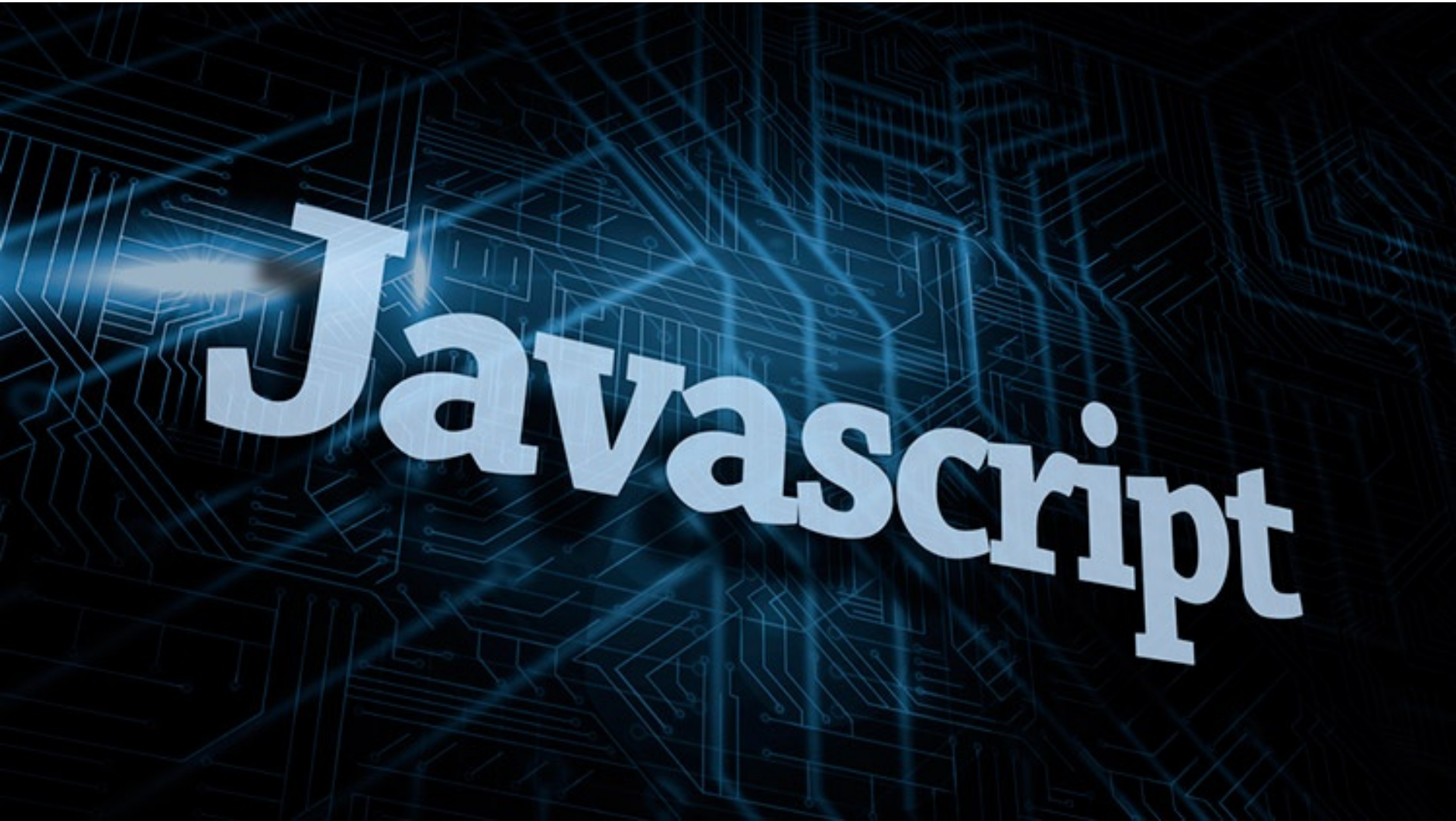
CURSO DESAROLLO WEB CON MEAN

WEB FULL STACK DEVELOPER

Germán Caballero Rodríguez
germanux@gmail.com

Programación avanzada con



INDICE

Excepciones

- Una excepción es un suceso anómalo que ocurre durante la ejecución de un programa e interrumpe el flujo normal de instrucciones.
- La **gestión de excepciones** es el mecanismo que se usa y se recomienda en javascript para el control de errores.

Excepciones

- El **manejo de excepciones** se estructura en tres bloques:
  - **Bloque try**. Instrucciones normales de código. Si salta una excepción se interrumpe el flujo de ejecución y se pasa a ejecutar el bloque **catch**.
  - **Bloque catch**. Instrucciones que tratarán la excepción generada. Un bloque catch dispone siempre de una referencia a la excepción generada.
  - **Bloque finally**. Instrucciones que se ejecutarán siempre después del bloque **try-catch**, se haya producido una excepción o no.

Excepciones

- Un ejemplo de construcción de un bloque **try-catch-finally**

```
try{
	//debe contener el código a probar
} catch(exception){
	//código a ejecutar en caso de fallo
} finally{
	//se ejecuta siempre, pase lo que pase en try o catch
}
```

Excepciones

- Después de un **try** debe haber obligatoriamente un **catch** o un **finally**
- Aunque se ejecute un **return** dentro del **try**, se sigue ejecutando el **finally**.
- Se permite el lanzamiento de excepciones manualmente

```
throw new Error('esto es un error!');
```

Consola

- Los navegadores, disponen de un consola, sobre la cual se pueden escribir mensajes de log, a parte de ejecutar código javascript.
- Para escribir mensajes
  - **console.log**() para enviar y registrar mensajes generales;
  - **console.dir**() para registrar un objeto y visualizar sus propiedades;
  - **console.warn**() para registrar mensajes de alerta;
  - **console.error**() para registrar mensajes de error.

Objetos

- Definidos para organizar el código y encapsular los métodos y funciones
- Array asociativo formado por métodos y propiedades del objeto
- Un array asociativo es un array donde las claves no son numéricas, sino que son textos.

```
var arrayAsociativo = new Array();
arrayAsociativo['uno'] = 1;
arrayAsociativo['dos'] = 2;
arrayAsociativo['tres'] = 3;
```

Objetos

- Como es un array asociativo, en caso de intentar acceder por posición, el resultado será **undefined**.
- Se puede acceder vía notación de puntos

```
arrayasociativo.uno = 12;
```

- Para asignar un valor a una propiedad, no es necesario que la propiedad haya sido definida.

Objetos

- Podemos asignar métodos mediante la notación de puntos

```
arrayasociativo.getuno = function() {
        this.uno;
}
```

- Se puede usar la palabra reservada **this** para acceder a propiedades del objeto.
- Se puede asociar a una función declarada con anterioridad

```
arrayasociativo.nuevafuncion = otrafuncionyacreada();
```

Objetos

- Podemos almacenar objetos dentro de objetos

```
var trabajador = new Object();

trabajador.nombre = “Victor”;
trabajador.apellido = “Herrero”;

trabajador.getNombreCompleto = new function() {
      return this.nombre + this.apellido;
}

trabajador.direccion = new Object();

trabajador.direccion.calle = ‘Mayor’;
trabajador.direccion.numero = 1;
```

Objetos

- **call**() Ejecuta una función como si fuera un método de otro objeto.

```
function multiplica(x){
        return this.valor*x
}
var unobjetonuevo = new object();
unobjetonuevo.valor=7;
multiplica.call(unobjetonuevo,2);
```

- **apply**() como call, pero admite un array como lista de parámetros

Json

- javascript object notation.
- Formato sencillo para intercambio de información
- Alternativa al formato xml
- XML sigue siendo muy superior
- Tipo mime **application/json**
- Permite generar estructuras complejas anidadas de una sola vez
- Se puede usar notación tradicional y notación json a la vez.

Json

- Con json, la creación de arrays es mas sencilla
- La creación de arrays asociativos también es mas sencilla
- Los arrays se definen dentro de [y]

```
var prueba = [“uno”,”dos”,”tres”];
```

- Los objetos están contenidos dentro de { y }
- Los elementos se separan con ,
- La separación entre la clave y el valor es con :

```
var pruebaasociativa = {uno:1,dos:2,tres:3}
```

Json

- Para transformar un String a JSON, se puede emplear

```
var texto = “{‘nombre’: ‘Victor’, ‘apellido’: ‘Herrero’}”;
var objeto = JSON.parse(texto);
```

- O en versiones antiguas

```
var texto = “{‘nombre’: ‘Victor’, ‘apellido’: ‘Herrero’}”;
var objeto = eval(‘(’ + texto + ‘)’);
```

- Esta ultima forma, ahora desaconsejada, ya que es menos segura, ya que no solo interpreta texto JSON, sino que también interpreta código javascript.

Json

- La operación inversa, la que permite obtener un String en formato JSON a partir de un objeto javascript

```
Var string = JSON.stringify({nombre: 'Victor', apellido : 'Herrero'});
```

Clases

- Las **Clases** son el concepto base de la orientación a objetos, permite definir la plantilla de los objetos.
- Se ha visto, como se pueden definir en javascript **Objetos** “al vuelo”, pero esto supone un problema, y este es, que cuando se quiera otro objeto con las mismas propiedades y métodos, hay que volver a realizar la definición, se necesita pues la **Clase** (plantilla).
- **Javascript** no permite el uso de la palabra **class**, solo para futuras versiones.

Clases

- Por tanto se puede decir que en **javascript** no existe el concepto de **Clase** que tienen otros lenguajes como java, sino el de **pseudoclases**.
- Estas **pseudoclases**, se basan en dos conceptos
  - **Funciones constructoras.**
  - **Prototype de los objetos.**

Clases

- **Funciones constructoras**
  - En **javascript** no existen los constructores
  - Se emulan por medio de funciones
  - Al crear el nuevo objeto

```
new Persona(1, 'Victor');
```

  - Realmente se llama a la función constructora

```
function Persona(nif, nombre){
    this.nombre = nombre;
    this.nif = nif;
}
```

  - Las funciones constructoras pueden definir tanto propiedades, como métodos.

Clases

- Las funciones constructoras pueden definir tanto propiedades, como métodos.
- Cualquier función de javascript se puede tratar como una clase en el sentido de que se puede instanciar (hacer un new de ella).

Clases

- En realidad, el operador **new**, lo que hace es reservar espacio nuevo en memoria para un objeto.
- En este caso el objeto es la función constructora.
- Como **this** hace referencia al objeto (la función), se están añadiendo por tanto al objeto **características** (atributos de clase) como **nombre** y **nif**.
- Esta practica no es conveniente con los métodos, ya que cada objeto tendría el suyo propio.

Clases

- Lo que esa función defina con this., la clase lo tendrá.

```
function UnaClase () {
    this.unParametro = 3;
    this.unaFuncion = function() {
        return("hola");
    }
}

a = new UnaClase;
alert(a.unParametro);
alert(a.unaFuncion());
```

Clases

- Para un constructor con parámetros, basta con ponerlos en la función.

```
function UnaClase (par) {
    this.unParametro = par;
    this.unaFuncion = function() {
        return(this.unParametro);
    }
}

a = new UnaClase(11);
alert(a.unParametro);
alert(a.unaFuncion());
```

Clases

- Y dado que se tiene **this** para hacer referencia al objeto concreto, se podría emplear el mismo método para todos los objetos de una tipología, ahorrando así memoria.
- Por lo tanto la función constructora no es suficiente.
- La opción para definir los métodos, y que estos sean únicos para cada objeto de una clase será definirlos sobre el **prototype**.

Clases: Callbacks

- Cabe destacar al hablar de clases, la particularidad que se produce cuando se emplea un **método** de un **objeto** como **callback** de un **evento**.
- En este caso, se produce una circunstancia inesperada en relación a **this**.

Clases: Callbacks

- Pongamos un ejemplo, se define una tipología.

```
Gestor = function (callback){
        var btEnviar = document.getElementById('btEnviar');
        btEnviar.onclick = callback;
}
```

- En el ejemplo se ve como la función constructora recibe por parámetro la referencia a una función (callback), que asigna como **handler** del evento **onclick** de un botón.

Clases: Callbacks

- Ahora vamos a definir otra tipologia con un método que será empleado como **callback**.

```
Manejador = function(dato){
        this.dato = dato;
}
Manejador.prototype.miCallback = function (){
        alert(this.dato);
}
```

- En este método, se hace referencia a **this**, que debería apuntar al objeto propietario del método **miCallback**, que debería ser un objeto **Manejador**, digo debería, ya que no lo va a ser.

Clases: Callbacks

- Solo queda instanciar y asignar.

```
var manejador = new Manejador("el dato del manejador");
var gestor = new Gestor(manejador.miCallback);
```

- Cuando se produzca el **click** sobre el **btEnviar**, querremos que se ejecute el método **miCallback** asociado a la instancia de **Manejador** que tiene como **dato**, "el dato del manejador", que deberá ser mostrado por pantalla en un **alert**.

Clases: Callbacks

- La realidad es que el anterior código indicará que **this** no tiene una propiedad dato, ya que **this** en ese caso hace referencia a **quien lanza el evento**, que es el **propietario real** del método y no al **teórico propietario** del método, que sería **Manejador**.
- El **propietario del método es quien lanza el evento**, ya que así lo indicamos cuando hacemos la asignación del **callback**.

```
btEnviar.onclick = callback;
```

Clases: Callbacks

- ¿Entonces como se puede obtener la referencia al objeto que es propietario del método inicialmente?
- Pues empleando un método que se proporciona de forma nativa en el prototype llamado **bind**().

```
var gestor = new Gestor(manejador.miCallback).bind(manejador);
```

Clases

- **Herencia**
  - Es posible establecer pseudoherencia entre pseudoclases, para ello hay que hacer que el prototype de las clases hijas, sea un objeto de la clase padre.

```
function Trabajador(codigoEmpleado){
      this. codigoEmpleado = codigoEmpleado;
}

Trabajador.prototype = new Persona;
```

Clases

- **Herencia**
  - Si el Padre tiene una función constructora con parámetros, para inicializar dichos parámetros, se invocará a dicha función constructora desde la del Hijo, empleando **call**, ya que no se tiene referencia a Padre, no existe **super**.

```
function Trabajador(nif, nombre, codigoEmpleado){
        call(this,nif,nombre);
        this. codigoEmpleado = codigoEmpleado;
}

Trabajador.prototype = new Persona;
```

Clases

- **Prototype**
  - Es el molde con el que se crean los objetos.
  - Si se modifica el prototipo, todos los objetos son modificados.
  - Los prototipos también tienen un problema, se van al otro extremo, todos los objetos comparten las mismas referencias, que para los métodos es deseable, pero para los atributos no, no se desea modificar el atributo de un objeto desde otro.

```
Persona.prototype.getNombre = function(){
    return this.nombre;
}
```

Clases

- **Prototype**
- Todas las clases javascript tienen una propiedad que se llama prototype que de alguna forma, es la clase padre por defecto.
- Para hacer que una clase herede de otra, hay que cambiar el prototype de la clase hija para que sea una instancia de la padre.

```
Persona.prototype.getNombre = function(){
    return this.nombre;
}
```

Clases

- **Prototype**

```
function Padre() {
    this.atributoPadre=3;
}

function Hijo() {
    this.atributoHijo=4;
}
Hijo.prototype = new Padre;

a = new Hijo;
alert (a.atributoPadre);
alert (a.atributoHijo);
```

Clases

Constructores padre con parámetros

- Si el constructor de la clase padre tiene parámetros, podemos llamarlo desde la clase hija usando la funcion call() que tienen definidas las clases por defecto

Clases

Constructores padre con parámetros

-

```
function Padre(attPadre) {
    this.atributoPadre=attPadre;
}

function Hijo(attHijo, attPadre) {
    Padre.call(this,attPadre);
    this.atributoHijo=attHijo;
}
Hijo.prototype = new Padre;

a = new Hijo(2,3);
alert (a.atributoPadre);
alert (a.atributoHijo);
```

Clases

- **Sobrescritura**
  - Se pueden modificar/crear propiedades y métodos de objetos predefinidos.

```
Array.prototype.indexof = function(…){
        …
}
```

  - Esta es la nueva función de la clase Array que permite buscar la posición de un objeto en un Array.

Clases

- **Paquetes**
- Las funciones tal cual las hemos declarado quedan como variables globales en la página y puede haber problemas de nombres en un proyecto grande con mucho código javascript.
- Una solución es intentar agruparlas.

Clases

- **Paquetes**
  - Dado que en **javascript** no existe un concepto similar al **paquete** de Java o el **namespace** de .NET, si se quiere organizar el código, habrá, de nuevo, que simular dicho comportamiento.
  - Para ello se definirán objetos asociados a variables globales, que desempeñaran esta misión.

```
com.ejemplo.javascript.entidades = {
        function Trabajador(nif, nombre, codigoEmpleado){
                call(this,nif,nombre);
                this. codigoEmpleado = codigoEmpleado;
        }
}
```

Clases

- **Paquetes**

```
paquete = