programación interpretado, por lo que no es necesario hacer nada con estos programas ni tan siquiera compilarlos. Los programas escritos con JavaScript se pueden probar directamente en cualquier navegador sin necesidad de procesos intermedios.

Más adelante, como ya veremos, JavaScript ha traspasado el ambiente de los navegadores y ha llegado incluso al servidor.

Incluir scripts en páginas y archivos .js

La integración entre JavaScript y HTML es muy fácil y variada, ya que hay al menos tres maneras para incluir código JavaScript en las páginas web.

Escribir JavaScript en el propio documento HTML

Las sentencias JavaScript se encierran entre etiquetas, o tags, `<script>` y se incluyen en cualquier parte del documento.

Aunque es correcto incluir cualquier bloque de sentencias en cualquier zona del documento, se recomienda definir el bloque de código JavaScript dentro de la cabecera del documento (dentro de la etiqueta `<head>`)

```
<!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>Ejemplo de código JavaScript en el propio documento</title>
<script type="text/javascript">
  alert("Un mensaje de prueba");
</script>
</head>

<body>
<p>Un párrafo de texto.</p>
</body>
</html>
```

Ilustración 2: Escribir JavaScript en el propio documento HTML

Para que la página HTML obtenida sea correcta, es imprescindible agregar el atributo `type` a la etiqueta `<script>`. Los valores a los que se iguala el atributo `type` están estandarizados y para el caso de JavaScript, el valor correcto es `text/javascript`.

Este método se emplea cuando se define un bloque pequeño de sentencias o cuando se quieren incluir códigos específicos en un determinado documento.

La principal desventaja es el no reutilizamiento ya que si se quiere hacer una modificación en el bloque de código, es necesario modificar todas las páginas que incluyen ese mismo bloque de código JavaScript (que habremos copiado en cada una de las respectivas páginas), por lo que no lo hace reutilizable.

Definir JavaScript en un archivo externo a nuestro documento

Las sentencias JavaScript se pueden escribir en un archivo externo de tipo JavaScript (.js) que los documentos HTML enlazan mediante la etiqueta (o tag) `<script>`.

Se pueden enlazar todos los archivos JavaScript que se necesiten y cada documento HTML puede enlazar tantos ficheros JavaScript como utilice.

Archivo `codigo.js`

```
alert("Un mensaje de prueba");
```

Documento XHTML

```
<!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>Ejemplo de código JavaScript en el propio documento</title>
<script type="text/javascript" src="/js/codigo.js"></script>
</head>

<body>
<p>Un párrafo de texto.</p>
</body>
</html>
```

Ilustración 3: Definir JavaScript en un archivo externo a nuestro documento

Aparte del atributo `type`, este método hace imprescindible definir el atributo `src`, que es el que indica la URL (o dirección) correspondiente al fichero JavaScript que se quiere unir.

Cada etiqueta `<script>` solamente puede unir un único fichero, pero en un mismo documento se pueden incluir tantas etiquetas (o tags) `<script>` como sean necesarias, lo que hace estos archivos reutilizables.

Los ficheros de tipo JavaScript son archivos normales de texto con la extensión `.js`, que se pueden crear con cualquier editor de texto.

La mayor ventaja de enlazar un fichero JavaScript externo es que se simplifica el código HTML del documento, que se puede reutilizar el mismo código JavaScript en todos los documentos del sitio web y que cualquier modificación

realizada en el fichero JavaScript se ve reflejada inmediatamente en todas las páginas HTML en los que está importado.

Incluir JavaScript dentro de las propias etiquetas HTML

Esta última forma es la menos utilizada, ya que consiste en crear bloques de JavaScript dentro del código HTML del documento.

```
<!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>Ejemplo de código JavaScript en el propio documento</title>
</head>

<body>
<p onclick="alert('Un mensaje de prueba')">Un párrafo de texto.</p>
</body>
</html>
```

Ilustración 4: Incluir JavaScript dentro de las propias etiquetas HTML

La mayor desventaja de esta forma es que ensucia innecesariamente el código HTML del documento y complica el mantenimiento del bloque de código en JavaScript.

Sintaxis de lenguaje

Definición de sintaxis: La sintaxis de un lenguaje de programación es el conjunto de reglas y parámetros que deben seguirse al redactar el código de los programas para que pasen a ser considerados como correctos y aceptables para ese lenguaje.

La sintaxis de JavaScript es muy similar a la de otros lenguajes:

- No importan las nuevas líneas y los espacios en blanco al igual que sucede en HTML ya que el propio intérprete del lenguaje los ignora.
- Es case sensitive: Distingue entre las mayúsculas y minúsculas
- En contra de otros lenguajes de programación no se definen el tipo de las variables (var). En JavaScript nunca sabemos el tipo de datos que va a contener una variable por lo que una misma variable puede almacenar diferentes tipos de datos.
- Cada sentencia en JavaScript acaba con el carácter ; (punto y coma)
Aunque no es necesario ya que el intérprete lee cada sentencia aunque no exista este carácter. Por convenio deberíamos incluirlo.

- Existe la opción de incluir comentarios para añadir información en el código fuente del programa. Estos comentarios suelen servir para dar información al propietario del código u otro desarrollador sobre el contenido del bloque de código en JavaScript. Los comentarios pueden ser de una sola línea o de varias líneas (en bloque).

Ejemplo de comentario de una sola línea:

```
// a continuación se muestra un mensaje  
alert("mensaje de prueba");
```

Los comentarios de una sola línea se definen añadiendo dos barras oblicuas (`//`) al principio de la línea.

Ejemplo de comentario de varias líneas:

```
/* Los comentarios de varias líneas son muy útiles  
cuando se necesita incluir bastante información  
en los comentarios */  
alert("mensaje de prueba");
```

Los comentarios multilínea se definen encerrando el texto del comentario entre los símbolos `/*` y `*/`.

Pregunta de refuerzo 1

Variables

Las variables de los lenguajes de desarrollo siguen una lógica similar a las variables utilizadas en otras ciencias como las físicas o las matemáticas. Una variable es un contenedor que se usa para almacenar y hacer referencia a otro valor.

De la misma manera que si en Física no existieran las variables no se podrían definir las fórmulas, en los lenguajes de programación no se podrían redactar códigos útiles sin las variables.

Sin las variables sería imposible escribir y crear "programas genéricos", es decir, códigos que funcionan de la misma manera independientemente de los valores concretos usados.

Si no se usaran variables, un código que suma dos números podría redactarse como:

`res = 4 + 2`

El código anterior es inútil ya que sólo sirve para el caso en el que el primer número de la suma sea igual a 4 y el segundo número sea igual a 2. En otras opciones, el código obtiene un *res* incorrecto.

Por otro lado, el código se puede reescribir de la siguiente forma usando variables para almacenar y referenciarse a cada número:

`num_1 = 4`

`num_2 = 2`

`res = num_1 + num_2`

Los elementos *num_1* y *num_2* son variables que retienen los valores que utiliza el código. El *res* se halla siempre en función del valor retenido por las variables, por lo cual, este código funciona de manera correcta para cualquier par de números que indiquemos. Si se varía el valor de las variables *num_1* y *num_2*, el código sigue trabajando correctamente.

Las variables en JavaScript se utilizan mediante la palabra reservada 'var'.

`var num_1 = 4;`

`var num_2 = 2;`

`var res = num_1 + num_2;`

La palabra 'var' solamente se indica al definir por primera vez la variable, y a eso lo llamamos 'declarar' una variable. Cuando se declara una variable no hace falta declarar también el tipo de dato que va a almacenar esa variable.

Si en el momento de declarar una variable se le otorga también un valor, se dice que la variable ha sido inicializada. En JavaScript no es obligatorio inicializar las variables, por lo que pueden ser inicializadas posteriormente.

```
var num_1;  
var num_2;  
num_1 = 4;  
num_2 = 2;  
var res = num_1 + num_2;
```

Podemos utilizar una variable no declarada en cualquier sentencia de código. Esta es una de las habilidades más sorprendentes de JavaScript y que muchos otros lenguajes de programación no tienen. JavaScript creará una variable global para esta variable no declarada y la asigna el valor que le corresponda por el código.

```
num_1 = 4;  
num_2 = 2;  
res = num_1 + num_2;
```

De cualquier otra forma, es recomendable declarar todas las variables que se vayan a usar.

El nombre de una variable también se le conoce como identificador y debe cumplir la siguiente normativa:

- El identificador únicamente puede estar formado por números, letras, y los símbolos '\$' y '_' a lo sumo.
- El primer carácter del identificador no debe ser un número.

Pregunta de refuerzo 2

Tipos de datos

Ya sabemos que todas las variables en JavaScript se crean a través de la palabra reservada "var" pero dependiendo de los valores que almacenen pueden ser de un tipo u otro. En otras palabras, el valor que almacena la variable otorga tipo a esa misma variable.

Numéricos

Se usan para contener valores numéricos enteros (llamados *integer*) o decimales (llamados *float*).

De esta manera, el valor que se asigna a la variable se realiza indicando directamente el número entero o decimal. Los números decimales utilizan el carácter '.' en vez de ',' para realizar la separación entre la parte entera y la parte decimal:

```
var entero = 99;      // variable tipo entero  
var decimal = 9384.23; // variable tipo decimal
```

Cadenas de texto

Se usan para contener caracteres, palabras y/o frases de texto. Para darle el valor a la variable, se encierran los valores entre comillas dobles o simples, que delimitan el inicio y el final de la frase, caracteres o palabras:

```
var sms = "Welcome to our city!";  
var nomProducto = 'Escoba';  
var letter = 'e';
```

A veces el texto que contienen estas variables no es tan fácil. Si el propio texto encerrado entre comillas dobles o simples tiene comillas dobles o simples que deben aparecer como parte del valor se haría así:

```
/* Comillas simples dentro de comillas dobles*/  
var text1 = "Una frase con 'comillas simples' dentro";  
/* Comillas dobles dentro de comillas simples*/  
var text2 = 'Una frase con "comillas dobles" dentro';
```

A veces, hacen falta caracteres especiales para definir un cambio de línea dentro del texto de nuestra variable, o incluso, quizás queramos meter comillas simples y dobles a la vez dentro de nuestra sentencia.

Si se quiere incluir...	Se debe incluir...
Una nueva línea	\n
Un tabulador	\t
Una comilla simple	\'
Una comilla doble	\\"
Una barra inclinada	\\"

De esta manera, podemos rehacer el ejemplo anterior:

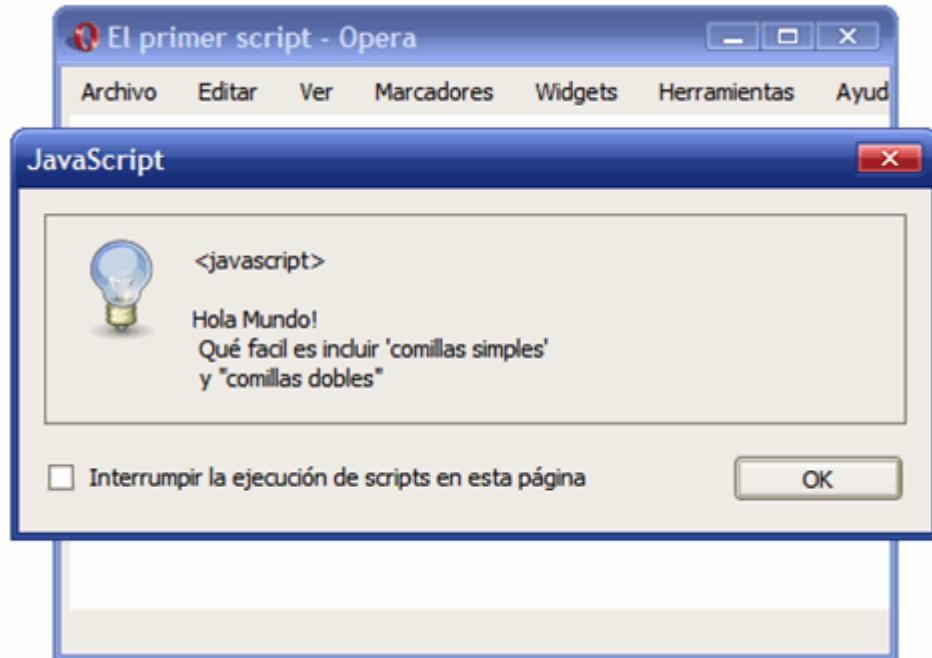
```
var text1 = 'Una frase con \'comillas simples\' dentro';
```

```
var text2 = "Una frase con \"comillas dobles\" dentro";
```

Este mecanismo de JavaScript se denomina "mecanismo de escape" o "caracteres de escape".

Ejercicio propuesto

- 1.- Queremos sacar una cadena de texto por pantalla tal y como se ve en la imagen. Esa cadena de texto se encuentra guardada en una variable con identificador "mensaje". Otorga el valor a esa variable.



Solución

```
var mensaje = "Hola Mundo! \n Qué facil es incluir \'comillas simples\' \n y\n \"comillas dobles\" ";
```

Arrays

También llamados vectores o matrices. Sin embargo, la denominación 'array' es la más utilizada y es un término comúnmente aceptado en el entorno de desarrollo.

Un array es una colección de variables, sin importar los tipos de los que sean cada una. Los arrays sirven para guardar colecciones de valores, de manera que serviría para agrupar diferentes variables. Por ejemplo tenemos esta sucesión de variables con los días de la semana:

```
var dia1 = "Lunes";  
var dia2 = "Martes";  
...  
var dia7 = "Domingo";
```

El código anterior es correcto aparte de poco eficiente y que complica el desarrollo de nuestra programación.

Variando el ejemplo anterior podemos disponerlo de la siguiente forma:

```
var diasSemana =  
["Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado", "Domingo"];
```

Hemos definido una única variable (diasSemana) que agrupa un conjunto de valores de variables (los propios días de la semana, que son cadenas). Por tanto la manera de crear o escribir un array es con el contenido entre corchetes ([]) y separando cada valor de nuestra variable por una coma (no se pone una coma en el elemento final de nuestro array):

```
var nombre_array = [valor1, valor2, ..., valorN];
```

Ya que tenemos definido nuestro primer array, podemos sacar algunos de sus valores de su interior de esta manera:

```
var diaSeleccionado = diasSemana [0]; // diaSeleccionado = "Lunes"  
var otroDia = diasSemana [5]; // otroDia = "Sábado"
```

La posición 0 del array sería el primer elemento del array (sea cual fuere) y el elemento del array que estaría en la sexta posición sería el elemento 5. Esto sucede porque los índices o posiciones dentro de un array empiezan con el elemento 0 y de ahí en adelante.

Finalmente debemos advertir que los array tampoco están *tipados* (no se les asigna un tipo) por lo que podemos meter elementos de cualquier tipo dentro del mismo array.

Ejercicio propuesto

Crear una única variable de identificador "meses" y almacenar en ella todos los meses del año. Guardar en dos variables con nombres "primerMes" y "ultimoMes" el primer y el último mes del año.

Respuesta

```
var meses = ["Enero", "Febrero", "Marzo", "Abril", "Mayo", "Junio", "Julio",
"Agosto", "Septiembre", "Octubre", "Noviembre", "Diciembre"];
```

```
var primerMes = meses[0];
```

```
var ultimoMes = meses[11];
```

Booleanos

Las variables de tipo booleano también son llamadas o denominadas con el nombre de variables de tipo lógico. Estas variables suelen servir para condiciones o para la programación lógica.

Una variable este tipo solo puede almacenar dos valores: true (verdadero) o false (falso).

```
var register = false;
```

```
var mayorEdad = true;
```

Operadores

Los operadores manipulan los valores de las variables, realizan cálculos matemáticos y comparan los valores de diferentes variables.

Asignación

Este operador es el principal y el más sencillo. Sirve para asignar un valor a una variable.

```
var num1 = 3;
```

Incremento y decremento

Ambos operadores sirven para decrementar o incrementar el valor de una variable.

```
var num = 5;  
++num;  
alert(num); // num = 6
```

El primer operador (el de incremento) se indica mediante el prefijo `++` y aumenta en una unidad el valor de la variable. El segundo operador (el de decremento) se indica mediante el prefijo `--` y disminuye en una unidad el valor de una variable.

```
var num = 5;  
--num;  
alert(num); // num = 4
```

Estos operadores también pueden ser indicados como sufijos en vez de prefijos:

```
var num = 5;  
num++;  
alert(num); // num = 6
```

Aunque el resultado en uno y otro caso es el mismo, no es lo mismo poner el operador como prefijo o sufijo:

```
var num1 = 5;  
var num2 = 2;  
num3 = num1++ + num2;  
// num3 = 7, num1 = 6
```

```
var num1 = 5;  
var num2 = 2;  
num3 = ++num1 + num2;  
// num3 = 8, num1 = 6
```

Cuando el operador `++` está como prefijo  Su valor se incrementa antes de la operación.

Cuando el operador `++` está como sufijo  Su valor se incrementa después de la operación.

Lógicos

Los operadores lógicos son adecuados para realizar condiciones y lógica matemática.

El resultado de estas operaciones siempre da como resultado un valor lógico o de booleano.

Negación

El operador de negación se utiliza para dar el valor contrario a una variable. En el caso de una variable que tenga el valor de un booleano, se le asignará el valor contrario tras usarse el operador de negación.

```
var vis = true;
```

```
alert(!vis); // Muestra "false" y no "true"
```

El operador de negación también se puede usar cuando el valor de la variable es un texto o un número. Si es un número y su valor es 0 se transforma en false y si es otro número diferente de 0 su valor es true, por lo que al negarlo en los dos casos sería su valor contrario. En el caso de las cadenas, si es cadena vacía ("") se transforma en false y con cualquier otra cadena su valor sería true.

```
var cant = 0;
```

```
vac = !cant; // vac = true
```

```
cant = 2;
```

```
vac = !cant; // vac = false
```

```
var mens = "";
```

```
mensVac = !mens; // mensVac = true
```

```
mens = "Bienvenido";
```

```
mensVac = !mens; // mensVac = false
```

AND

Este operador sirve para combinar los valores de dos variables, usando lógica matemática y solo dando true si ambos valores son true. En otro caso el valor final es false. El operador se define mediante el símbolo "`&&`".

OR

Este operador sirve para combinar los valores de dos variables, usando lógica matemática y solo dando true si alguno de los valores son true. En otro caso el valor final es false. El operador se define mediante el símbolo "||"

Matemáticos

Los operadores declarados son: suma (+), resta (-), multiplicación (*) y división (/). Estos operadores son todos matemáticos.

```
var num1 = 10;  
var num2 = 5;  
  
res = num1 / num2; // res = 2  
  
res = 3 + num1;    // res = 13  
  
res = num2 - 4;    // res = 1  
  
res = num1 * num2; // res = 50
```

Aparte de estos anteriormente comentados, existe el operador de "módulo" que obtiene como valor el resto de una división.

Este operador se indica mediante el símbolo "%".

```
var num1 = 10;  
var num2 = 5;  
  
res = num1 % num2; // res = 0  
  
num1 = 9;  
  
num2 = 5;  
  
res = num1 % num2; // res = 4
```

Relacionales

Los relacionales: mayor que (>), menor que (<), mayor o igual (>=), menor o igual (<=), igual que (==) y distinto de (!=).

El resultado de ellos siempre es un valor de booleano.

```
var num1 = 3;  
var num2 = 5;  
  
res = num1 > num2; // res = false  
  
res = num1 < num2; // res = true
```

```
num1 = 5;  
num2 = 5;  
res = num1 >= num2; // res = true  
res = num1 <= num2; // res = true  
res = num1 == num2; // res = true  
res = num1 != num2; // res = false
```

Ejercicio propuesto

Teniendo este array:

```
var valores = [true, 5, false, "hola", "adios", 2];
```

1. Entre los dos elementos de texto, determinar cuál es mayor.
2. Usando los dos elementos booleanos, utilizar un operador para dar un resultado de true y otro de false.
3. Resolver cinco operaciones diferentes con los dos elementos numéricos.

Respuesta

```
var valores = [true, 5, false, "hola", "adios", 2];  
// Cual de los 2 elementos de texto es mayor.Si el resultado es true, el primer texto es mayor  
var resultado = valores[3] > valores[4]; //resultado determina el mayor
```

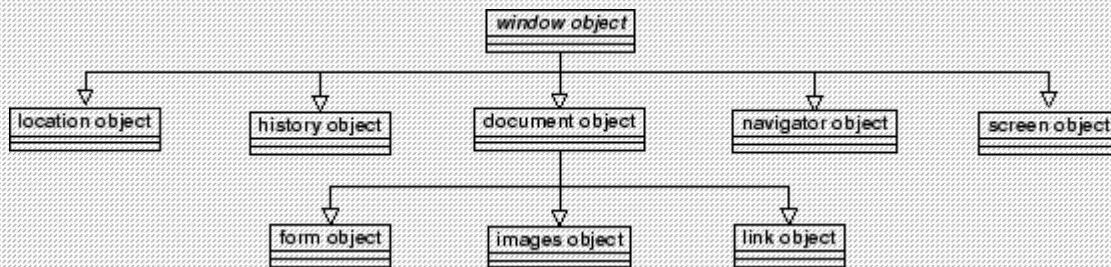
```
// Combinar valores booleanos  
var valor1 = valores[0]; var valor2 = valores[2];  
// Obtener un resultado TRUE  
var resultado = valor1 || valor2; //resultado guarda true  
// Obtener un resultado FALSE  
resultado = valor1 && valor2; //resultado guarda false
```

```
// Operaciones matemáticas  
var num1 = valores[1]; var num2 = valores[5];  
var suma = num1 + num2;  
var resta = num1 - num2;  
var multiplicacion = num1 * num2;  
var division = num1 / num2;  
var modulo = num1 % num2;
```

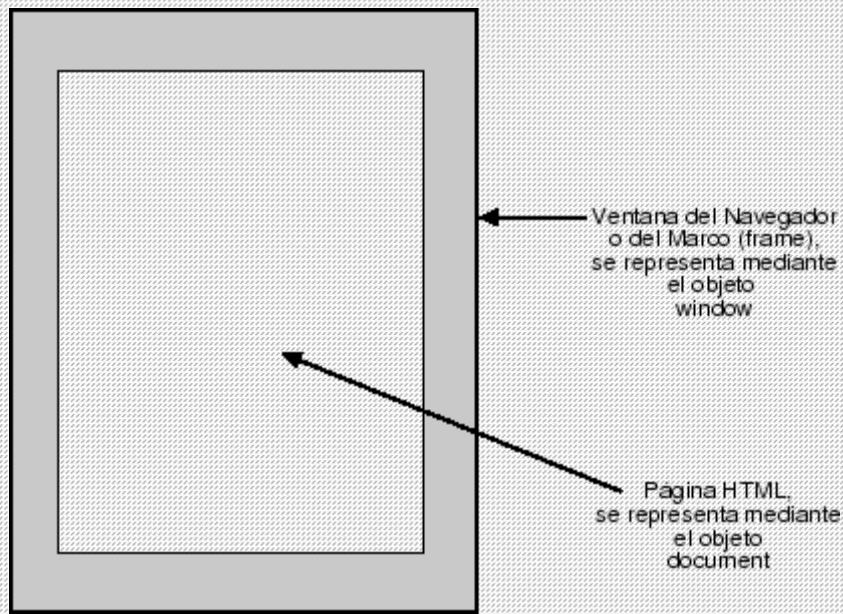
Objetos del navegador

Cuando se carga una página en un navegador se crean un número de objetos característicos del navegador según el contenido de la página.

La siguiente figura muestra la jerarquía de clases del Modelo de Objetos del Documento (Document Object Model).



El objeto **window** es el de más alto nivel, contiene las propiedades de la ventana y en el supuesto de trabajar con marcos (frames), se genera un objeto **window** para cada uno. El objeto **document** contiene todas las propiedades del documento actual, como son: su color de fondo, enlaces, imágenes, etc.



El objeto **navigator** contiene las propiedades del navegador. El objeto **location** contiene las propiedades de la URL activa. El objeto **history** contiene las propiedades que representan a las URL que el usuario ha visitado

anteriormente. Es como una caché. El objeto screen contiene información referente a la resolución de la pantalla que muestra la URL.

El objeto window

Contiene las propiedades básicas de la ventana y sus componentes. Algunas de los datos más elementales son:

- *defaultStatus* contiene el mensaje que aparece en la barra de estado)
- *frames* es una matriz que representa todos los frames de la ventana
- *length* contiene el número de frames de la ventana
- *name* contiene el nombre de la ventana
- *self* hace referencia a la ventana activa

El siguiente ejemplo muestra cómo modificar el mensaje que aparece en la barra de estado del navegador.

```
<html>
<head><title> Ejemplo del Objeto window</title> </head>
<body>
<script language=JavaScript>
window.defaultStatus = "Hola ;-), este mensaje aparece en la barra de estado";
</script>
</body>
</html>
```

El objeto window también posee una serie de métodos que permiten ejecutar funciones específicas con las ventanas, como por ejemplo, crear ventanas y cuadros de diálogo. También es posible determinar el aspecto que tendrá la nueva ventana del navegador mediante los campos de datos que permiten configurar el menú, la barra de herramientas, la barra de estado, etc. El siguiente ejemplo muestra cómo abrir una nueva ventana desde la ventana actual.

```
<html>
<head>
<title>Ejemplo de creación de ventana</title>
<script language="JavaScript">
function AbrirVentana() {
```

```
ventana=open("", "nueva", "toolbar=no,directories=no,menubar=no,width=180,height=180");

ventana.document.write("<HEAD><TITLE>Nueva Ventana</TITLE></HEAD><BODY>");

ventana.document.write("<FONT SIZE=4 COLOR=red>Nueva Ventana</FONT><BR><BR><BR>>");

ventana.document.write("<FORM><INPUT TYPE='button' VALUE='Cerrar' onClick='self.close()'></FORM>");

}

</script>

</head>

<body>

<form>

<input type="button" value="Abrir una ventana" onClick="AbrirVentana();">

<br>

</form>

</body>

</html>
```

El objeto location

El objeto location contiene toda la información sobre la URL que se está visualizando, así como todos los detalles de esa dirección (puerto, protocolo, etc.).

El objeto screen

Permite obtener información sobre la resolución de la pantalla. En el siguiente ejemplo, se establece el color de fondo de la página de acuerdo a la resolución que soporte la pantalla del usuario.

```
<html>

<head><title> Ejemplo del Objeto screen</title> </head>

<body>

<script language=JavaScript>
```

```
switch (window.screen.colorDepth)
{
    case 1: case 4:
        document.bgColor = "white";
        break;
    case 8: case 15: case 16:
        document.bgColor = "blue";
        break;
    case 24: case 32:
        document.bgColor = "skyblue";
        break;
    default:
        document.bgColor = "white";
}
document.write("Su pantalla soporta color de " + window.screen.colorDepth +
bit");
</script>
</body>
</html>
```

El objeto document - La página en sí

El objeto document hace referencia a determinadas características de la página, como son su color de fondo (bgColor), el color de sus enlaces, etc.

El código que se muestra a continuación carga una imagen dependiendo de la elección que haga el usuario.

```
<html>
<head><title> Ejemplo del Objeto document</title>
<!-- Se muestra un número diferente de imágenes dependiendo
-- del valor que introduzca el usuario
-- dato: src
```

```
-->

</head>

<body>

<IMG NAME=img1 SRC="" BORDER=0 WIDTH=200 HEIGHT=150>

<script language=JavaScript>

var myImages = new Array("usa.gif", "canada.gif", "jamaica.gif", "mexico.gif");

var imgIndex = prompt("Enter a number from 0 to 3","");
document.images["img1"].src = myImages[imgIndex];

</script>

</body>

</html>
```

A continuación vemos un ejemplo que permite conectar código a los eventos de la página web. El primero de ellos simplemente muestra una ventana de alerta, mientras que el segundo va modificando de forma aleatoria la imagen que se carga.

```
<html>

<head><title> Ejemplo de Eventos</title>

</head>

<body>

<script language=JavaScript>

function linkSomePage_onclick() {

    alert('Este enlace no lleva a ninguna parte');

    return false;
}

</script>

<A HREF="somepage.htm" NAME="linkSomePage">

    Pincha Aquí

</A>

<script language=JavaScript>
```

```
window.document.links[0].onclick = linkSomePage_onclick;  
</script>  
</body>  
</html>  
  
<html>  
<head><title> Ejemplo del Objeto document</title>  
<!-- Se carga una imagen aleatoria  
-->  
<script language=JavaScript>  
var myImages = new Array("usa.gif","canada.gif","jamaica.gif","mexico.gif");  
function changeImg(imgNumber) {  
    var imgClicked = document.images[imgNumber];  
    var newImgNumber = Math.round(Math.random() * 3);  
    while (imgClicked.src.indexOf(myImages[newImgNumber]) != -1) {  
        newImgNumber = Math.round(Math.random() * 3);  
    }  
    imgClicked.src = myImages[newImgNumber];  
    return false;  
}  
</script>  
</head>  
<body>  
<A HREF="" NAME="linkImg1" onclick="return changeImg(0)">  
    <IMG NAME=img1 SRC="usa.gif" BORDER=0 >  
</A>  
<A HREF="" NAME="linkImg2" onclick="return changeImg(1)">  
    <IMG NAME=img1 SRC="mexico.gif" BORDER=0 >  
</A>
```

```
</body>
```

```
</html>
```

El objeto history

El objeto history contiene información sobre los enlaces que el usuario ha visitado. Se utiliza principalmente para generar botones de avance y retroceso.

El objeto navigator

El objeto navigator permite obtener información del navegador con el que se está visualizando el documento. El siguiente código JavaScript detecta el navegador que se está utilizando y abre la página específica del mismo.

```
<html>
<head><title> Ejemplo del Objeto navigator</title>
<!-- Se detecta el navegador con el que se ha abierto la página
-->
<script language=JavaScript>
<!--
function getBrowserName() {
    var lsBrowser = navigator.appName;
    if (lsBrowser.indexOf("Microsoft") >= 0) {
        lsBrowser = "MSIE";
    }
    else if (lsBrowser.indexOf("Netscape") >= 0) {
        lsBrowser = "NETSCAPE";
    }
    else {
        lsBrowser = "UNKNOWN";
    }
    return lsBrowser;
}
function getOS() {
    var userPlat = "unknown";
```

```
var navInfo = navigator.userAgent;

if ((navInfo.indexOf("windows NT") != -1)
    || (navInfo.indexOf("windows 95") != -1 )
    || (navInfo.indexOf("windows 98") != -1 )
    || (navInfo.indexOf("WinNT") != -1 )
    || (navInfo.indexOf("Win95") != -1 )
    || (navInfo.indexOf("Win98") != -1 )) {
    userPlat = "Win32";
}

else if(navInfo.indexOf("Win16") != -1) {
    userPlat = "Win16";
}

else if(navInfo.indexOf("Macintosh") != -1) {
    userPlat = "PPC";
}

else if(navInfo.indexOf("68K") != -1) {
    userPlat = "68K";
}

return userPlat;
}

function getBrowserVersion() {
    var findIndex;
    var browserVersion = 0;
    var browser = getBrowserName();
    if (browser == "MSIE") {
        browserVersion = navigator.userAgent;
        findIndex = browserVersion.indexOf(browser) + 5;
        browserVersion = parseInt(browserVersion.substring(findIndex,findIndex + 1));
    }
}
```

```
else {
    browserVersion = parseInt(navigator.appVersion.substring(0,1));
}
return browserVersion;
}

-->

</script>
</head>
<body>
<script language=JavaScript>
<!--
var userOS = getOS();
var browserName = getBrowserName();
var browserVersion = getBrowserVersion();
if (browserVersion < 4 || browserName == "UNKNOWN" || userOS == "Win16") {
    document.write("<H2>Sorry this browser version is not supported</H2>")
}
else if (browserName == "NETSCAPE") {
    location.replace("NetscapePage.html");
}
else {
    location.replace("MSIEPage.html");
}
-->
</script>
<noscript>
<H2>Esta página requiere un navegador que soporte JavaScript</H2>
</noscript>
</body></html>
```

Pregunta de refuerzo 3

Control de flujo

Los códigos que se pueden escribir usando solo variables y operadores, son una sucesión de instrucciones básicas.

Hay programas complejos como recorrer un array o establecer una condición que no pueden ser realizadas simplemente con una sucesión de instrucciones básicas, es por ello que necesitamos instrucciones de control de flujo que nos pueden elegir líneas para ejecutar dentro de nuestro código o repetir una serie de líneas un número de veces según una condición.

Estructura if

Es una estructura de condición, si se comprueba el valor true de esa condición se entra dentro del bloque de código encerrado entre {...}, si no lo cumple no entra y, por tanto, no ejecuta esas líneas.

if(condicion) {...}

Ejercicio propuesto

Completar las condiciones de los if para que se entre en los bloques:

var num1 = 5;

var num2 = 8;

if(...) { alert("num1 no es mayor que num2");}

if(...) {alert("num2 es positivo");}

if(...) {alert("num1 es negativo o distinto de cero");}

if(...) {alert("Incrementar en 1 el valor de num1 no lo hace mayor o igual que num2");}

Respuesta

if(num1 <= num2) {...}

if(num2 >= 0) {...}

if(num1 < 0 || num1 != 0) {...}

if(++num1 < num2) {... }

Estructura if...else

Muchas veces necesitamos ejecutar bloques diferentes dependiendo de una condición. Añadiendo la estructura anterior, se agrega un bloque "else" que permite ejecutarse en el caso que no se cumpla la condición del "if".

if(condicion) {...}

else {...}

A parte, se pueden poner otros bloques de código para que se ejecuten si la condición del "if" no se cumple y queremos comprobar que se satisface otra condición. Estos van con la estructura "else if".

if(condicion) {...}

else if(condicion2) {...}

else {...}

En este caso anterior el bloque "else" solo se ejecuta si no se ejecuta ningún bloque anterior. Se pueden poner tantos bloques "else if" como se quiera, teniendo en cuenta que solo se comprobará su condición si todos los bloques anteriores han dado false en sus respectivas condiciones. Por tanto, el orden en el que incluyamos los bloques "else if" es importante. El bloque "else", por ello, solo se ejecutará al final (y solo se incluirá al final), cuando todos los demás bloques no han satisfecho a su condición.

Estructura for

La estructura "for" permite usar repeticiones (denominadas bucles), para que reiteren líneas de código mientras se satisfaga una condición.

for(inicializacion; condicion; actualizacion) {...}

El funcionamiento de la estructura "for" es: repite las líneas del trozo de código, encerrado entre {...}, mientras se cumpla la condición, actualizando con cada repetición los valores cambiantes de la condición.

La "inicialización"  los valores iniciales de las variables que controlan la repetición.

La "condición"  decide si continua o se detiene la repetición.

La "actualización"  valor que se asigna después de cada repetición.

Ejercicio propuesto

El factorial de un entero n es una operación que consiste en multiplicar todos los factores:

$$n \times (n-1) \times (n-2) \times \dots \times 1$$

Por ejemplo, el factorial de 4 (escrito como 4!) es igual a:

$$4! = 4 \times 3 \times 2 \times 1 = 24$$

Con la estructura for, realice este funcionamiento.

Respuesta

```
var num = 4;  
var res = 1;  
for(var i=1; i<=num; i++) {  
    res *= i; //res guardará finalmente el resultado del factorial  
}
```

Estructura for...in

Es una variante de la estructura for y sirve para objetos y arrays dentro del lenguaje JavaScript.

```
for(indice in array) {...}
```

En este caso anterior, nos serviría para recorrer todos los elementos contenidos dentro del array.

```
var numbers =[0,1,2];  
for(i in numbers) {var a =numbers[i];}
```

Daríamos 3 iteraciones dentro de nuestro bucle, una por cada elemento de nuestro array. Esta estructura es la óptima para recorrer un array o un objeto en JavaScript ya que funciona sea cual sea el número de elementos de nuestro array.

Funciones

Para manejar nuestras diferentes variables JavaScript hace uso de funciones y propiedades, que ya se encuentran en el propio lenguaje. A continuación veremos las funciones según su utilidad.

Funciones útiles para cadenas de texto

- length, halla la longitud de una cadena de texto (el número de caracteres que la forman).

```
var men = "Hola Mundo";  
  
var numLetras = men.length; // numLetras = 10
```

- +, se emplea para concatenar varias cadenas de texto

```
var men1 = "Hola";  
  
var men2 = " Mundo";  
  
var men = men1 + men2; // men = "Hola Mundo"
```

- Aparte del operador +, tiene el mismo funcionamiento concat()

```
var men1 = "Hola";  
  
var men2 = men1.concat(" Mundo"); // men2 = "Hola Mundo"
```

Y también con variables numéricas:

```
var var1 = "Hola "; var var2 = 3;  
  
var men = var1 + var2; // men = "Hola 3"
```

- toUpperCase(), convierte los caracteres a mayúsculas.

```
var men1 = "Hola";  
  
var men2 = men1.toUpperCase(); // men2 = "HOLA"
```

- toLowerCase(), convierte los caracteres a minúsculas.

```
var men1 = "HoLA";  
  
var men2 = men1.toLowerCase(); // men2 = "hola"
```

- `charAt(posicion)`, halla el carácter de la posición.

```
var men = "Hola";
var l = men.charAt(0); // letra = H
l = men.charAt(3); // letra = a
```

- `indexOf(caracter)`, halla la posición en la que se encuentra el carácter indicado. Si no está devuelve -1, y si está varias veces su primera aparición.

```
var men = "Hola";
var pos = men.indexOf('a'); // pos = 3
pos = men.indexOf('z'); // pos = -1
```

- `lastIndexOf(caracter)`, halla la última posición en la que se encuentra el carácter. Si no está devuelve -1.

```
var men = "Hola";
var pos = men.lastIndexOf('a'); // pos = 3
pos = men.lastIndexOf('z'); // pos = -1
```

- `substring(inicio, final)`, saca un trozo de una cadena de texto. El parámetro 'final' no es obligatorio. Si no se pone corta la cadena de texto hasta el final del string.

```
var men = "Hola Mundo";
var por = men.substring(2); // por = "la Mundo"
por = men.substring(5); // por = "Mundo"
por = men.substring(1, 8); // por = "ola Mun"
por = men.substring(3, 4); // por = "a"
por = men.substring(7); // por = "ndo"
```

- `split(separador)`, divide la cadena de texto en diferentes trozos, definiendo un separador para dividir esa cadena, y mete las porciones dentro de un array.

```
var men = "Hello World, be a string!";
var p = men.split(" ");
// p = ["Hello ", " World, ", " be ", " a ", " string!"];
```

Funciones útiles para arrays

- `length`, halla el número de elementos dentro de un array.

```
var v = ["a", "e", "i", "o", "u"];
var num = v.length; // num = 5
```

- `concat()`, concatena los elementos de varios arrays.

```
var a1 = [1, 2, 3];
a2 = a1.concat(4, 5, 6); // a2 = [1, 2, 3, 4, 5, 6]
a3 = a1.concat([4, 5, 6]); // a3 = [1, 2, 3, 4, 5, 6]
```

- `join(separador)`, une los elementos de un array para formar una cadena de texto. Es lo contrario al "split". Se indica un separador para unir los elementos de la cadena.

```
var a = ["hola", "mundo"];
var men = a.join(""); // men = "holamundo"
men = a.join(" "); // men = "hola mundo"
```

- `pop()`, suprime el último elemento del array y lo mete en la variable seleccionada.

```
var a = [1, 2, 3];
var u = a.pop(); // ahora a = [1, 2], u = 3
```

- push(), agrega un elemento (o varios) a nuestro array.

```
var a = [1, 2, 3];  
a.push(4); // ahora a = [1, 2, 3, 4]
```

- shift(), suprime el primer elemento de nuestro array y lo mete en la variable seleccionada.

```
var a = [1, 2, 3];  
var p = a.shift(); // ahora a = [2, 3], p = 1
```

- unshift(), agrega un elemento (o varios) al principio de nuestro array.

```
var a = [1, 2, 3];  
a.unshift(0); // ahora a = [0, 1, 2, 3]
```

- reverse(), coloca los elementos de un array en su orden inverso.

```
var a = [1, 2, 3];  
a.reverse(); // ahora a = [3, 2, 1]
```

Funciones útiles para números

- NaN, (del inglés, "Not a Number") el lenguaje JavaScript devuelve esto si la variable con la que estamos trabajando o el resultado de una operación no es un número.

```
var num1 = 0; var num2 = 0;  
alert(num1/num2); // muestra el valor NaN
```

- isNaN(), protege nuestro código de valores no numéricos.

```
var num1 = 0; var num2 = 0;  
if(isNaN(num1/num2)) {...} else {...}
```

- Infinity, es el valor de infinito cuando las operaciones dan dicho resultado.

```
var num1 = 10; var num2 = 0;  
alert(num1/num2); // valor Infinity
```

- `toFixed(digitos)`, fija el número de decimales que tiene que resolver una operación y redondea si es necesario.

```
var num1 = 4564.34567;  
num1.toFixed(2); // 4564.35  
num1.toFixed(6); // 4564.345670  
num1.toFixed(); // 4564
```

El ámbito de las variables

El ámbito de una variable ("scope"): ubicación en el código en la que se define la variable. Hay dos tipos de ámbitos: global y local.

```
function crea() { var m = "Mensaje";}  
crea(); alert(m);
```

Este mensaje no muestra ningún mensaje ya que la variable ha sido inicializada dentro de una función por lo que se dice que es una variable local de esa función.

```
function crea() {var m = "Mensaje";alert(m);}  
crea();
```

En este caso anterior el mensaje si se muestra ya que el "alert" ha sido ejecutado dentro de la misma función y tiene por tanto acceso a esa variable local.

Ahora vamos a hablar de las variables globales:

```
var m = "Mensaje";  
function muestra() {alert(m);}
```

La variable "m" la hemos definido fuera de cualquier función por lo que se convierte en una variable global inicializada en el momento en el que se ejecuta dicha sentencia. Por tanto, todas las funciones tienen acceso a esa variable.

Un problema habitual es llamar a una variable global y otra local de la misma manera. Si esto sucede, dentro de la función donde esté definida esa local, la local es la que toma prevalencia frente a la global.

```
var m = "global";  
function muestra() {var m = "local"; alert(mensaje);}  
alert(m);muestra();alert(m);
```

Si ejecutamos dicho código obtenemos:

global local global

Ejercicio Propuesto

Codifique un javascript que llame en un alert a una función y que compruebe si el texto de ese alert está en mayúsculas o minúsculas.

Respuesta

```
<!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>Ejercicio </title>
<script type="text/javascript">
function info(cadena) {
    var resultado = "La cadena "+" "+cadena+" ";
    // Comprobar mayúsculas y minúsculas
    if(cadena == cadena.toUpperCase()) {
        resultado += " está formada sólo por mayúsculas";
    }
    else if(cadena == cadena.toLowerCase()) {
        resultado += " está formada sólo por minúsculas";
    }
    else {
        resultado += " está formada por mayúsculas y minúsculas";
    }
    return resultado;
}
alert(info("OVNI = OBJETO VOLADOR NO IDENTIFICADO"));
alert(info("En un lugar de la mancha..."));
</script>
</head>
<body>
</body>
</html>
```

Eventos y su manejo mediante funciones

Hasta el momento, todos los códigos que hemos visto siguen ejecutándose sentencia a sentencia, sin interactuar con el usuario.

Estos códigos son poco útiles en programas de JavaScript normales ya que se espera y desea una interacción con los clientes que estén usando esa página web. Así, clickar con el botón del ratón en ciertos apartados, mover el mismo ratón, o teclear pueden ser eventos que se produzcan dentro de nuestra aplicación y se traduzcan en funciones que se tienen que ejecutar en ciertos momentos en JavaScript.

El propio lenguaje nos aporta una serie de eventos para medir la interacción del usuario con nuestra aplicación. Estas funciones creadas en JavaScript son denominadas "manejadores de eventos" o "event handlers".

Hay tres tipos de manejadores de eventos:

- Como atributos de las etiquetas HTML.
- Como funciones en código JavaScript externo.
- "Semánticos".

Evento	Descripción	Elementos para los que está definido
onblur	Deseleccionar el elemento	<button>, <input>, <label>, <select>, <textarea>, <body>
onchange	Deseleccionar un elemento que se ha modificado	<input>, <select>, <textarea>
onclick	Pinchar y soltar el ratón	Todos los elementos
ondblclick	Pinchar dos veces seguidas con el ratón	Todos los elementos
onfocus	Seleccionar un elemento	<button>, <input>, <label>, <select>, <textarea>, <body>
onkeydown	Pulsar una tecla (sin soltar)	Elementos de formulario y <body>
onkeypress	Pulsar una tecla	Elementos de formulario y <body>
onkeyup	Soltar una tecla pulsada	Elementos de formulario y <body>
onload	La página se ha cargado completamente	<body>
onmousedown	Pulsar (sin soltar) un botón del ratón	Todos los elementos
onmousemove	Mover el ratón	Todos los elementos

Evento	Descripción	Elementos para los que está definido
onmouseout	El ratón "sale" del elemento (pasa por encima de otro elemento)	Todos los elementos
onmouseover	El ratón "entra" en el elemento (pasa por encima del elemento)	Todos los elementos
onmouseup	Soltar el botón que estaba pulsado en el ratón	Todos los elementos
onreset	Inicializar el formulario (borrar todos sus datos)	<form>
onresize	Se ha modificado el tamaño de la ventana del navegador	<body>
onselect	Seleccionar un texto	<input>, <textarea>
onsubmit	Enviar el formulario	<form>
onunload	Se abandona la página (por ejemplo al cerrar el navegador)	<body>

Pregunta de refuerzo 4

Manejadores de eventos como atributos HTML

Se incluye en un atributo del propio elemento HTML.

```
<input type="button" value="Pinchame" onclick="alert('Gracias');"/>
```

En este ejemplo, cuando nuestro cliente pinche en el botón se ejecutará todo el contenido que hay dentro del atributo “onclick” que es nuestro manejador de eventos.

Este método es quizás el menos práctico ya que impide la reutilización del código JavaScript (que solo se encuentra asignado a ese atributo).

También se pueden poner varios manejadores en una misma etiqueta, pero en diferentes atributos que son los manejadores, claro.

```
<div onclick="alert('Has pulsado');"  
onmouseover="alert('Has pasado con el ratón');">
```

Manejadores de eventos y variable this

La variable “this” es especial en JavaScript. Se usa para que tome como valor el invocador del evento dentro del manejador. Es decir, qué componente nos ha invocado a la función.

```
<div id="contenido" style="width:180px; height:80px; border:thin solid silver"  
onmouseover="this.style.borderColor='green';"  
onmouseout="this.style.borderColor='red';">  
  
    Sección de contenidos...</div>
```

En este código, los manejadores de eventos “onmouseout” y el “onmouseover” (que cambian el color del borde de nuestro elemento) utilizan la variable “this” para hacer referencia al elemento que nos ha invocado al evento, este es, el div con id “contenido”. De esta manera, aunque cambie el id del elemento nuestro código seguirá en funcionamiento.

Manejadores de eventos como funciones externas

Es mucho mejor para nuestro código y para el reutilizamiento externalizar todas nuestras sentencias JavaScript en funciones externas.

De ese modo, llamamos en nuestros manejadores de eventos a estas funciones externas.

```
function muestra() {alert('Gracias');}  
  
<input type="button" value="Pinchame" onclick="muestra()" />
```

En este caso no podemos utilizar la variable "this" con lo que tenemos que pasar el elemento en cuestión que queremos variar.

```
function resaltar(elemento) {switch(elemento.style.borderColor) {... }}  
  
<div style="width:150px; height:60px; border:thin solid silver"  
onmouseover="resaltar(this)" onmouseout="resaltar(this)">  
  
Sección de contenidos...</div>
```

Manejadores de eventos semánticos

Esta implementación del manejador se basa en la externalización del código JavaScript, seleccionando el componente al que queremos agregar un manejador de evento y desvinculando completamente nuestro código HTML del código JavaScript.

Es la evolución ideal debido a sus múltiples ventajas entre las que entran la reutilización del código HTML desvinculándola de todo funcionamiento mediante JavaScript, pudiendo reescribir dicho funcionamiento continuamente.

```
// Función externa  
  
function muestra() {alert('Gracias');}  
  
// Asignar la función externa al elemento  
  
document.getElementById("boton").onclick = muestra;  
  
// Elemento XHTML  
  
<input id="boton" type="button" value="Pinchame" />
```

Objetos en Javascript y la especificación JSON

JSON (acrónimo de JavaScript Object Notation), es un formato para el intercambio de datos por la red, donde usualmente se utilizaba XML. Es un conjunto de datos, comprendidos entre los que puede medir JavaScript que son objetos, Arrays, cadenas, booleanos y números en Javascript.

Llegó sobre 2001 gracias al apoyo incondicional de Douglas Crockford. Yahoo! ayudó a su difusión gracias a la adición de este formato en algunos de sus servicios web más innovadores. Google comienza a realizar sus feeds en JSON para su protocolo web GData a finales del 2006.

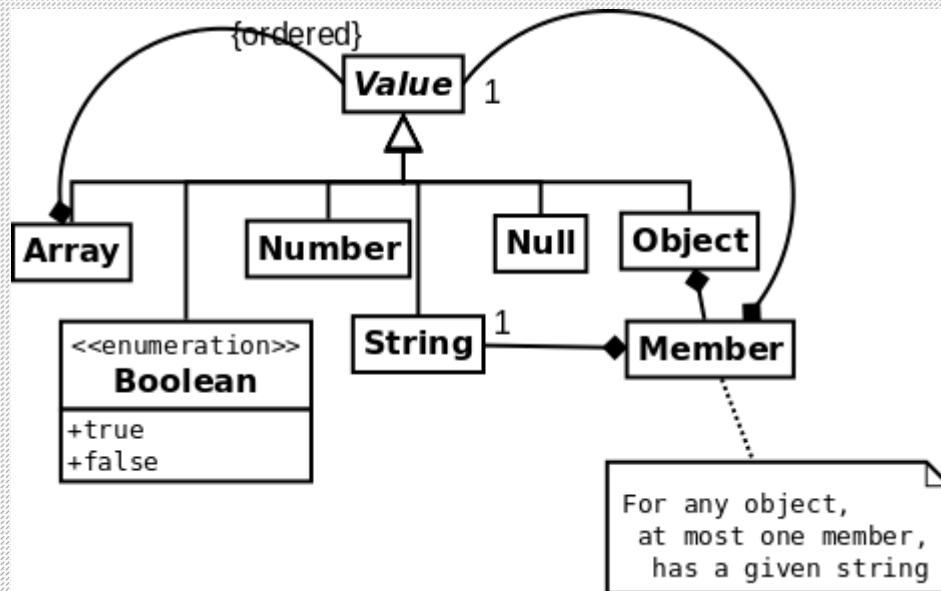
Es considerado como un lenguaje independiente de formato de los datos cuya especificación es descrita en RFC4627.

```
{  
  "id" : "0001",  
  "type" : "donut",  
  "name" : "Cake",  
  "image" : {  
    "url" : "images/0001.jpg",  
    "width" : 200,  
    "height" : 200  
  },  
  "thumbnail" : {  
    "url" : "images/thumbnails/0001.jpg",  
    "width" : 32,  
    "height" : 32  
  },  
  "dateEntry" : "2010-12-05"  
}
```

Particularidades de JSON sobre Javascript

Algunas de las particularidades o reglas del formato JSON a tener en cuenta son:

- Son duplas nombre-valor y los nombres van delimitados por comillas, tanto simples como dobles, aunque pueden aparecer sin ellas.
- JSON puede representar los seis tipos de valores de JavaScript: objetos, Arrays, números, cadenas, booleanos y null.



- Las fechas no son un tipo de objeto propio.
- Los números no pueden ir precedidos de ceros a no ser en el caso de notación decimal (Ejemplo: 0.001).
- JSON es considerado un lenguaje independiente
- Sus objetos deben ser considerados como cadenas Javascript, no como objetos nativos.

Objetos Literales

```

var oCliente = {
  dni : "44035648",
  nombres : "Maria",
  apellidos : "Jimenez",
  edad : 22,
  activo : true
};
  
```

Almacenan información en pares nombre : valor

```
color : "rojo",
```

Se puede acceder a Estas propiedades

Mediante el nombre del objeto **y la sintaxis de punto.**

```
alert(oCliente.activo);
```

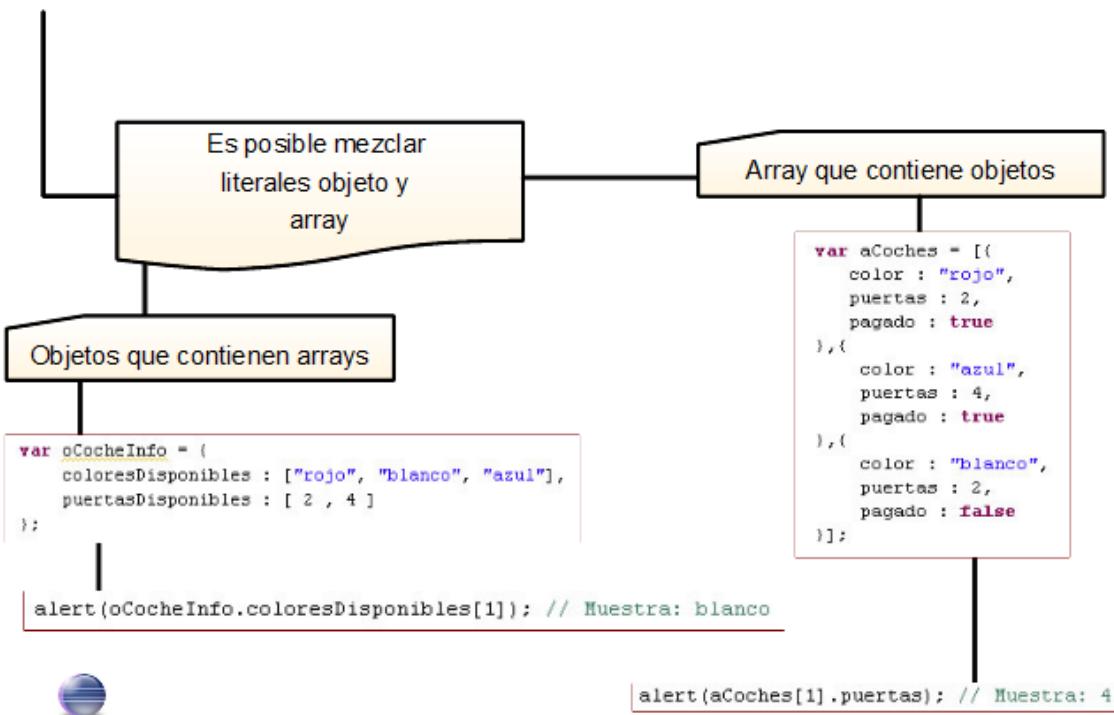
Mediante corchetes y nombre de la propiedad

```
alert(oCliente["activo"]);
```

4.1

Es posible mezclar objetos y arrays, arrays y objetos, de esta manera:

Mezclar Literales



Transformar JSON a String y String a JSON

Muchos de los códigos JavaScript que utilizan JSON para su funcionamiento o transmisión de información necesitan transformar en numerosas ocasiones a String (cadenas de texto) y viceversa.

Hay muchas maneras de hacer eso, antiguamente se utilizaba una función denominada "eval" que realizaba esta transformación (no sin muchos fallos y desventajas)

También se pueden realizar estas transformaciones con bibliotecas de terceros como pueden ser las de JQuery y Mootools (entre otras).

Sin embargo, con la llegada del ECMAScript 5, se ha implementado un nuevo objeto JSON basado en la API programada por el propio Douglas Crockford. Sus métodos más interesantes son parse() y stringify().

parse() → transforma de string a json.

stringify() → transforma de json a string.

Ejercicio propuesto

Construya un JSON con una dupla cuyo valor sea un entero, otra una cadena, y otra un array con otros objetos.

Respuesta

```
var json = { "id":1, "nombre":'Fernando',
  "domicilios":[{"calle":"Laudinos, 16"}, {"calle":"Trenes, 5"}]};
```

El estándar DOM

En el momento en el que se desarrolló el lenguaje XML, apareció la imprescindible necesidad de procesamiento y manipulación del contenido de los archivos XML mediante los lenguajes de desarrollo.

XML es sencillo de redactar pero complejo para su procesamiento y manipulación. Por ello, surgen algunas técnicas entre las que se encuentra DOM.

DOM (Document Object Model) es un agregado de utilidades diseñadas para la manipulación de XML. Además, DOM también se usar para manipulación de documentos XHTML y HTML.

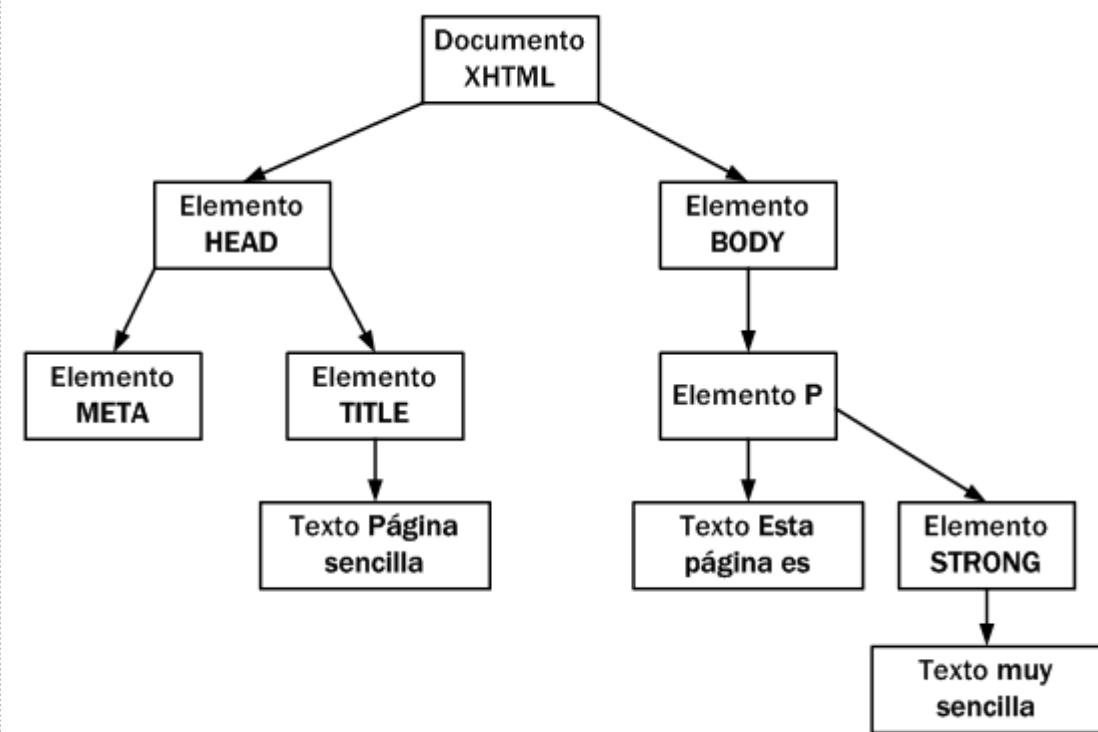
DOM es una API de funciones que se pueden usar para la manipulación de las páginas XHTML de forma eficiente y rápida.

Antes de usar las funciones, DOM convierte internamente el XML original en una estructura fácilmente manejable formada por una jerarquía de nodos. De esta manera, DOM transforma el XML en una serie de nodos interconectados en árbol.

El árbol que se genera no representa únicamente los contenidos del fichero origen (mediante los nodos del árbol) sino que representa sus relaciones (mediante las ramas del árbol que conectan los nodos).

En ocasiones DOM se asocia con el desarrollo web y con JavaScript, la API de DOM es independiente de cualquier lenguaje de desarrollo. DOM está disponible en la mayoría de lenguajes de desarrollo empleados comúnmente.

```
<!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>
```



Manipulación de un documento HTML con DOM y JavaScript

Una de las principales ventajas del uso del DOM es que permite a los desarrolladores web disponer de un control preciso sobre la estructura o forma del documento HTML o XML que están controlando.

Las funciones que usa DOM permiten añadir, eliminar, modificar y reemplazar cualquier nodo de cualquier documento sencillamente.

La primera especificación de DOM (DOM Level 1) se definió en el año 1998 y homogeneizó el desarrollo del DHTML o HTML dinámico en todos los

navegadores, ya que permitía variar el contenido de las páginas sin necesidad de volver a cargar toda la página entera.

Los documentos XML y HTML son convertidos por DOM en una jerarquía de nodos. Los nodos pueden ser de diferentes tipos.

- Document: nodo raíz de los documentos HTML y XML. Todos los demás salen de él.
- DocumentType: nodo que contiene la representación del DTD empleado en la página (indicado mediante el DOCTYPE).
- Element: contenido definido por un par de etiquetas (o tags) de apertura y cierre (`<etiqueta>...</etiqueta>`) o de una etiqueta abreviada que se autocierra (`<etiqueta/>`). Es el único nodo que puede tener tanto nodos hijos como atributos.
- Attr: el par nombre-de-atributo/valor.
- Text: el contenido del texto que se halla entre una etiqueta de apertura y una de cierre. También guarda el contenido de una sección de tipo CDATA.
- CDataSection: nodo que muestra una sección tipo `<![CDATA[]]>`.
- Comment: un comentario de XML.
- Y otros menos usuales: DocumentFragment, Entity, EntityReference, ProcessingInstruction y Notation.

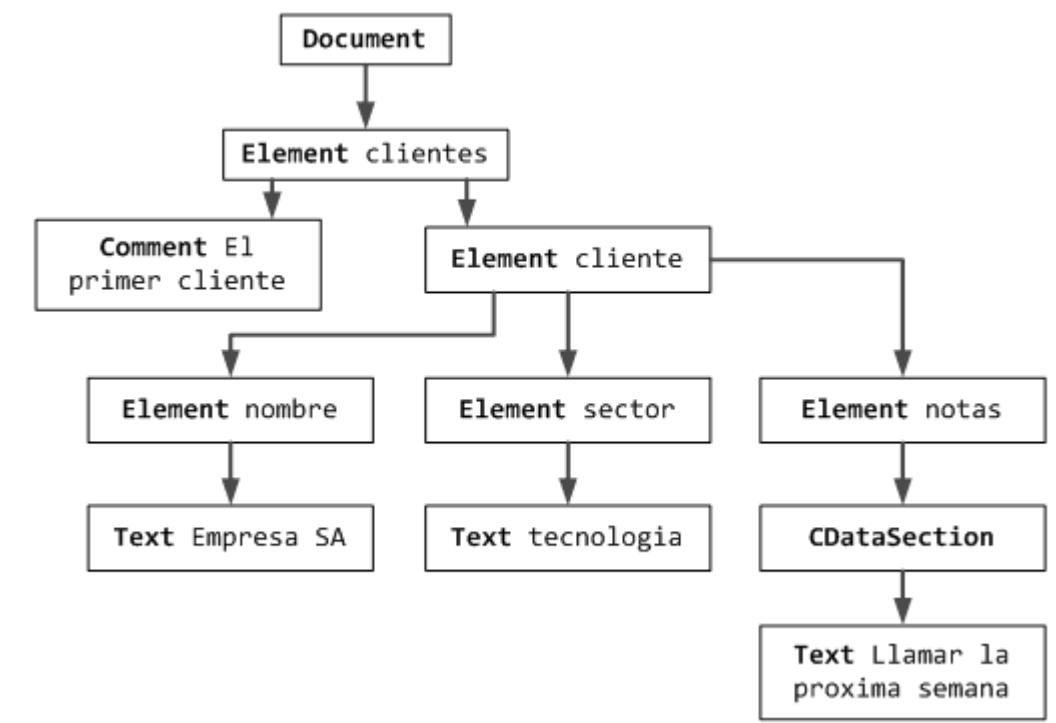
Ejercicio propuesto

Dado este XML:

```
<?xml version="1.0"?>
<clientes>
  <!-- El primer cliente -->
  <cliente>
    <nombre>Empresa SA</nombre>
    <sector>Tecnologia</sector>
    <notas><![CDATA[
      Llamar la proxima semana
    ]]></notas>
  </cliente>
</clientes>
```

Dibuje su árbol jerárquico.

Respuesta



Selectores y DOM

Una vez DOM ha formado automáticamente el árbol completo de nodos de la página, ya es posible usar sus funciones para obtener la información sobre los nodos o controlar su contenido.

JavaScript crea el objeto "Node" para tratar las propiedades y métodos necesarios para el procesamiento y manipulación de los documentos.

Primeramente, el objeto Node crea las siguientes constantes para la identificación de los tipos de nodos:

- `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`

Y a partir de este momento, podemos usar cualquier función o propiedad de DOM para nuestro código JavaScript.

Propiedad/Método	Valor devuelto	Descripción
<code>nodeName</code>	<code>String</code>	El nombre del nodo (no está definido para algunos tipos de nodo)
<code>nodeValue</code>	<code>String</code>	El valor del nodo (no está definido para algunos tipos de nodo)
<code>nodeType</code>	<code>Number</code>	Una de las 12 constantes definidas anteriormente
<code>ownerDocument</code>	<code>Document</code>	Referencia del documento al que pertenece el nodo
<code>firstChild</code>	<code>Node</code>	Referencia del primer nodo de la lista <code>childNodes</code>
<code>lastChild</code>	<code>Node</code>	Referencia del último nodo de la lista <code>childNodes</code>
<code>childNodes</code>	<code>NodeList</code>	Lista de todos los nodos hijo del nodo actual
<code>previousSibling</code>	<code>Node</code>	Referencia del nodo hermano anterior o <code>null</code> si este nodo es el primer hermano
<code>hasChildNodes()</code>	<code>Boolean</code>	Devuelve <code>true</code> si el nodo actual tiene uno o más nodos hijo
<code>attributes</code>	<code>NamedNodeMap</code>	Se emplea con nodos de tipo <code>Element</code> . Contiene objetos de tipo <code>Attr</code> que definen todos los atributos del elemento
<code>appendChild(nodo)</code>	<code>Node</code>	Añade un nuevo nodo al final de la lista <code>childNodes</code>
<code>removeChild(nodo)</code>	<code>Node</code>	Elimina un nodo de la lista <code>childNodes</code>
<code>replaceChild(nuevoNodo, anteriorNodo)</code>	<code>Node</code>	Reemplaza el nodo <code>anteriorNodo</code> por el nodo <code>nuevoNodo</code>
<code>insertBefore(nuevoNodo, anteriorNodo)</code>	<code>Node</code>	Inserta el nodo <code>nuevoNodo</code> antes que la posición del nodo <code>anteriorNodo</code> dentro de la lista <code>childNodes</code>

Actividad Práctica 1

Ajax

AJAX se presentó por vez primera en el artículo "Ajax: A New Approach to Web Applications" de Jesse James Garrett en 2005. Anteriormente no existía un término que hiciera referencia a un tipo de programación web que estaba surgiendo.

Realmente, el término AJAX es un acrónimo (Asynchronous JavaScript + XML).

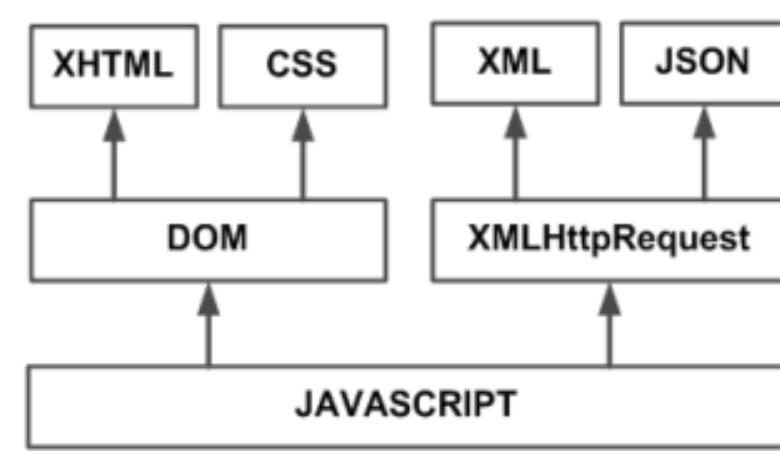
El artículo define AJAX de la siguiente forma:

Ajax no es una tecnología en sí mismo. En realidad, se trata de varias tecnologías independientes que se unen de formas nuevas y sorprendentes

Funcionamiento y tecnologías implicadas

Tecnologías que forman AJAX

- XHTML y CSS, crea una presentación basada en estándares.
- XMLHttpRequest, es el objeto encargado del intercambio asíncrono de información.
- DOM, para la manipulación e interacción dinámica de la capa de presentación.
- XML, XSLT y JSON, son las tecnologías que constituyen el intercambio y la manipulación de la información.
- JavaScript, como unión de todas tecnologías.

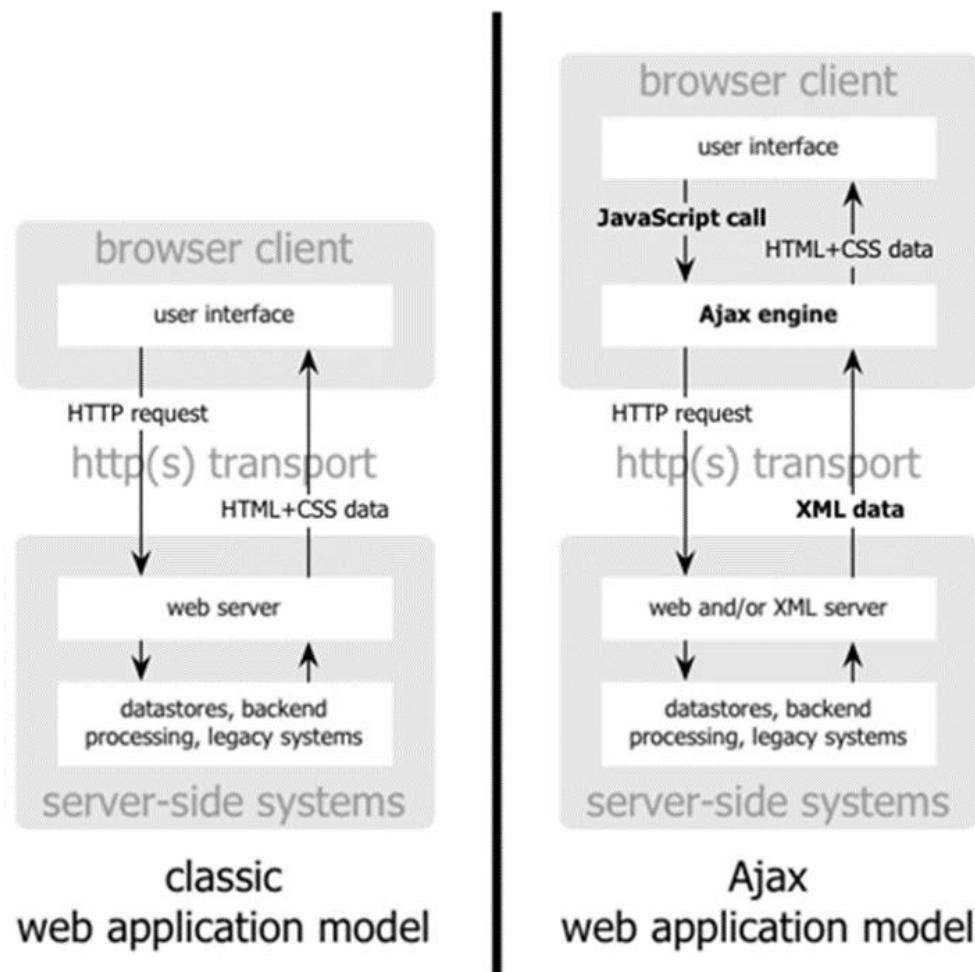


Funcionamiento de AJAX

El desarrollo de aplicaciones AJAX requiere un conocimiento avanzado de todas y cada una de las tecnologías anteriores de cada uno de los componentes.

En los antiguos desarrollos web, acciones que realice nuestro cliente sobre la página desencadenaban llamadas al servidor. Y una vez el servidor hubiera acabado con ese tráfico de información devolvía y, por tanto, recargaba, la página web en nuestro cliente.

Viendo el esquema inferior, el de la izquierda muestra el modelo antiguo para el desarrollo web. El de la derecha hace entrever el proceso de AJAX.

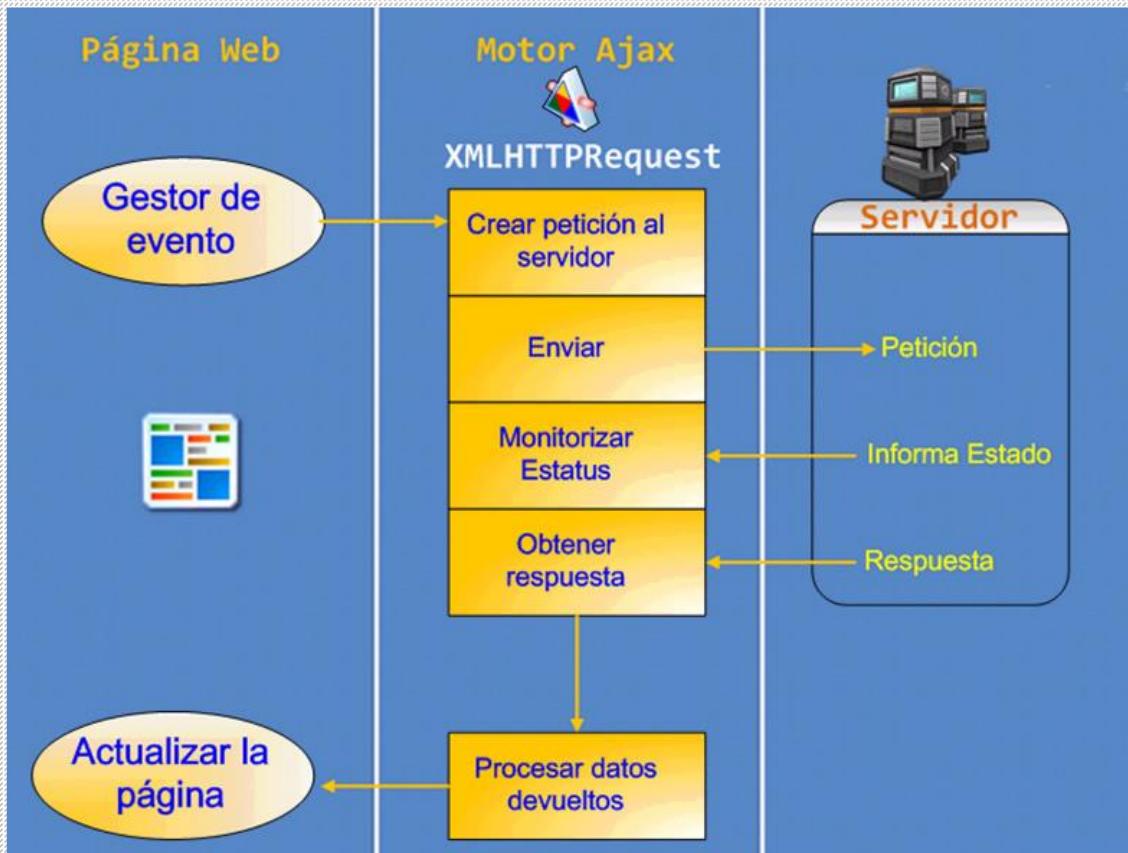


La técnica antigua que permite crear desarrollos web funciona correctamente, pero ralentiza la aplicación y el usuario recibe a cada interacción una petición del servidor que le obliga a recargar la página.

Esto suele ser bastante molesto para el usuario porque a cada cosa que haga se recarga la página por lo que su experiencia con la aplicación es bastante lenta y tediosa.

AJAX mejora la interacción del usuario con nuestra web, evita las recargas constantes, ya que el tráfico de información con el servidor se produce en un segundo plano.

Peticiones ajax y manejo de respuestas



El objeto XMLHttpRequest:

- Objetivo: realizar peticiones asíncronas al servidor.
- Es la columna vertebral de todos los desarrollos con AJAX.
- Aceptado por todos los clientes web (Microsoft lo mete en IE 5 como un objeto ActiveX).

Propiedades del objeto XMLHttpRequest

Propiedades	Descripción
onreadystatechange	Determina que función será llamada cuando la propiedad readyState del objeto cambie.
readyState	Número entero que indica el status de la petición: 0 = No iniciada 1 = Cargando 2 = Cargado 3 = Interactivo 4 = Completado
responseText	Datos devueltos por el servidor en forma de string de texto
responseXML	Datos devueltos por el servidor expresados como un objeto documento.
status	Código estatus HTTP devuelto por el servidor: 200 = OK (Petición correcta) 204 = No Content (Documento sin datos) 301 = Moved Permanently (Recurso Movido) 401 = Not Authorized (Necesita autenticación) 403 = Forbidden (Rechazada por servidor) 404 = Not Found (No existe en servidor) 408 = Request Timeout (Tiempo sobrepasado) 500 = Server Error (Error en el servidor)

Métodos del objeto XMLHttpRequest

Propiedades	Descripción
abort()	Detiene la petición actual.
getAllResponseHeaders()	Devuelve todas las cabeceras como un string.
getResponseHeader(x)	Devuelve el valor de la cabecera x como un string.
open('method', 'URL', 'a')	Especifica el método HTTP (por ejemplo, GET o POST), la URL objetivo, y si la petición debe ser manejada asincrónicamente (Si, a='True' defecto; No, a='false').
send(content)	Envía la petición
setRequestHeader('label' , 'value')	Configura un par parámetro y valor label=value y lo asigna a la cabecera para ser enviado con la petición.

Actividad práctica 2

Un ejemplo práctico

A continuación vamos a ver un pequeño ejemplo práctico, con una petición GET al servidor para conseguir un fichero llamado *ajax.txt*.

```
<!DOCTYPE html>
<html>
<head>
<script>
function loadXMLDoc()
{
var xmlhttp;
if (window.XMLHttpRequest)
{// code for IE7+, Firefox, Chrome, Opera, Safari
xmlhttp=new XMLHttpRequest();
}
else
{// code for IE6, IE5
xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
}
xmlhttp.onreadystatechange=function()
{
if (xmlhttp.readyState==4 && xmlhttp.status==200)
{
document.getElementById("myDiv").innerHTML=xmlhttp.responseText;
}
}
xmlhttp.open("GET","ajax.txt",true);
xmlhttp.send();
}
</script>
</head>
<body>
<div id="myDiv"><h2>Let AJAX change this text</h2></div>
<button type="button" onclick="loadXMLDoc()">Change Content</button>
</body>
</html>
```

Vamos a analizar el código superior.

El botón de mi html lanza con el evento click la función *loadXMLDoc()* la cual crea una variable llamada *xmlhttp*.

El primer condicional nos sirve para saber si nuestro navegador actual puede crear el objeto XMLHttpRequest que dependiendo del navegador lo creará de una manera u otra.

Abrimos la comunicación GET para pedir el fichero *ajax.txt* y enviamos la petición (se puede enviar tanto asíncrona como síncronamente dependiendo del valor del booleano).

La función `onreadystatechange()` se ejecutará a cada cambio de estado y cuando la petición sea de 200 y el *readystate* de 4 significará que se ha completado con éxito (con cualquier otra solución debemos tratar el error o los errores que hemos visto en las tablas anteriores).

Finalmente seleccionamos el *div* en cuestión (a partir del API del DOM) y cambiamos su texto plano por el que me ha venido en mi fichero *ajax.txt*.

XML vs. JSON

Si comparamos los dos formatos, queda claro que uno no es mejor que otro. Como siempre, de lo que se trata, es saber cómo queremos tratar nuestra información y con qué queremos conectarnos para ese trato.

Si estamos hablando de envío de datos de un extremo a otro (como en el caso de dos servicios web o para dos servidores independientes) XML parece una elección correcta. Aunque, también hay que decir que en el caso de los servicios web han aumentado el uso de REST frente a SOAP, servicios que van enfocados al uso de JSON.

Si son datos que debemos transmitir en una misma aplicación internamente o, aparte, estamos haciendo uso de AJAX, JSON debería ser nuestra elección sin ningún tipo de dudas ya que la devolución del objeto de AJAX es fácilmente convertible.

Ejercicio propuesto

Dado este código:

```
<!DOCTYPE html>
<html>
<head>
<script>
function loadXMLDoc()
{
var xmlhttp;
if (window.XMLHttpRequest)
{// code for IE7+, Firefox, Chrome, Opera, Safari
xmlhttp=new XMLHttpRequest();
}
else
{// code for IE6, IE5
xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
}
xmlhttp.onreadystatechange=function()
{
1
{
 xmlDoc=xmlhttp.responseXML;
...
document.getElementById("myDiv").innerHTML=txt;
}
}
2
xmlhttp.send();
}
</script>
</head>
<body>
<h2>My CD Collection:</h2>
<div id="myDiv"></div>
<button type="button" onclick="loadXMLDoc()">Get my CD collection</button>
</body>
</html>
```

Rellenar las líneas 1 y 2 para que el programa recupere con éxito los datos de un fichero llamado cd_catalog.xml en el servidor.

Respuesta

```
1- if (xmlhttp.readyState==4 && xmlhttp.status==200)
2- xmlhttp.open("GET","cd_catalog.xml",true);
```

Actividad 3, 4 y 5

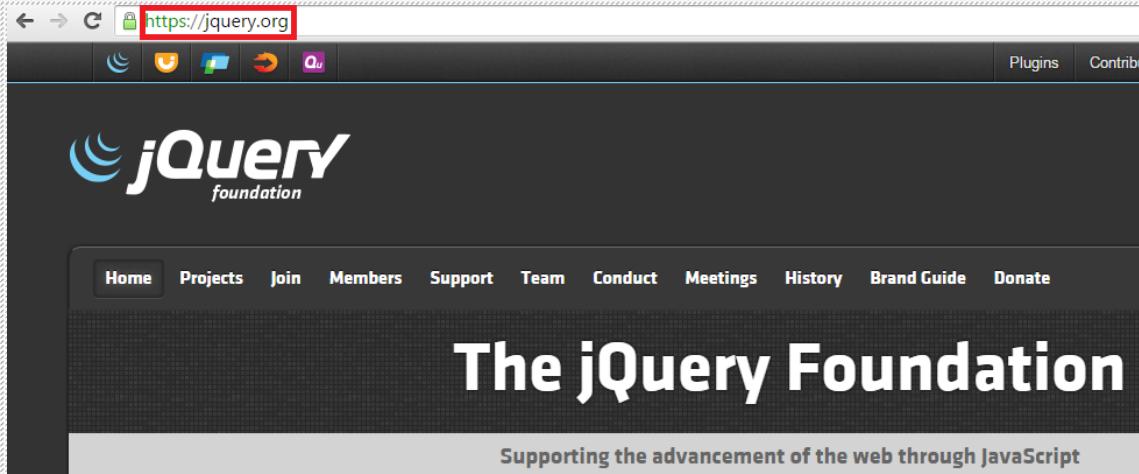
La librería JQuery

Descarga y utilización

JQuery es una librería JavaScript que nos permite acceder a los objetos del DOM de manera simple creada por John Resig (Mozilla).

Como los desarrollos web son cada vez más complejos, con funcionalidades de autocompletar, drag & drop, validaciones y un largo etcétera; se necesitan algunas herramientas que no nos las hagan desarrollar desde cero. Esto son las librerías de terceros, entre ellas JQuery, que están hechas para simplificar y aclarar nuestro código JavaScript. Adicionalmente nos suelen resolver los problemas de códigos que funcionen de manera diferente en distintos navegadores.

Para utilizarla primeramente debemos descargar la librería del sitio oficial (<https://jquery.org/>)



Después, debemos incluirla en nuestras páginas web mediante esta línea:

```
<script type="text/javascript" src="jquery.js"></script>
```

Del sitio oficial de JQuery podemos descargar la versión descomprimida que ocupa alrededor de 60 Kb que es el archivo jquery.js que vemos en el atributo "src", sin embargo cuando la incluyamos en nuestra aplicación (y por motivos de peso) deberíamos descargar la versión comprimida que pesa 20 Kb.

Ventajas al usar JQuery

- Ahorra muchas líneas de código.
- Transparenta el soporte de nuestra aplicación para los principales navegadores.
- Provee un mecanismo para capturar los eventos.
- Provee un conglomerado de funciones para hacer animaciones en la página de una manera muy simple.
- Integra funcionalidades para trabajar con AJAX.

Manera de desarrollar con JQuery

En las librerías de terceros (como en muchas otras materias) debemos adaptarnos a sus mecanismos de uso ya que intentar desarrollar como lo hacemos habitualmente podría hacer que nuestro código no tuviera el nivel que fuera de esperar. Lo adecuado sería ir viendo cual es la mecánica al trabajar con dicha librería mediante problemas sencillos e ir complicándolo a medida que entendemos la propia librería.

La función principal de JQuery se llama \$.

A la función \$ le podemos pasar distintos valores:

`x=$(document);`

...

`x=$("#boton1");`

En el primer caso le estamos pasando todo el documento (nuestra página) y en el segundo caso le estamos pasando el identificador (id) de un componente de nuestra página mediante un selector (como veremos en el apartado siguiente).

La función \$ nos devuelve un objeto de tipo JQuery.

Vamos a ver este código de ejemplo que nos servirá de base para los siguientes:

```
var x;  
x=$(document);  
x.ready(iniEventos);
```

Inicialmente nos creamos una variable en JavaScript a la que llamamos x. La inicializamos una línea más tarde dándole el valor de todo el documento y, por tanto, la variable x contiene como valor nuestro documento. En la tercera línea

llamamos al método ready de JQuery y le pasamos como parámetro el nombre de la función a la que queremos llamar. El método ready tiene una funcionalidad: cuando todos los elementos de nuestro documento se carguen en el navegador llamará a la función que le hemos pasado mediante parámetro.

El código de esta función:

```
function iniEventos (){var x; x=$("#b1"); x.click(presionB1);}
```

Usamos de nuevo la función \$ para crear un objeto de la clase JQuery pero ahora asociándolo a un botón que se encuentra en el documento y que tiene el identificador (id) de b1. Por último llamamos al método click pasando como parámetro el nombre de la función que se ejecutará al presionar dicho botón.

Este ha sido un ejemplo simplificado de la utilización de JQuery.

Selectores

Selector por ID

El primer selector que nos encontramos es aquel que selecciona un componente dentro de nuestro mapa documental mediante su atributo "id".

Como sabemos, el atributo "id" que tiene un componente debería ser único dentro del documento (esto es que no hubiera otro componente con el mismo valor de id)

Para seleccionar únicamente un componente se usa este tipo de selector y su sintaxis es la que sigue:

```
var a = $("#nombre_del_id");
```

Como se ve, hay que preceder al id del signo '#' y entrecomillar. Dentro de la variable a, por tanto, estará nuestro componente seleccionado mediante su id.

Selector por tipo

El selector por tipo es aquel que selecciona por el nombre de la etiqueta (o tag), es decir, por el tipo de los componentes que se encuentran dentro de nuestro mapa documental.

En contra del selector visto anteriormente, este selector puede seleccionar más de un elemento por lo que devuelve un array de valores con todos los elementos seleccionados.

Su sintaxis es la que sigue:

```
var a = $("tipo");
```

Únicamente debemos ingresar en el string entrecomillado el tipo de los elementos que queremos seleccionar, como:

```
var a = $("div");
```

En este ejemplo, la variable a contendría un array con todos los div que hubiera en nuestro documento.

Selector por clase

El selector por clase es un tipo de selector que selecciona a través del atributo class que tengan los componentes.

Así, con un mismo selector, podemos seleccionar componentes de diferentes tipos solo con que compartieran la misma clase.

Como ocurre con el selector por tipo, este tipo de selector puede seleccionar más de un componente a la vez con lo que nos devolverá un array de elementos.

Su sintaxis es la que sigue:

```
x$(".nombre_de_clase");
```

Usamos el signo “.” al inicio del nombre de nuestro estilo para indicar a la librería que estamos seleccionando por clase. Como hemos dicho diferentes tipos de componentes pueden tener la misma clase así que:

```
x$(".redondeado");
```

Esta línea seleccionaría todos los elementos que tuvieran la clase redondeado.

```
<input class="redondeado" ... />  
<div class="redondeado" ... />
```

Así que x contendría al final un array de dos posiciones, una para el input y otra para el div.

Selectores anidados

Podemos anidar selectores con una separación mediante de esta manera:

```
var a = $("div.redondeado");
```

Con esta línea lo que seleccionaría nuestro selector sería un div (o varios) y dentro de ellos aquellos componentes que tuvieran la clase redondeado. La selección se lee de izquierda a derecha siendo las siguientes selecciones discriminantes de las anteriores. Se pueden poner tantas como si quieran mientras tengan sentido:

```
var a = $("#uno div.redondeado");
```

Como anotación podemos comentar que no tiene sentido poner el selector por id en el medio de una selección o al final porque se entiende que este tipo de selector es único y, por tanto, no se necesitaría discriminar las selecciones anteriores para llegar a él.

Agrupación de selectores

Quizás queramos unir en una misma variable diferentes selecciones a la vez. JQuery nos ofrece una sintaxis simple para ello:

```
var a = $("#uno, div, .redondeado");
```

La separación por comas dentro de nuestra selección se entiende cuando, siguiendo el ejemplo, queremos seleccionar el componente cuyo id es uno, todos los componentes de tipo div y todos los componentes que tengan la clase redondeado. Todos ellos quedarían en el mismo array guardado como valor en la variable a.

Ejercicio propuesto

Elija la opción que:

Seleccione únicamente todos los li que son contenidos en el primer ul sin seleccionar ninguno del segundo ul.

```
<h2>Lista 1</h2>
<ul id="lista1">
<li>Opción número 1</li>
<li>Opción número 2</li>
<li>Opción número 3</li>
<li>Opción número 4</li>
</ul>
<h2>Lista 2</h2>
<ul id="lista2">
<li>Opción número 1</li>
<li>Opción número 2</li>
<li>Opción número 3</li>
<li>Opción número 4</li>
</ul>
```

Opciones:

- a) `x=$("li");`
- b) `x=$("#lista1 ul");`
- c) `x=$("#lista1 li");`
- d) `x=$("#ul li");`

Respuesta

La respuesta correcta es la **c)**.

La **a)** selecciona todos los *li* mientras que la **b)** solo selecciona el *ul* de la lista con identificador id *lista1*. La sintaxis de la **d)** es incorrecta.

Funciones principales

- `text()`, `text(valor)`

La primera extrae el texto plano contenido en los elementos seleccionados.

La segunda reemplaza ese texto plano con lo que hayamos pasado como valor.

- `attr(nombre de propiedad)`, `attr(nombre de propiedad,valor)` y `removeAttr(nombre de propiedad)`

La primera obtiene el valor de un atributo para nuestra selección, la segunda modifica el atributo con el nuevo valor dentro de nuestra selección y la tercera elimina el atributo de nuestra selección.

- `addClass(valor)` y `removeClass(valor)`

El primero añade la clase que hemos pasado en valor a nuestra selección.

El segundo, en cambio, la elimina.

- `html()` y `html(valor)`

El primero obtiene el html contenido dentro de nuestra selección. El segundo modifica el html contenido en nuestra selección por el nuevo html que pasamos por valor.

- `show()` y `hide()`

El primero muestra nuestra selección por el navegador, el segundo la oculta. Ambas admiten medidas de tiempo para que se realicen estas funciones (medidas de tiempo en milisegundos o mediante palabras reservadas como "slow" o "fast").

- `fadeIn()` y `fadeOut()`

Estos dos métodos son muy parecidos a los dos anteriores. El primero muestra nuestra selección paulatinamente, el segundo la oculta.

Producen un efecto mucho más bonito y suelen vincularse antes o después de un show o hide. Ambas admiten medidas de tiempo para que se realicen estas funciones (medidas de tiempo en milisegundos o mediante palabras reservadas como "slow" o "fast"). También existe el faceto que admite como parámetro un número de 0 a 1 con decimales siendo 1 nuestra selección completamente visible y 0 completamente

invisible. Por lo que los números decimales comprendidos entre 0 y 1 darían a nuestra selección una apariencia transparente o *fantasmal*.

- **toggle**

Este método, cada vez que se ejecuta, cambia nuestra selección alternativamente de visible a invisible, y de invisible a visible.

- **each**

Es la iteración de los elementos de un array que nos implementa JQuery.

```
var x;  
x=$([elementos]);  
x.each([nombre de funcion])
```

La función se ejecutaría para cada uno de los elementos de nuestro array guardado en la variable x.

Manipulación de los elementos del DOM

Hay algunas funciones en JQuery preparadas para manipular los elementos del DOM. Pasamos a listar algunas de ellas:

- **empty()**

Vacia de contenido nuestra selección.

- **append(valor)**

Añade un componente o un html que pasamos por parámetro al final de nuestra selección (justo antes de la etiqueta de cierre).

- **prepend(valor)**

Lo mismo que la anterior salvo que el componente o el html se añade justo después del cierre de la etiqueta de apertura.

- **remove()**

Elimina nuestra selección de nuestro mapa documental.

- **eq(valor)**

Selecciona un componente de nuestra selección pasándole como valor la posición que ocupa en el array.

Tratamiento de eventos con JQuery

Existen funciones que sirven para añadir escuchadores de eventos a nuestras selecciones, es decir, agregar funcionalidad a nuestras selecciones en el momento en el que se escuchen alguno de los eventos que hemos descrito.

- `click()`

Escucha el evento click del ratón y se le puede pasar por parámetro la función que queramos ejecutar cuando se escuche ese evento.

- `dblclick()`

Escucha el evento dobleclick del ratón y se le puede pasar por parámetro la función que queramos ejecutar cuando se escuche ese evento.

- `mouseover() y mouseout()`

El primero se lanza cuando pasamos por encima de nuestra selección con el ratón y el segundo cuando salimos de ella, aparte se le puede pasar por parámetro a ambas la función que queramos ejecutar cuando se escuche ese evento.

- `hover(funcion1, funcion2)`

Es la fusión de mouseover y mouseout. La `funcion1` se ejecutará cuando se escuche el mouseover y la `funcion2` cuando se escuche el mouseout.

- `mousemove()`

Se ejecuta con cada movimiento del ratón en nuestra selección. Se le puede pasar por parámetro la función que queramos ejecutar cuando se escuche ese evento.

- `Mousedown() y mouseup()`

El primero escucha cuando presionamos el botón del ratón (no en un click, sino en presión) y dura lo que la presión dura. El segundo es cuando liberamos la presión en el botón del ratón. A los dos se le puede pasar por parámetro la función que queramos ejecutar cuando se escuche ese evento.

- `focus() y blur()`

El primero actúa cuando se gana el foco mientras el segundo actúa cuando se pierde. Se les puede pasar por parámetro la función que queramos ejecutar cuando se escuchen estos eventos.

Ejercicio propuesto

Dado este html, escriba un código en JQuery para que los botones realicen lo que tienen en el atributo value:

```
<ul>
<li>Primer item.</li>
<li>Segundo item.</li>
<li>Tercer item.</li>
<li>Cuarto item.</li>
</ul>
<input type="button" id="boton1" value="Eliminar la lista completa."><br>
<input type="button" id="boton3" value="Añadir un elemento al final de la lista"><br>
<input type="button" id="boton4" value="Añadir un elemento al principio de la lista"><br>
<input type="button" id="boton5" value="Eliminar el último elemento."><br>
<input type="button" id="boton6" value="Eliminar el primer elemento."><br>
```

Respuesta

```
var x=$(document);
x.ready(initializarEventos);

function inicializarEventos()
{
    var x;
    x=$("#boton1");x.click(eliminarElementos);
    x=$("#boton3");x.click(anadirElementoFinal);
    x=$("#boton4");x.click(anadirElementoPrincipio);
    x=$("#boton5");x.click(eliminarElementoFinal);
    x=$("#boton6");x.click(eliminarElementoPrincipio);
}
function eliminarElementos(){var x; x=$("#ul"); x.empty();}

function anadirElementoFinal(){var x; x=$("#ul"); x.append("<li>otro</li>");}

function anadirElementoPrincipio(){var x; x=$("#ul"); x.prepend("<li>otro</li>");}

function eliminarElementoFinal(){var x; x=$("#li"); var cantidad=x.length;
x=x.eq(cantidad-1); x.remove();}

function eliminarElementoPrincipio(){ var x=$("#li"); x=x.eq(0); x.remove();}
```

Actividad práctica 6

Ajax con JQuery

Las utilidades y las funciones que están relacionadas con AJAX son parte principal de la librería JQuery.

Tenemos un método para hacer peticiones AJAX que es `$.ajax()` (no hay que olvidar el punto entre \$ y ajax).

Aparte, se han descrito varias funciones que han partido de este método inicial que son: `$.get()`, `$.post()`, `$.load()`, etc. Estas están preparadas para tareas predeterminadas.

Sintaxis:

```
$.ajax(opciones);
```

La URL que se trata se incluye dentro del array de opciones.

```
$.ajax({
  url: '/ruta/hasta/pagina.php',
  type: 'POST',
  async: true,
  data: 'parametro1=valor1&parametro2=valor2',
  success: procesaRespuesta,
  error: muestraError
});
```

La siguiente tabla muestra todas las opciones que se pueden definir para el método `$.ajax()`:

Opción	Descripción
async	Indica si la petición es asíncrona. Su valor por defecto es true, el habitual para las peticiones AJAX
beforeSend	Permite indicar una función que modifique el objeto XMLHttpRequest antes de realizar la petición. El propio objeto XMLHttpRequest se pasa como único argumento de la función
complete	Permite establecer la función que se ejecuta cuando una petición se ha completado (y después de ejecutar, si se han establecido, las funciones de success o error). La función recibe el objeto XMLHttpRequest como primer parámetro y el resultado de la petición como segundo argumento

contentType	Indica el valor de la cabecera Content-Type utilizada para realizar la petición. Su valor por defecto es application/x-www-form-urlencoded
data	Información que se incluye en la petición. Se utiliza para enviar parámetros al servidor. Si es una cadena de texto, se envía tal cual, por lo que su formato debería ser parametro1=valor1¶metro2=valor2. También se puede indicar un array asociativo de pares clave/valor que se convierten automáticamente en una cadena tipo <i>query string</i>
dataType	El tipo de dato que se espera como respuesta. Si no se indica ningún valor, jQuery lo deduce a partir de las cabeceras de la respuesta. Los posibles valores son: xml (se devuelve un documento XML correspondiente al valor responseXML), html (devuelve directamente la respuesta del servidor mediante el valor responseText), script (se evalúa la respuesta como si fuera JavaScript y se devuelve el resultado) y json (se evalúa la respuesta como si fuera JSON y se devuelve el objeto JavaScript generado)
error	Indica la función que se ejecuta cuando se produce un error durante la petición. Esta función recibe el objeto XMLHttpRequest como primer parámetro, una cadena de texto indicando el error como segundo parámetro y un objeto con la excepción producida como tercer parámetro
ifModified	Permite considerar como correcta la petición solamente si la respuesta recibida es diferente de la anterior respuesta. Por defecto su valor es false
processData	Indica si se transforman los datos de la opción data para convertirlos en una cadena de texto. Si se indica un valor de false, no se realiza esta transformación automática
success	Permite establecer la función que se ejecuta cuando una petición se ha completado de forma correcta. La función recibe como primer parámetro los datos recibidos del servidor, previamente formateados según se especifique en la opción dataType
timeout	Indica el tiempo máximo, en milisegundos, que la petición espera la respuesta del servidor antes de anular la petición
type	El tipo de petición que se realiza. Su valor por defecto es GET, aunque también se puede utilizar el método POST
url	La URL del servidor a la que se realiza la petición

A parte, existen varias funciones relacionadas que son versiones especializadas y simplificadas de esa función. Por ejemplo, `$.get()` y `$.post()` se usan para tratar peticiones GET y POST:

```
$.get('/ruta/hasta/pagina.php'); // Petición GET simple  
// Petición GET con envío de parámetros y función que procesa la respuesta  
$.get('/ruta/pagina.php', { producto: '34' },  
    function(envio) { alert('Respuesta = '+envio);});
```

Las POST se escriben de la misma manera, únicamente cambiando `$.get()` por `$.post()`. Sintaxis:

```
$.post(url, datos, funcionManejadora);
```

El único parámetro que es obligatorio es el de 'url' donde indicamos a qué dirección vamos a ir para realizar la comunicación. El segundo parámetro (datos) es lo que queremos enviar a esa dirección (habitualmente un objeto JSON) y el tercer parámetro (funcionManejadora) es una función JavaScript para el tratamiento de la devolución de los datos mediante la llamada de Ajax.

Existe una versión modificada del `$.get()` que se escribe `$.getIfModified()`, que mide si la respuesta es diferente a la anteriormente recibida.

La función `$.load()` agrega el contenido de la respuesta del servidor en el componente del documento que se indica.

```
<div id="info"></div>  
$('#info').load('/ruta/hasta/pagina.php');
```

También dispone de una versión específica denominada `$.loadIfModified()` que carga la respuesta del servidor en el elemento sólo si esa respuesta es diferente a la última recibida.

Para finalizar, existen las funciones `$.getJSON()` y `$.getScript()` que cargan y ejecutan respectivamente una respuesta de tipo JSON y una respuesta con código JavaScript.

Pregunta de refuerzo 8

Funciones para CSS

La librería contiene funciones para manipular las propiedades CSS de los componentes de nuestro documento.

Una vez seleccionados un único elemento o diferentes elementos como nuestra selección podemos variar sus propiedades CSS de una manera simple con JQuery.

```
$('div').css('background');  
$('div').css('color', '#000000');  
$('div').css({ padding: '3px', color: '#CC0000' });// Establece varias propiedades CSS  
$('div').height();// Obtiene la altura en píxel del primer 'div' de la página  
$('div').height('150px');// Establece la altura en píxel de todos los 'div' de la página  
$('div').width();// Obtiene la anchura en píxel del primer 'div' de la página  
$('div').width('300px');// Establece la anchura en píxel de todos los 'div' de la página
```

Ejercicio propuesto

Dado este JSON que se encuentra en la raíz de nuestro programa en un fichero llamado datos.json:

```
{"primos": [ 1, 2, 3, 5, 7, 11, 13 ]}
```

Escriba un código que mediante JQuery y Ajax obtenga este objeto JSON del servidor, recorra el array de primos y los muestre por consola.

Respuesta

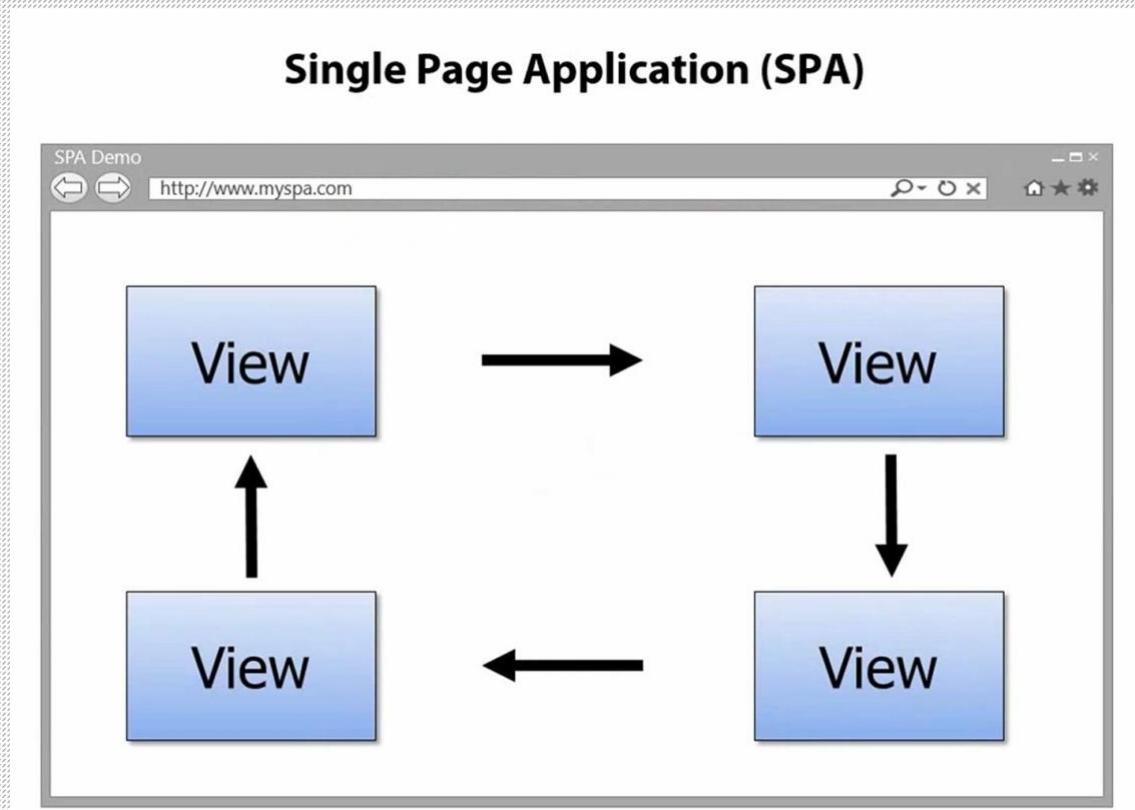
```
$.getJSON("./datos.json", function(datos) {  
    $.each(datos["primos"], function(idx,primo) {  
        alert("Número primo: " + primo);  
    });  
});
```

Actividad práctica 7

La librería AngularJS

Nos encontramos ante un framework JavaScript de código abierto que se denomina AngularJS, que está siendo respaldado por Google, y ayuda con la construcción de las Single Page Applications o aplicaciones de una sola página.

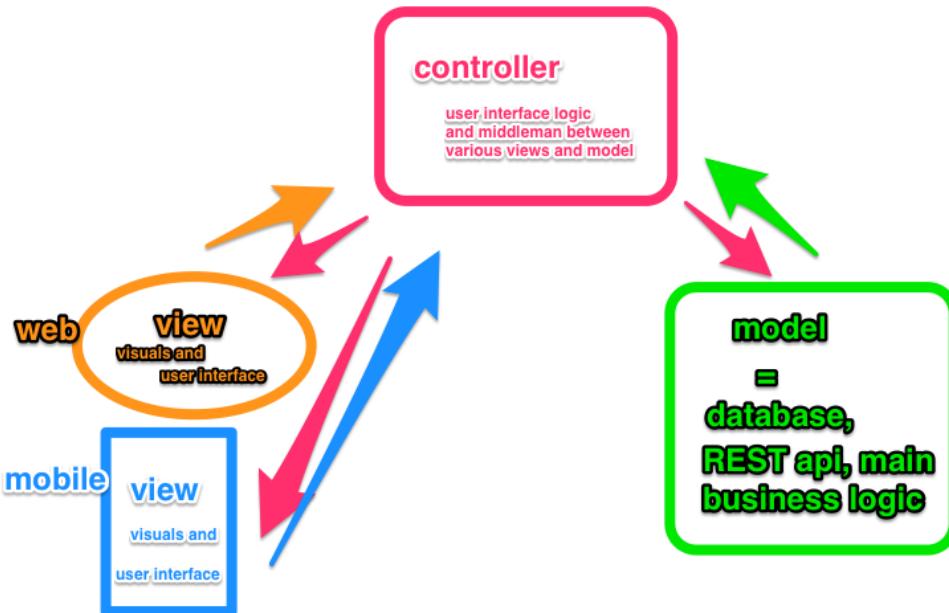
El patrón Single Page Applications define que podemos construir o desarrollar aplicaciones web en una única página html, teniendo todo el ciclo de vida seleccionado en dicha página, y variando los componentes y controles con códigos JavaScript y las librerías o frameworks como AngularJS.



Aparte, también es adecuado seguir el patrón Modelo Vista Controlador (MVC), que muchos otros frameworks de desarrollo lo programaban en el lado servidor, pero que con AngularJS se hace factible desarrollar en el lado cliente.

model-view-controller

<http://www.angularstutorial.com>

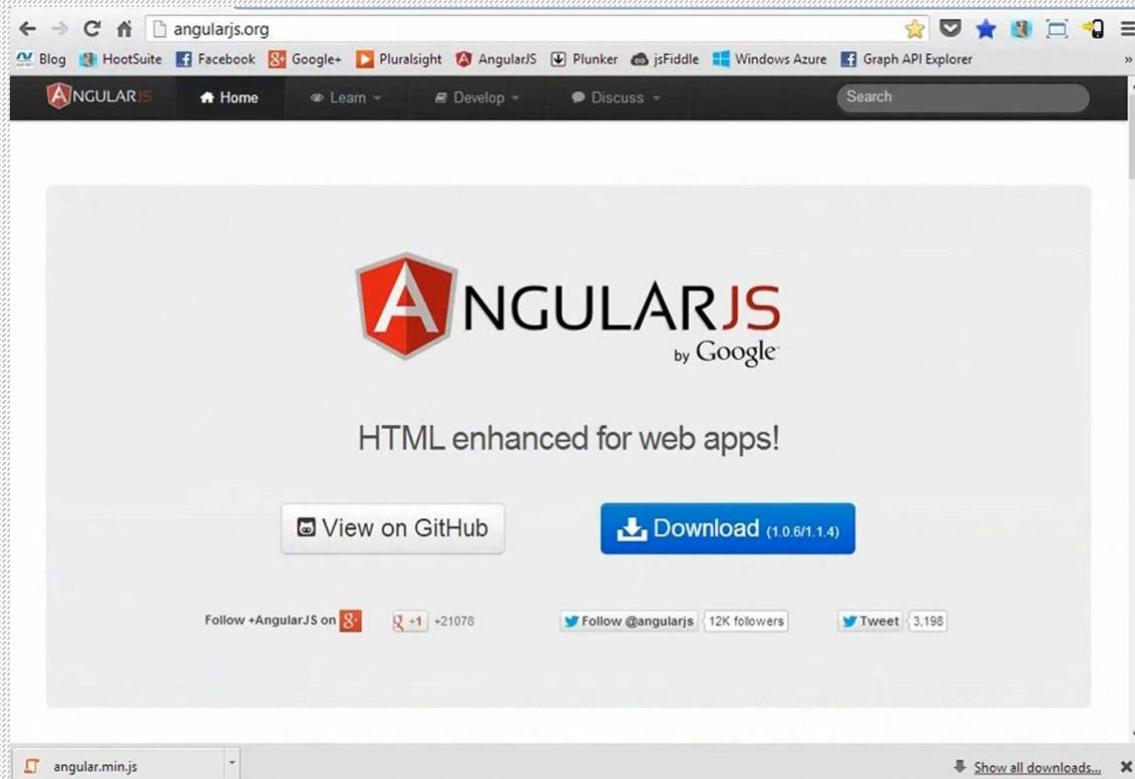


En un mundo en el que necesitamos mucha programación en la capa de las vistas, con diferentes dispositivos que pueden llegar a nuestra aplicación (desde terminales móviles hasta ordenadores de sobremesa), la utilización de frameworks como AngularJS no solo es buena sino que se hace inmediatamente necesaria.



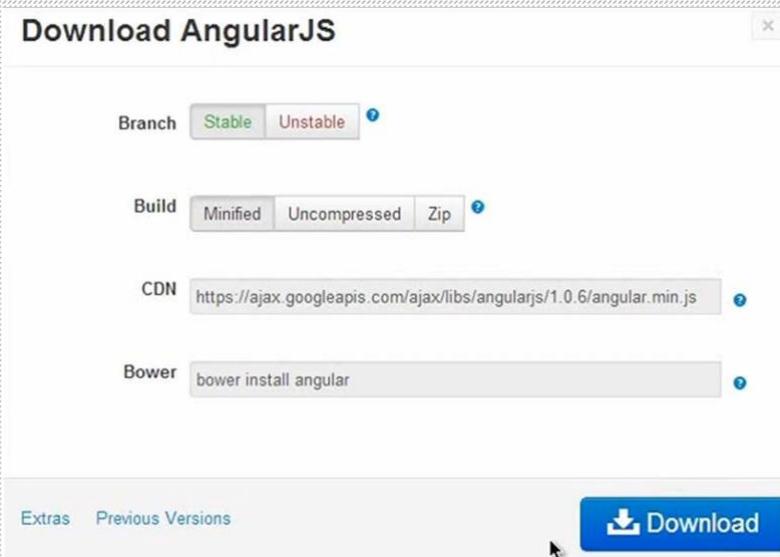
Descarga de la librería

Para empezar con AngularJS únicamente se tiene que descargar la librería que se encuentra en la página <https://angularjs.org/>



En esta misma web también podemos leer documentación muy variada y un gran número de ejemplos para poder comenzar.

Podemos descargar versiones mínimos o versiones más completas con librerías de terceros.

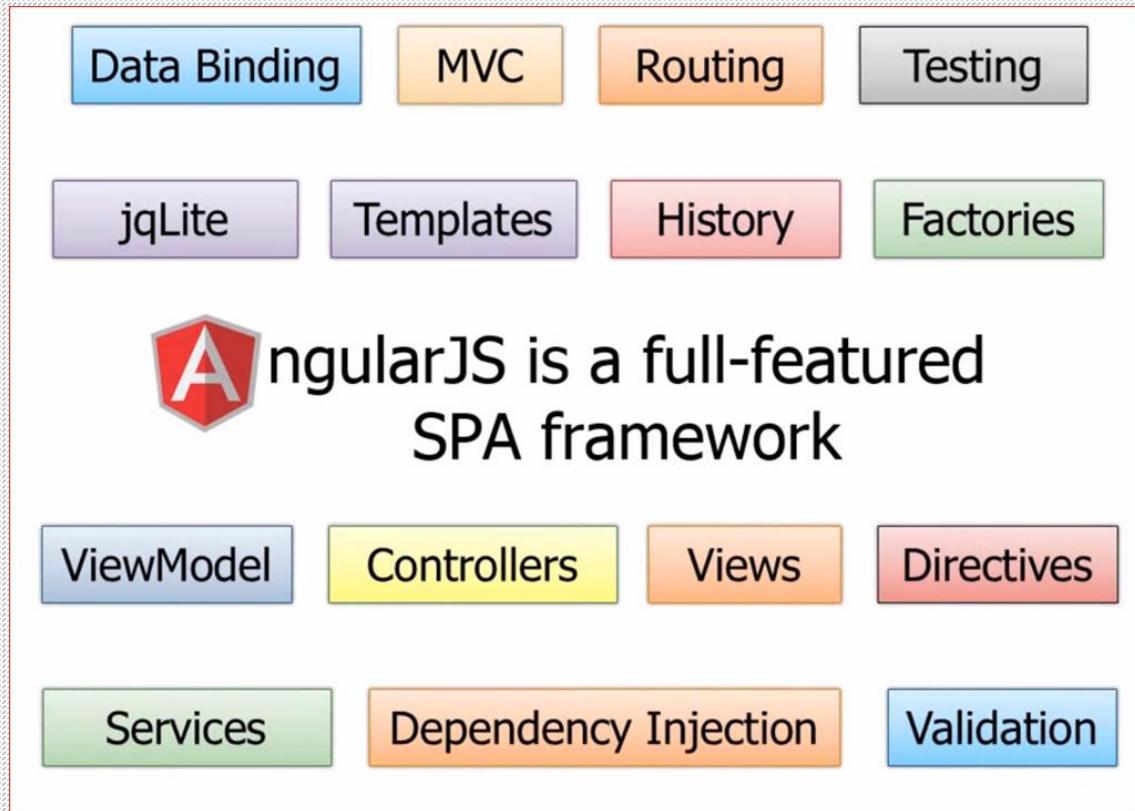


Principios de funcionamiento y elementos básicos

AngularJS revisa el HTML que puede contener atributos de las etiquetas personalizadas adicionales (de la propia librería), obedece a las directivas de los atributos personalizados, y une los elementos de entrada o salida del documento a un modelo representado por las variables de JavaScript.

Los valores de las variables de JavaScript se pueden actualizar de manera manual, o ser recuperados de los JSON estáticos o dinámicos.

Como se puede ver en la imagen, hay muchos elementos y conceptos disponibles en la librería que pasaremos a comentar durante este manual.



Directivas

Las directivas son atributos que se incluyen dentro de las etiquetas de nuestro código HTML para conectar la librería AngularJS con esos componentes que existen en nuestro mapa documental.

Directivas hay muchas, y ahora pasaremos a redactar las más importantes.

ngApp (ng-app)

Auto arranca una aplicación Angular, nos selecciona el elemento raíz y se sitúa como atributo dentro de la etiqueta que deseas que sea la raíz de la aplicación.

Se declara tal como sigue:

```
<html ng-app>
```

Aunque más adelante veremos cómo declarar módulos, se puede enlazar un módulo a esta directiva de la siguiente manera:

```
<html ng-app="nombre_del_modulo">
```

ngController (ng-controller)

Directiva que se usa para los controladores. Y aunque recuperaremos esta directiva en la sección posterior vamos a darle una definición:

La directiva que indica a la vista que trabajará con nuestro controlador y enlaza un \$scope, el modelo que esté dentro del ámbito de la directiva podrá ser accedido desde el controlador que nos hemos asignado.

La sintaxis de ngController:

```
<body>
<div ng-controller="nombre_de_controlador">
<h1>Hola </h1>
<div/>
</body>
```

ngModel (ng-model)

Directiva que muestra el modelo o dato, obtiene la información insertada por el usuario en algún componente de nuestro formulario, sea cual fuere. Si se quiere obtener el texto que nuestro cliente rellena en un input, basta con asociarle un modelo y entonces será accedido tanto en el controlador como la vista mediante el nombre del modelo.

Su sintaxis:

```
<body>
<div ng-controller="miControlador">
<label>Nombre</label><input type="text" ng-model="nombre">
```

```
<span>Hola {{nombre}}</span>
<div/>
</body>
```

ngClick (ng-click)

Relacionado con el evento click, al que se le puede asociar cualquier tipo de funcionalidad.

```
<body>
<div ng-controller="miControlador">
  <label>Nombre:</label><input type="text" ng-model="nombre">
  <button ng-click="enviar()">Enviar</button>
</div>
</body>
```

ngInit (ng-init)

Permite evaluar una expresión en el scope donde se está trabajando.

```
<body>
<div ng-controller="miControlador">
  <div>
    <button ng-click="count = count + 1" ng-init="count = 0">Enviar</button>
    <span>{{count}}</span>
  </div>
</div>
</body>
```

ngRepeat (ng-repeat)

Permite iterar una colección de datos, crear una plantilla (template) por cada elemento de la colección y mostrarlo en la vista.

```
<body>
<div ng-controller="miControlador">
  <div ng-init="lista =
```

```
[{nombre:Fer, edad:12},{nombre:Victor, edad:13},  
 {nombre:Eva, edad:14},{nombre:Susana, edad:15},  
 {nombre:Alejo, edad:16} ]"/>  
  
<ul>  
 <li ng-repeat="a in lista">{{a.nombre}}: {{a.edad}} años</li>  
</ul>  
</div>  
</div>  
</body>
```

ngChange (ng-change)

Detecta cualquier variación que se produzca dentro de una etiqueta de entrada, la manera de usarla es la siguiente:

```
<body>  
 <div ng-controller="miControlador">  
   <input type="checkbox" ng-model="si" ng-change="favor()"> A favor  
   <input type="checkbox" ng-model="no" ng-change="contra()"> En contra  
   <h3>Total Votos: {{total}}</h3>  
 </div>  
</body>
```

En el lado del controlador:

```
app.controller('miControlador', function($scope){  
 $scope.total = 0;  
 $scope.favor = function (){$scope.total++};  
 $scope.contra = function (){$scope.total--};  
});
```

ngShow (ng-show) | ngHide (ng-hide)

Permiten mostrar y ocultar alguna parte de la vista según una condición que podemos asignarle. ngShow muestra y ngHide oculta.

```
<body>
  <div ng-controller="miControlador">
    <input type="checkbox" ng-model="dato1"> Muestra
    <input type="checkbox" ng-model="dato2"> Oculta
    <h3 ng-show="dato1">Hola</h3>
    <h3 ng-hide="dato2">Adiós</h3>
  </div>
</body>
```

ngBind (ng-bind)

Esta directiva tiene la misma función que las llaves {{}}, sin embargo, ng-bind tiene un mejor funcionamiento en cuanto a tiempo.

```
<body>
  <div ng-controller="miControlador">
    <input type="text" ng-model="nom">
    <span>{{nom}}</span>
    <span ng-bind="nom">{{nom}}</span>
  </div>
</body>
```

Creando nuestras propias directivas

Muchas veces necesitamos construir nuestras propias directivas y por ello AngularJS nos facilita poder crearla a nuestro gusto a medida de nuestra necesidad. Mucho del código a continuación necesitará de conceptos posteriores por lo que animamos a recuperarlo en ese momento.

```
var app = angular.module('MiModulo',[]);
app.controller('MiControlador', function($scope){
  $scope.cliente = {
    nombre: 'Jhon',
    direccion: 'Av. Jose pardo 481'
  };
});
```

```
//Nuestra directiva  
app.directive('miCliente', function() {  
  return {  
    template: 'Nombre: {{cliente.nombre}} Dirección: {{cliente.direccion}}'  
  };  
});
```

En la parte de la vista usaríamos nuestra directiva así:

```
<body>  
  <div ng-controller="MiControlador">  
    <div mi-cliente></div>  
  </div>  
</body>
```

Actividad práctica 8

Controllers (Controladores)

Los controladores en AngularJS son los encargados de controlar los datos de las aplicaciones AngularJS. Los controladores son objetos de JavaScript.

Controladores en AngularJS

Las aplicaciones AngularJS son controladas por los controladores. Un controlador no es más que un objeto JavaScript, que se crea con el constructor de objetos de JavaScript. El ámbito de la aplicación está definido por \$scope y se corresponde con el elemento HTML asociado a la aplicación.

```
<div ng-app="" ng-controller="cochesController">  
  Marca: <input type="text" ng-model="coche.marca"><br>  
  Modelo: <input type="text" ng-model="coche.modelo"><br>  
  El Coche es: {{coche.marca + " " + coche.modelo}}  
</div>  
<script>  
function cochesController($scope) {  
  $scope.coche = {
```

```

marca: "Audi",
modelo: "A5"

};

}

</script>

```

Como vemos en el ejemplo, la aplicación AngularJS está definida por la directiva *ng-app*. La aplicación corre dentro de elemento *<div>*. La directiva *ng-controller* indica el nombre del objeto correspondiente al controlador. La función *cocheController* es el constructor estándar del objeto JavaScript. El objeto controlador tiene una propiedad: *\$scope.coche*. El objeto *coche* tiene dos propiedades: marca y modelo. La directiva *ng-model* enlaza los campos *<input/>* a las propiedades del controlador (marca y modelo).

Propiedades del Controlador

Un controlador también puede tener funciones como propiedades del controlador. Ejemplo:

```

<div ng-app="" ng-controller="cocheController">
  Marca: <input type="text" ng-model="coche.marca"><br>
  Modelo: <input type="text" ng-model="coche.modelo"><br>
  El coche es: {{coche.nombreCompleto()}}
</div>

<script>
function cocheController($scope) {
  $scope.coche = {
    marca: "Audi",
    modelo: "A3",
    nombreCompleto: function() {
      var x;
      x = $scope.coche;
      return x.marca + " " + x.modelo;
    }
}

```

```
};  
}  
</script>
```

Métodos del Controlador

Un controlador también puede tener métodos.

```
<div ng-app="" ng-controller="cocheController">  
  Marca: <input type="text" ng-model="coche.marca"><br>  
  Modelo: <input type="text" ng-model="coche.modelo"><br>  
  El coche es: {{ nombreCompletoDelCoche() }}  
</div>  
<script>  
function personController($scope) {  
  $scope.coche = {  
    marca: "Audi",  
    modelo: "A3",  
  };  
  $scope.nombreCompletoDelCoche = function() {  
    var x;  
    x = $scope.coche;  
    return x.marca + " " + x.modelo;  
  };  
}  
</script>
```

Controladores en ficheros externos

En aplicaciones más grandes, es habitual tener los controladores en ficheros JavaScript externos. Solo debes copiar el código entre las etiquetas `<script>` en un fichero externo llamado `cocheController.js`.

```
<div ng-app="" ng-controller="cocheController">  
  Marca: <input type="text" ng-model="coche.marca"><br>
```

Modelo: <input type="text" ng-model="coche.modelo">

El Coche es: {{coche.marca + " " + coche.modelo}}

</div>

<script src="cocheController.js"></script>

Ejercicio propuesto

Teniendo este controlador en un fichero externo llamado controlador.js:

```
function alumnosController($scope) {  
    $scope.alumnos = [  
        {nombre:'Paco',pais:'España'},  
        {nombre:'Manuel',pais:'Chile'},  
        {nombre:'Laura',pais:'Argentina'}  
    ];  
}
```

Y teniendo este HTML:

```
<div ng-app="" ng-controller="alumnosController">  
    <ul>  
        1  
        {{ alumno.nombre + ', ' + alumno.pais }}  
    </li>  
    </ul>  
    </div>  
<script src="alumnosController.js"></script>
```

¿Cuál será la línea 1 para que liste todos los alumnos y no falle?

- a) <li ng-repeat="alumno in alumnos">
- b) <li ng-repeat="alumnos in alumno">
- c) <li ng-repeat="alumno">
- d) <li ng-repeat="alumnos">

Respuesta

La respuesta correcta es la **a)** ya que el objeto con el que parametrizamos la búsqueda es alumno (como vemos en las líneas inferiores a la 1 en el código) y la lista queda guardada en la variable alumnos que es la que está ligada al scope.

Módulos

Tus aplicaciones en AngularJS estarán compuestas por módulos. Todos los controladores de AngularJS deben pertenecer a un módulo.

Con el uso de módulos mantenemos nuestro código limpio y bien organizado.

Ejemplo de Módulo en AngularJS

En este ejemplo, "miAplicacion.js" contiene un módulo de definición de la aplicación "miAplicacion", "miControlador.js" contiene un controlador:

```
< div ng-app = "miAplicacion" ng-controller = "miControlador" >  
{{ nombre + " " + apellidos}}  
</ div >  
</s cript>  
< script src = "miAplicacion.js" ></ script >  
< script src = "miControlador.js" ></ script >
```

Los Controladores ensucian el ámbito Global de JavaScript

En todos los ejemplos que hemos ido viendo se han usado funciones Globales.

Esto no está bien.

Las variables globales y las funciones globales no se deben usar en las aplicaciones de AngularJS, ya que podrán ser sobreescritos o eliminados por otros scripts.

Para corregir este problema, en AngularJS, hacemos uso de los módulos.

Usando un controlador simple:

```
<!DOCTYPE html>
<html>
<body>
<div ng-app = "" ng-controller = "miControlador" >
{{ nombre + " " + apellidos }}
</div>
<script>
function miControlador ($scope) {
$scope.nombre = "Mario";
$scope.apellidos = "Flores";
}
</script>
<script src = "//ajax.googleapis.com/ajax/libs/angularjs/1.2.15/angular.min.js"
></script>
</body>
</html>
```

Usando un controlador perteneciente a un módulo:

```
<!DOCTYPE html>
<html>
<head>
<script src = "//ajax.googleapis.com/ajax/libs/angularjs/1.2.15/angular.min.js"
></script>
</head>
<body>
<div ng-app = "miAplicacion" ng-controller = "miControlador" >
{{ nombre + " " + apellidos }}
</div>
<script>
var app = angular.module( "miAplicacion" , []);

```

```
app.controller( "miControlador" , function ($scope) {  
    $scope.nombre = "Mario";  
    $scope.apellidos = "Flores";  
});  
</script>  
</body>  
</html>
```

Definición de los Módulos

Un consejo recomendado para las aplicaciones web, es colocar todos los script al final del documento HTML antes de cerrar la etiqueta <body>.

Esto provoca la página cargue mucho más rápido, ya que la carga del HTML no estará bloqueada por la carga de los scripts.

En muchas aplicaciones de AngularJS verás que la librería Angular es cargada en la sección <head> de la web.

En el ejemplo anterior, AngularJS es cargado en la sección <head>, lo hacemos así porque la llamada a *angular.module* sólo puede hacerse después de que haya cargado la librería Angular.

Como se podrá pensar, otra solución completamente válida, es cargar la librería de AngularJS en el <body>, antes de nuestros scripts de AngularJS.

```
<!DOCTYPE html>  
<html>  
<body>  
<div ng-app = "miAplicacion" ng-controller = "miControlador" >  
{{ nombre + " " + apellidos }}  
</div>  
<script src = "//ajax.googleapis.com/ajax/libs/angularjs/1.2.15/angular.min.js"></script>  
<script>  
var app = angular.module( "miAplicacion" , []);  
app.controller( "miControlador" , function ($scope) {
```

```
$scope.nombre = "Mario";
$scope.apellidos = "Flores";
});
</ script >
</ body >
</ html >
```

Ficheros de una Aplicación AngularJS

Ahora que ya sabemos que son los módulos, y cómo funcionan, es hora de construir nuestros propios ficheros que compondrán la aplicación AngularJS.

Tu aplicación debe tener al menos un archivo para el módulo y otro archivo por cada controlador. Primero creamos el script para el módulo, lo llamaremos "miAplicacion.js":

```
var app = angular.module( "myApp" , []);
```

Luego creamos los controladores. En este caso "miControlador.js":

```
app.controller( "miControlador" , function ($scope) {
$scope.nombre= "Fernando";
$scope.apellidos= "Gil";
});
```

Por último, editamos la página HTML para cargar estos archivos:

```
<!DOCTYPE html>
< html >
< body >
< div ng-app = "myApp" ng-controller = "myCtrl" >
{{ firstName + " " + lastName }}
</ div >
< script src = "//ajax.googleapis.com/ajax/libs/angularjs/1.2.15/angular.min.js"
></ script >
< script src = "myApp.js" ></ script >
< script src = "myCtrl.js" ></ script >
</ body ></ html >
```

Actividad práctica 9

JavaScript en el lado del servidor: NodeJS

JavaScript del lado del servidor

Las primeras evoluciones de JavaScript vivían en los navegadores. Pero esto es sólo un contexto. Debemos tener claro que JavaScript es un lenguaje "completo": Se utiliza en muchos otros contextos.

Node.js realmente es sólo otro contexto: te permite ejecutar e interpretar código JavaScript en el backend, es decir, fuera del navegador.

Para utilizar el código JavaScript en el backend, éste necesita ser interpretado y, bueno, ejecutado. Esto es lo que Node.js hace, con el uso de la Máquina Virtual V8 de Google, el mismo entorno de ejecución para JavaScript que el navegador de Google, Chrome, usa.

Aparte, Node.js aporta muchos módulos útiles, de manera que no tienes que escribir todo de cero, como por ejemplo, algo que ponga un String a la consola.

Por tanto, Node.js es ciertamente dos cosas: una librería y un entorno de ejecución.

Para hacer uso de éstas (la librería y el entorno), necesitas instalar Node.js en tu equipo.

Instalación y utilización

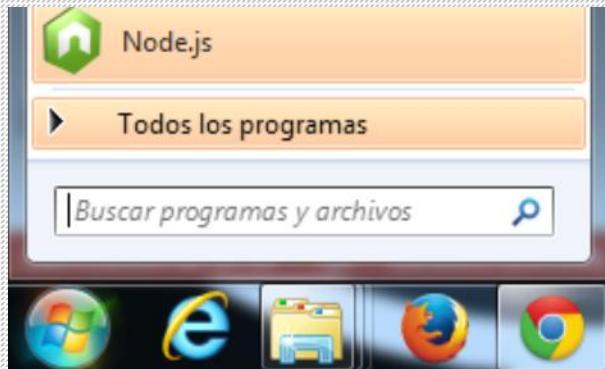
Instalación

Entrar en NodeJS (<https://nodejs.org/en/download/>) y descargar el ejecutable para instalar NodeJS según nuestro SO.

The screenshot shows the official Node.js download page at <https://nodejs.org/en/download/>. At the top, there's a navigation bar with links to HOME, ABOUT, DOWNLOADS, DOCS, FOUNDATION, GET INVOLVED, SECURITY, and NEWS. Below the navigation is a section titled "Downloads" with the sub-instruction: "Download the Node.js source code or a pre-built installer for your platform, and start developing today." There are three main download options: "Windows Installer" (msi), "Macintosh Installer" (dmg), and "Source Code" (tar.gz). Below these are two tables listing binary distributions for different platforms and architectures.

	32-bit	64-bit
Windows Binary (.exe)	32-bit	64-bit
Mac OS X Binaries (.tar.gz)	64-bit	
Linux Binaries (.tar.gz)	32-bit	64-bit
SunOS Binaries (.tar.gz)	32-bit	64-bit
ARM Binaries (.tar.gz)	ARMv6	ARMv7
Source Code	node-v4.1.0.tar.gz	

La instalación es muy sencilla y una vez instalado (a partir de nuestro ejecutable) ya podemos ejecutar NodeJs para que aparezca una consola de comandos.



Se puede desarrollar en cualquier IDE de desarrollo que permita escribir código JS de tal manera que podemos ejecutar un js desde línea de comandos (se presupone un archivo js que llamado helloworld.js)

A screenshot of a Microsoft Windows Command Prompt window. The title bar shows "C:\Windows\system32\cmd.exe - node helloworld.js". The command line at the bottom of the window shows the command "node helloworld.js" being run. The output of the command is displayed above the command line, showing the standard Node.js copyright notice: "Microsoft Windows [Versión 6.1.7601] Copyright © 2009 Microsoft Corporation. Reservados todos los derechos." and the path "C:\Users\Usuario>".

Utilización

Vamos, pues, a realizar nuestra primera aplicación Node.js: "Hola Mundo".

Podemos escribirla con cualquier editor de textos.

Creamos un archivo llamado holamundo.js y lo rellenamos con este código:

```
console.log("Hola Mundo");
```

Salvamos el archivo, y lo ejecutamos a través de Node.js (en consola dentro de nuestro equipo):

```
node holamundo.js
```

Este debería devolver Hola Mundo en tu monitor.

Creación de un servidor HTTP

Aunque más adelante hablaremos de la introducción de módulos en NodeJS, es necesario abordar un primer módulo imprescindible para la creación de un servidor de Http. Este es el módulo *http*.

Para usar el servidor y el cliente HTTP se debe añadir require('http').

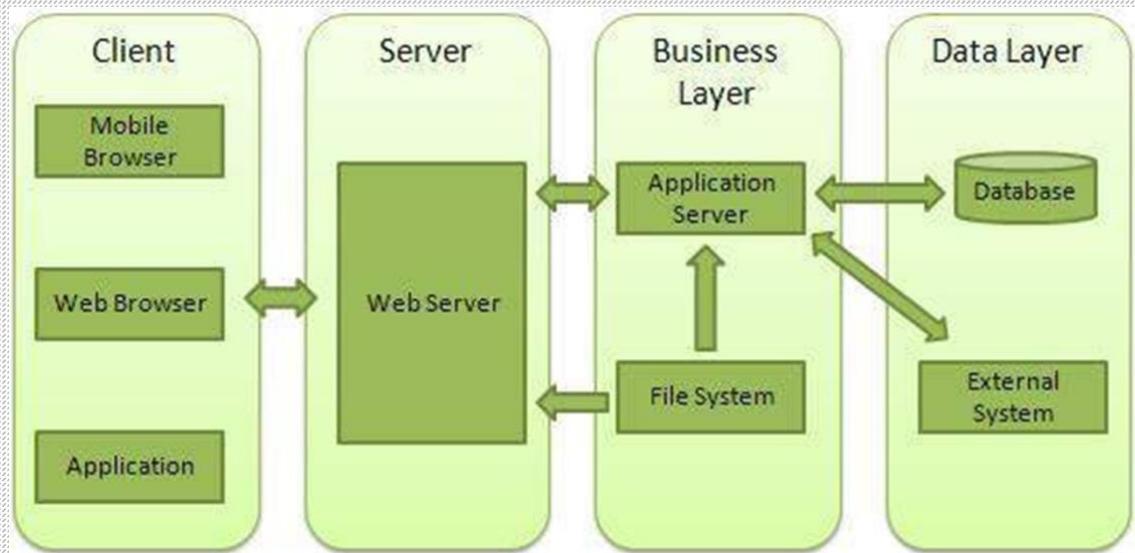
Las interfaces HTTP en Node están diseñadas para soportar muchas de las características del protocolo que tradicionalmente han sido difíciles de usar. En particular, los mensajes grandes, seguramente fragmentado. La interfaz se asegura de que las peticiones o respuestas nunca se almacenen completamente en un búfer--se permite al usuario hacer stream de datos.

Las cabeceras de los mensajes HTTP se representan por un objeto como este:

```
{ 'content-length': '123', 'content-type': 'text/plain', 'connection': 'keep-alive',  
'accept': '*' }
```

Las claves se convierten a minúsculas. Los valores no se modifican.

Para soportar el espectro completo de las posibles aplicaciones HTTP, la API HTTP de Node es de muy bajo nivel. Se encarga únicamente de manejar el stream y del parsing del mensaje. Parsea el mensaje en sus cabeceras y body pero no parsea las cabeceras o el body.



Como hemos comentado, para crear un servidor HTTP en NodeJS necesitamos la carga del módulo http pero, aparte, podemos necesitar la carga de otros módulos de node como el de filesystem y el de url (ambos en el ejemplo inferior). El primero serviría para buscar en el sistema de ficheros la página a devolver con la petición y el segundo para coger la url del envío de nuestro cliente.

```
//Carga de los módulos requeridos para el programa
var http = require('http');
var fs = require('fs');
var url = require('url');
//Creación del servidor
http.createServer(function (request, response) {
    // Cogemos el path que nos ha entrado por la request
    var pathname = url.parse(request.url).pathname;

    // Leemos el fichero requerido para que sea enviado
    fs.readFile(pathname.substr(1), function (err, data) {
        if (err) {
            console.log(err);
            // Página no encontrada
            // HTTP Status: 404 : NOT FOUND
            // Content Type: text/plain
            response.writeHead(404, {'Content-Type': 'text/html'});
        }else{
            // Página encontrada
            // HTTP Status: 200 : OK
            // Content Type: text/plain
            response.writeHead(200, {'Content-Type': 'text/html'});
        }
    });
});
```

```
// Damos la respuesta a nuestra petición
    response.write(data.toString());
}
// Enviamos la respuesta
response.end();
});
}).listen(8081);
```

Una vez tengamos nuestro servidor lanzado vemos que lo hemos puesto a escuchar en el puerto 8081 por lo que un cliente desde el navegador podría acceder a él de esta manera:

<http://127.0.0.1:8081/index.htm>

Esta petición se haría sobre localhost (127.0.0.1) al puerto 8081 y pidiendo la página index.html que se encontraría en los recursos del servidor.

Ejercicio propuesto

Crear un servidor http que, a través de dos rutas en el navegador, nos devuelva lo siguiente:

http://localhost:3000/hola → ¡Hola! (como texto plano)
http://localhost:3000/adios → ¡Adios! (como texto plano)

En otro caso podemos devolver un 404.

Respuesta

```
var http = require("http")
, url = require("url");
var server = http.createServer(),
routes = {
  "/hola": function(req, res) {
    res.end("Hola!");
  },
  "/adios":function(req, res) {
    res.end("Adios!");
  }
};
server.on("request", function(req, res) {
  var urlData = url.parse(req.url),
  path = urlData.pathname;
  if(path in routes) {
    return routes[path](req, res)
  }
});
```

```
    } else {
      res.writeHead(404);
      res.end("Not found!");
    }
  );
server.listen(3000);
```

Actividad práctica 10

Manipulación de peticiones

Para poder realizar la manipulación de peticiones es necesario instalar el módulo de express en nuestra aplicación. Pero, ¿para qué sirve este módulo y que es lo que nos ofrece?

¿Qué es el módulo express?

Express es un módulo de NodeJS que se puede instalar a través de la herramienta *npm*.

En una definición exacta sería: **el framework que se lanza sobre un servidor http de NodeJS** para manipular las rutas y dar acceso de un modo sencillo al cliente para acceder al ciclo de vida de la aplicación.

Instalación

Para instalar dicho módulo solo sería necesario escribir esta línea desde consola de comandos:

```
$ npm install -g express
```

Si escribimos la siguiente línea nos devolverá la versión de express que se ha descargado (en el ejemplo la 3.3.5):

```
$ express --version
```

3.3.5

Ahora que ya tenemos los paquetes que necesitamos, podemos empezar a escribir nuestra aplicación. Creamos un nuevo archivo llamado *app.js* en el directorio del proyecto, este archivo será el que inicie el servidor. Empezamos, escribiendo las dependencias que necesitamos:

```
var express = require('express');
var http = require('http');
```

express es el framework, como ya hemos comentado.

http es el módulo del servidor para NodeJS.

Ahora creamos nuestra aplicación:

```
var app = express();
```

Le indicamos a express en que puerto vamos a estar escuchando:

```
app.set('port', process.env.PORT || 3000);
```

process.env.PORT es una variable de entorno, si no está configurada para guardar el puerto en que debe correr la aplicación, entonces toma el 3000.

Por último creamos e iniciamos el servidor:

```
http.createServer(app).listen(app.get('port'), function(){
  console.log('Express server listening on port ' + app.get('port'));
});
```

Ahora ya tenemos lo mínimo necesario para iniciar la aplicación.

Creando rutas

Las rutas nos permiten direccionar peticiones a los controladores correctos.

Vamos a empezar agregando el código de un controlador para una ruta:

```
app.get('/', function(request, response) {
  response.send('¡Hola, Express!');
});
```

Si corremos nuestra app en la consola (parados en directorio de la aplicación) node app.js y vamos a "http://localhost:3000/" en nuestro explorador de preferencia, debemos ver el mensaje "¡Hola, Express!"

Recibiendo parámetros

Si queremos recibir algún parámetro en una ruta debemos especificar en el String el nombre del parámetro con ":" adelante:

```
app.get('/users/:userName', function(request, response) {
  var name = request.params.userName;
  response.send('¡Hola, ' + name + '!');
});
```

Ahora si corremos la app y vamos a "http://localhost:3000/users/oscar" veremos que se despliega el mensaje "¡Hola, oscar!".

Recibiendo POST

También podemos recibir requests de tipo POST de la siguiente manera:

```
app.post('/users', function(request, response) {  
    var username = request.body.username;  
    response.send('¡Hola, ' + username + '!');  
});
```

Antes de correr este código debemos agregar bodyParser fuera del método, porque express no parsea el cuerpo del request por defecto:

```
app.use(express.bodyParser());
```

Ahora podemos hacerle un post desde cualquier app que nos permita hacerlo.

Se puede utilizar una extensión de Chrome llamada Postman, desde ahí le podemos enviar lo siguiente a "http://localhost:3000/users":

POST /users HTTP/1.1

Host: localhost:3000

Authorization: ApiKey appClient:xxxxxxxxxxxxxxxxxxxxxxxxxxxx

Cache-Control: no-cache

----WebKitFormBoundaryE19zNvXGzXaLvS5C

Content-Disposition: form-data; name="username"

oscar1234

----WebKitFormBoundaryE19zNvXGzXaLvS5C

Y deberá retornar:

¡Hola, oscar1234!

De esta misma manera también podemos recibir requests PUT y DELETE utilizando app.put() y app.delete() respectivamente.

Usando expresiones regulares como ruta

También podemos usar expresiones regulares como rutas, por ejemplo, podríamos usar "/personal/(\d*)/?(edit)?/" como una ruta, especificando así que debe haber un dígito en el medio y que la palabra "edit" es opcional.

```
app.get('/personal/(\d*)/?(edit)?/', function (request, response) {  
    var message = 'el perfil del empleado #' + request.params[0];  
    response.send(message);  
});
```

```
if (request.params[1] === 'edit') {
    message = 'Editando ' + message;
} else {
    message = 'Viendo ' + message;
}
response.send(message);
});
```

Si corremos la app y vamos a "http://localhost:3000/personal/15" veremos que se despliega el mensaje "Viendo el perfil del empleado #15", y si agregamos "/edit" al final veremos que el mensaje cambia a "Editando el perfil del empleado #15".

Luego de todos estos cambios tu archivo app.js debe lucir así:

```
var express = require('express');
var http = require('http');
var app = express();
// all environments
app.set('port', process.env.PORT || 3000);
app.use(express.bodyParser());
app.get('/', function(request, response) {
    response.send(';Hola, Express!');
});
app.post('/users', function(request, response) {
    var username = request.body.username;
    response.send(';Hola, ' + username + '!');
});
app.get(/personal\/(\d*)\?(edit)?/, function (request, response) {
    var message = 'el perfil del empleado #' + request.params[0];
    if (request.params[1] === 'edit') {
        message = 'Editando ' + message;
    }
    response.send(message);
});
```

```
    } else {
        message = 'Viendo ' + message;
    }
    response.send(message);
});

http.createServer(app).listen(app.get('port'), function(){
    console.log('Express server listening on port ' + app.get('port'));
});
});
```

Ejercicio Propuesto

Teniendo este ejemplo de ruta parametrizada que puede ser:

/users/:uid/friends/:friendname

¿Cuáles de las siguientes rutas que escribe nuestro cliente no son válidas para la ruta parametrizada anteriormente vista?

- e) /users/34123/friends/arturo
- f) /users/87345/friends/irene
- g) /users/friends
- h) /users

Respuesta

Son válidos todos excepto los dos últimos. Ni **c)** ni **d)** siguen la misma ruta puesta anteriormente (les faltan los parámetros de *:uid* y *:friendname*).

Módulos principales

NodeJS posee varios módulos compilados en binario. Estos módulos son descritos con más detalle en las siguientes secciones del documento.

Los módulos básicos son definidos en el código fuente de NodeJS en la carpeta *lib/*.

Los módulos básicos tienen la preferencia de cargarse primero si su identificador es pasado desde require(). Por ejemplo, require('http') siempre devolverá lo construido en el módulo HTTP, incluso si hay un fichero con ese nombre.

A continuación vamos a pasar a reseñar algunos de los módulos más importantes que podemos agregar a nuestras aplicaciones de los centenares y centenares que hay. Como nota, se debe comentar, que podemos construirnos nuestros propios módulos dentro de nuestra aplicación, con el objetivo de modularizar nuestro código.

console

Marcado en el API como STDIO, ofrece el objeto console para imprimir mensajes por la salida estándar: stdout y stderr. Los mensajes van desde los habituales info o log hasta trazar la pila de errores con trace.

timers

Ofrece las funciones globales para el manejo de contadores que realizarán la acción especificada pasado el tiempo que se les programa. Debido a la forma como está diseñado Node, relacionado con el bucle de eventos del que se hablará en un futuro, no se puede garantizar que el tiempo de ejecución de dicha acción sea exactamente el marcado, sino uno aproximado cercano a él, cuando el bucle esté en disposición de hacerlo.

module

Proporciona el sistema de módulos según impone CommonJS. Cada módulo que se carga o el propio programa, está modelado según module, que se verá como una variable, module, dentro del mismo módulo. Con ella se tienen disponibles tanto el mecanismo de carga require() como aquellas funciones y variables que exporta, en module.exports, que destacan entre otras menos corrientes que están a un nivel informativo: módulo que ha cargado el actual (module.parent), módulos que carga el actual (module.children)...

buffer

Es el objeto por defecto en Node para el manejo de datos binarios. Sin embargo, la introducción en JavaScript de los typedArrays desplazará a los Buffers como manera de tratar esta clase de datos.

Los módulos siguientes, listados por su identificador, también forman parte del núcleo de Node, aunque no se cargan al inicio, pero se exponen a través del API:

util

Conjunto de utilidades principalmente para saber de si un objeto es de tipo array, error, fecha, expresión regular... También ofrece un mecanismo para extender clases de JavaScript a través de herencia:

inherits(constructor, superConstructor);

events

Provee la fundamental clase EventEmitter de la que cualquier objeto que emite eventos en Node hereda. Si alguna clase del código de un programa debe emitir eventos, ésta tiene que heredar de EventEmitter.

stream

Interfaz abstracta que representa los flujos de caracteres de Unix de la cual muchas clases en Node heredan.

crypto

Algoritmos y capacidades de cifrado para otros módulos y para el código de programa en general.

tls

Comunicaciones cifradas en la capa de transporte con el protocolo TLS/SSL, que proporciona infraestructura de clave pública/privada.

string_decoder

Proporciona una manera de, a partir de un Buffer, obtener cadenas de caracteres codificados en utf-8.

fs

Funciones para trabajar con el sistema de ficheros de la manera que establece el estándar POSIX. Todos los métodos permiten trabajar de forma asíncrona (el programa sigue su curso y Node avisa cuando ha terminado la operación con el

fichero) o síncrona (la ejecución del programa se detiene hasta que se haya completado la operación con el fichero).

path

Operaciones de manejo y transformación de la ruta de archivos y directorios, a nivel de nombre, sin consultar el sistema de ficheros.

net

Creación y manejo asíncrono de servidores y clientes, que implementan la interfaz Stream mencionada antes, sobre el protocolo de transporte TCP.

dgram

Creación y manejo asíncrono de datagramas sobre el protocolo transporte UDP.

dns

Métodos para tratar con el protocolo DNS para la resolución de nombres de dominio de Internet.

http

Interfaz de bajo nivel, ya que sólo maneja los Streams y el paso de mensajes, para la creación y uso de conexiones bajo el protocolo HTTP, tanto del lado del cliente como del servidor. Diseñada para dar soporte hasta a las características más complejas del protocolo como chunk-encoding.

https

Versión del protocolo HTTP sobre conexiones seguras TLS/SSL.

url

Formateo y análisis de los campos de las URL.

querystrings

Utilidades para trabajar con las queries en el protocolo HTTP. Una query son los parámetros que se envían al servidor en las peticiones HTTP. Dependiendo del tipo de petición (GET o POST), pueden formar parte de la URL por lo que deben codificarse o escaparse y concatenarse de una manera especial para que sean interpretadas como tal.

readline

Permite la lectura línea por línea de un Stream, especialmente indicado para el de la entrada estándar (STDIN).

repl

Bucle de lectura y evaluación de la entrada estándar, para incluir en programas que necesiten uno. Es exactamente el mismo módulo que usa Node cuando se inicia sin argumentos, en el modo REPL comentado con anterioridad.

vm

Compilación y ejecución bajo demanda de código.

child_process

Creación de procesos hijos y comunicación y manejo de su entrada, salida y error estándar con ellos de una manera no bloqueante.

assert

Funciones para la escritura de tests unitarios.

tty

Permite ajustar el modo de trabajo de la entrada estándar si ésta es un terminal.

zlib

Compresión/descompresión de Streams con los algoritmos zlib y gzip. Estos formatos se usan, por ejemplo, en el protocolo HTTP para comprimir los datos provenientes del servidor. Es conveniente tener en cuenta que los procesos de compresión y descompresión pueden ser muy costosos en términos de memoria y consumo de CPU.

os

Acceso a información relativa al sistema operativo y recursos hardware sobre los que corre Node.

_debugger

Es el depurador de código que Node tiene incorporado, a través de la opción debug de la línea de comandos. En realidad es un cliente que hace uso de las facilidades de depuración que el intérprete de Javascript que utiliza

Node ofrece a través de una conexión TCP al puerto 5858. Por tanto, no es un módulo que se importe a través de require() sino el modo de ejecución Debug del que se ha hablado antes.

cluster

Creación y gestión de grupos de procesos Node trabajando en red para distribuir la carga en arquitecturas con procesadores multi-core.

punycode

Implementación del algoritmo Punycode, disponible a partir de la versión 0.6.2, para uso del módulo url. El algoritmo Punycode se emplea para convertir de una manera unívoca y reversible cadenas de caracteres Unicode a cadenas de caracteres ASCII con caracteres compatibles en nombres de red.

El propósito es que los nombres de dominio internacionalizados (en inglés, IDNA), aquellos con caracteres propios de un país, se transformen en cadenas soportadas globalmente.

domain

Módulo experimental en fase de desarrollo y, por tanto, no cargado por defecto para evitar problemas, aunque los autores de la plataforma aseguran un impacto mínimo. La idea detrás de este él es la de agrupar múltiples acciones de Entrada/Salida diferentes de tal manera que se dotan de un contexto definido para manejar los errores que puedan derivarse de ellas.

De esta manera el contexto no se pierde e incluso el programa continua su ejecución.

Quedan una serie de librerías, que no se mencionan en la documentación del API pero que existen en el directorio lib/ del código fuente. Estas librerías tienen propósitos auxiliares para el resto de los módulos, aunque se pueden utilizarlas a través de require():

_linklist

Implementa una lista doblemente enlazada. Esta estructura de datos se emplea en timers.js, el módulo que provee funcionalidad de temporización.

Su función es encadenar temporizadores que tengan el mismo tiempo de espera, timeout. Esta es una manera muy eficiente de manejar enormes cantidades de temporizadores que se activan por inactividad, como los timeouts de los sockets, en los que se reinicia el contador si se detecta actividad en él. Cuando esto ocurre, el temporizador, que está situado en la cabeza de la lista, se pone a la cola y se recalcula el tiempo en que debe expirar el primero.

buffer_ieee754

Implementa la lectura y escritura de números en formato de coma flotante según el estándar IEEE754 del IEEE16 que el módulo buffer emplea para las operaciones con Doubles y Floats.

constants

Todas las constantes posibles disponibles de la plataforma como, por ejemplo, las relacionadas con POSIX para señales del sistema operativo y modos de manejo de ficheros. Sólo realiza un binding con node_constants.cc.

freelist

Proporciona una sencilla estructura de pool o conjunto de objetos de la misma clase (de hecho, el constructor de los mismos es un argumento necesario).

Su utilidad se pone de manifiesto en el módulo http, donde se mantiene un conjunto de parsers HTTP reutilizables, que se encargan de procesar las peticiones HTTP que recibe un Servidor.

sys

Es un módulo deprecado, en su lugar se debe emplear el módulo utils.

Ejercicio propuesto

Utilice el módulo *fs* (file system) y el de *stream* para copiar el contenido del fichero *input.txt* a *output.txt*

Respuesta

Inicialmente solo hace falta cargar el módulo de *fs*, ya que el de *stream* es básico. Para leer y escribir en el fichero utilizamos streams con los métodos de *fs*, además de utilizar el método *pipe* de *stream* para copiar el contenido de un stream al otro.

```
var fs = require("fs");

var readerStream = fs.createReadStream('input.txt');

var writerStream = fs.createWriteStream('output.txt');

readerStream.pipe(writerStream);

console.log("Program Ended");
```

Actividad práctica 11 y 12

Combinación de tecnologías en la creación de una página web

A continuación vamos a definir una práctica completa, con todas las tecnologías usadas en este módulo, que aportará el punto de vista global del porqué de las cosas.

Tendremos Node, Angular y JQuery combinados, aparte de mucho JavaScript y el módulo de Express sobre Node, con un ejemplo de conexión a un sistema de MongoDB como sistema de información, conectado a través del módulo de mongoose de Node.

La idea es entrar desde un navegador e ir puliendo los pasos hasta poder realizar una CRUD en nuestro sistema de información.

Diferencia entre "frontend" y "backend"

Las aplicaciones que aúnan código de escritorio y web contienen un número de diferentes elementos, todos trabajando juntos unos con otros.

En la mayoría de los códigos o aplicaciones, los sistemas de información trabajarán junto con el código para dar lógica del sistema, mientras que la interfaz del usuario proveerá el acceso a dicha funcionalidad.

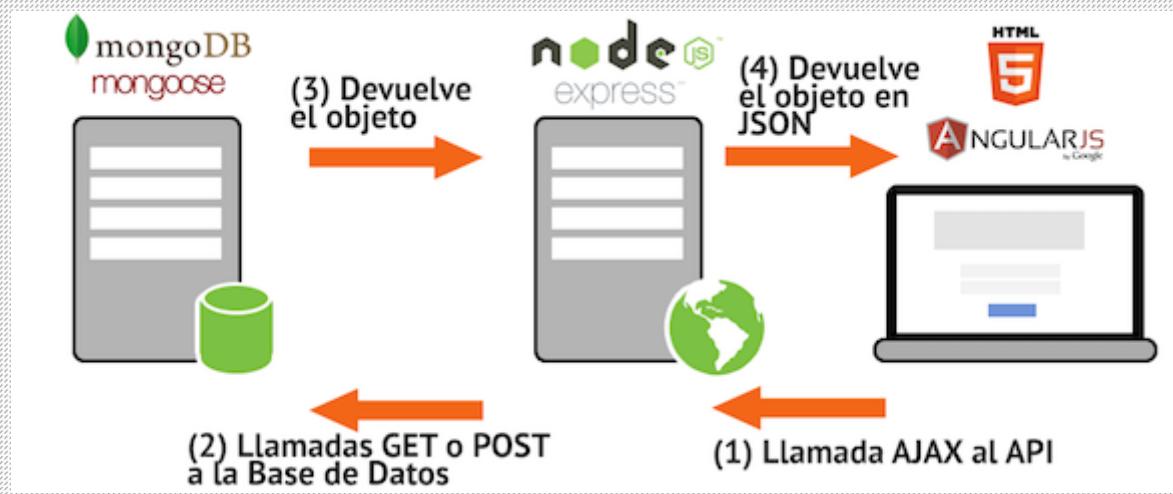
Generalmente, los componentes del "frontend" son aquellos con los que el usuario interactúa, mientras que los componentes del "backend" proveen los datos, servicios y los fundamentos de la lógica de la aplicación.

Algunas tecnologías, sin embargo, pueden usarse para el procesamiento tanto del "frontend" como del "backend".

En los anteriores puntos hemos visto Angular como ejemplo de tecnología "frontend" y Node como "backend".

Arquitectura de la aplicación web SPA + API REST

Primero de todo, veamos un diagrama visual del flujo que va a seguir aplicación con las tecnologías que empleamos en cada parte:



Esquema arquitectura de aplicación web con AngularJS y NodeJS

Desde el frontend, con Angular hacemos llamadas AJAX a nuestra API en el servidor Node. Este consulta a la base de datos (Mongo) dependiendo de la llamada realizada. La BD devuelve el objeto como respuesta a Node y este lo sirve como JSON a Angular que lo muestra en el frontend sin necesidad de recargar la página, creando así una Single Page Application (desde ahora, SPA).

Estructura de archivos

La estructura de archivos va a ser muy sencilla, no vamos a modularizar para que el código nos sea más simple. Estos son los ficheros que tendremos:

- *public*
- *index.html*
- *main.js*
- server.js*
- package.json*

main.js contendrá toda la lógica del frontend, es donde tendremos los controladores de Angular JS y llamaremos vía AJAX al API para pedir contenido, borrarlo, etc...

index.html será nuestro único fichero HTML y por tanto nuestra única página, toda la funcionalidad estará en ella.

server.js es nuestro fichero Node donde estará la configuración del servidor y las rutas a nuestro API.

package.json es el fichero donde están los datos de la aplicación y las dependencias utilizadas, como toda aplicación Node.

Empezaremos por package.json para indicar que dependencias vamos a necesitar, que simplemente serán Express y Mongoose:

```
{  
  "name": "angular-todo",  
  "version": "0.0.1",  
  "description": "Simple Angular TODO app based in MEAN stack",  
  "main": "server.js",  
  "dependencies": {  
    "express": "~3.x",  
    "mongoose": "latest"  
  }  
}
```

Después de esto, en una terminal ejecutamos *npm install* y se nos instalarán las dependencias para poder empezar a utilizarlas.

Implementando nuestro API REST

Ahora pasaremos al archivo server.js que será el fichero donde esté la configuración del servidor, así como la conexión a la base de datos y las rutas de nuestro API.

En los comentarios del código he explicado a grandes rasgos que hace cada línea.

Como todo fichero de servidor de Node, primero añadimos las librerías que necesitamos (express y mongoose).

```
//server.js  
  
var express = require('express');  
  
var app = express();  
  
var mongoose = require('mongoose');  
  
// Conexión con la base de datos  
mongoose.connect('mongodb://localhost:27017/angular-todo');  
  
// Configuración  
app.configure(function() {
```

```
// Localización de los ficheros estáticos
app.use(express.static(__dirname + '/public'));

// Muestra un log de todos los request en la consola
app.use(express.logger('dev'));

// Permite cambiar el HTML con el método POST
app.use(express.bodyParser());

// Simula DELETE y PUT
app.use(express.methodOverride());

});

// Escucha en el puerto 8080 y corre el server
app.listen(8080, function() {
  console.log('App listening on port 8080');
});
```

Creación del modelo de datos

El siguiente paso es construir el modelo de la base de datos que modele las tareas o "ToDos". Esto lo hacemos con Mongoose y nuestro modelo será muy sencillo ya que solo cuenta con un atributo "Text" que define la tarea. Este código lo insertamos en *server.js* antes de la línea donde se inicia el servidor con *app.listen*.

```
// Definición de modelos
var Todo = mongoose.model('Todo', {
  text: String
});
```

Definición de rutas o Endpoints del API

Tras esto nos queda construir las rutas que llamarán a nuestro API y que utilizaremos desde el frontend. En esta tabla se muestran las 3 llamadas que vamos a implementar y que definirán nuestra API:

HTTP	URL	Descripción
GET	/api/todos	Devuelve todas las tareas de la BD
POST	/api/todos	Crea una tarea
DELETE	/api/todos/:todo	Borra una tarea

Veamos las rutas. Estas irán también en el archivo *server.js*, justo antes de cuando se inicia el servidor y escucha en el puerto con *app.listen*.

```
// Rutas de nuestro API

// GET de todos los TODOs
app.get('/api/todos', function(req, res) {
  Todo.find(function(err, todos) {
    if(err) {
      res.send(err);
    }
    res.json(todos);
  });
});

// POST que crea un TODO y devuelve todos tras la creación
app.post('/api/todos', function(req, res) {
  Todo.create({
    text: req.body.text,
    done: false
  }, function(err, todo){
    if(err) {
      res.send(err);
    }
  });
});
```

```
Todo.find(function(err, todos) {
    if(err){
        res.send(err);
    }
    res.json(todos);
});

});

// DELETE un TODO específico y devuelve todos tras borrarlo.

app.delete('/api/todos/:todo', function(req, res) {
    Todo.remove({
        _id: req.params.todo
    }, function(err, todo) {
        if(err){
            res.send(err);
        }
        Todo.find(function(err, todos) {
            if(err){
                res.send(err);
            }
            res.json(todos);
        });
    })
});

// Carga una vista HTML simple donde irá nuestra Single App Page
// Angular Manejará el Frontend

app.get('*', function(req, res) {
    res.sendFile('./public/index.html');
});
```

Gracias a Mongoose podemos buscar(find), borrar(remove) y crear(create) de una manera muy sencilla. La última ruta no corresponde al API, si no que será la encargada de mostrar el html donde ejecutaremos toda la lógica del Frontend.

Desarrollo de la parte Frontend con AngularJS

Todo lo que hemos hecho hasta ahora corresponde al Backend de la aplicación. Ahora empezaremos con lo que de verdad importa, el desarrollo frontend con Angular.

Tendremos toda la lógica en el fichero *main.js*, primero crearemos un módulo que será el que defina toda nuestra aplicación

```
angular.module('angularTodo', []);
```

Y seguidamente la función mainController que será el controlador de la aplicación:

```
function mainController($scope, $http) {
    $scope.formData = {};
    // Cuando se cargue la página, pide del API todos los TODOs
    $http.get('/api/todos')
        .success(function(data) {
            $scope.todos = data;
            console.log(data)
        })
        .error(function(data) {
            console.log('Error: ' + data);
        });
    // Cuando se añade un nuevo TODO, manda el texto a la API
    $scope.createTodo = function(){
        $http.post('/api/todos', $scope.formData)
            .success(function(data) {
                $scope.formData = {};
                $scope.todos = data;
            })
    }
}
```

```
        console.log(data);
    })
    .error(function(data) {
        console.log('Error:' + data);
    });
};

// Borra un TODO despues de checkearlo como acabado
$scope.deleteTodo = function(id) {
    $http.delete('/api/todos/' + id)
        .success(function(data) {
            $scope.todos = data;
            console.log(data);
        })
        .error(function(data) {
            console.log('Error:' + data);
        });
};
}
```

Pasemos a explicar algunos conceptos de esta parte.

En el objeto `$scope` se almacenan todas las variables dentro del ámbito del controlador. En el HTML, todo lo que se encuentre dentro de la directiva `ng-controller="mainController"` es controlable desde el objeto `$scope`.

Y el objeto `$http` es el que hace toda la magia, ya que nos permite hacer llamadas AJAX a nuestro API con pocas líneas de código.

Con estos dos objetos creamos las 3 funciones que hacen las 3 peticiones que aceptan nuestra API, el GET de todas las tareas almacenadas, el POST de creación de una nueva tarea y el DELETE de una tarea.

Y por último nos queda el HTML en el que maquetaremos los resultados que nos trae el API.

Necesitamos indicar que parte de la página corresponde a la aplicación Angular, eso lo hacemos con la directiva *ng-app*. En nuestro caso lo hemos puesto en el tag *<html>* ya que todo el HTML, es la aplicación.

```
<html lang="en" ng-app="angularTodo">...
```

Una aplicación Angular puede tener varios controladores, en este ejemplo solo tenemos uno, el *mainController*, y debemos decir en el HTML que parte es la corresponde a esta función, eso lo hacemos con la directiva *ng-controller*. En nuestro caso la hemos puesto en el *body* porque no hay más. Si tuviésemos más controladores, se pueden poner en otros *section*, *article* o *div* y tener varios en la página.

```
<body ng-controller="mainController">...
```

Para mostrar la lista de tareas que devuelve el GET, utilizaremos la directiva *ng-repeat* que nos permite crear una iteración al estilo de un *for*.

```
<div class="checkbox" ng-repeat="todo in todos">  
  <label>  
    <input type="checkbox" ng-click="deleteTodo(todo._id)"> {{ todo.text }}  
  </label>  
</div>
```

Con esto creamos un input de tipo *checkbox* por cada objeto que nos devuelve la llamada al API. Y con la directiva *ng-click* creamos un evento que escucha cuando marquemos el *checkbox* para llamar a la función *deleteTodo()* a la cual se le pasa como parámetro el id de la tarea para que llame al DELETE del API.

Por último, tenemos un formulario con un input de tipo texto donde escribimos tareas nuevas y las mandamos por POST al API. Aquí usamos una nueva directiva, *ng-model*, que es la que controla el Modelo en este caso la tarea y su texto, y de nuevo *ng-click* en el botón de *submit* para llamar a la función *createTodo()* del controlador que hace el POST al API y a la Base de datos.

```
<form>  
  <div class="form-group">  
    <input type="text" class="form-control input-lg text-center"  
      placeholder="Inserta una tarea nueva" ng-model="formData.text">  
  </div>  
  <button class="btn btn-primary btn-lg" ng-click="createTodo()">
```

Añadir
 </button>
 </form>

Con esto estaría todo. Solo nos queda incluir las librerías de JQuery y Angular como scripts al final de la página, aparte de usar también una hoja de estilos para que no sea tan fea la aplicación.

El código HTML completo sería así:

```
<!doctype html>

<html lang="en" ng-app="angularTodo">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Angular TODO app</title>
    <link rel="stylesheet"
      href="//netdna.bootstrapcdn.com/bootstrap/3.0.3/css/bootstrap.min.css" >
  </head>
  <body ng-controller="mainController">
    <div class="container">
      <!--Cabecera-->
      <div class="jumbotron text-center">
        <h1>Angular TODO List <span class="label label-info">{{ todos.length }}</span></h1>
      </div>
      <!--Lista de Todos-->
      <div id="todo-list" class="row">
        <div class="col-sm-4 col-sm-offset-4">
          <div class="checkbox" ng-repeat="todo in todos">
            <label>
              <input type="checkbox" ng-click="deleteTodo(todo._id)"> {{ todo.text }}
            </label>
          </div>
        </div>
      </div>
    </div>
  </body>
</html>
```

```

        </label>
    </div>
</div>
</div>

<!--Formulario para insertar nuevos Todo-->      <div id="todo-form"
class="row">
    <div class="col-sm-8 col-sm-offset-2 text-center">
        <form>
            <div class="form-group">
                <input type="text" class="form-control input-lg text-center"
placeholder="Inserta una tarea nueva" ng-model="formData.text">
            </div>
                <button class="btn btn-primary btn-lg" ng-
click="createTodo()">Añadir</button>
            </form>
        </div>
    </div>
</div>
</div>
<script
src="//ajax.googleapis.com/ajax/libs/jquery/2.0.3/jquery.min.js"></script> <script
src="//ajax.googleapis.com/ajax/libs/angularjs/1.0.8/angular.min.js"></script>
<script src="main.js"></script>
</body>
</html>

```

Y ya tenemos nuestra aplicación de ToDos. Solo tenemos que correr el servidor en un terminal con `node server.js` e ir a un navegador a la URL `http://localhost:8080` y tendremos algo como esto:

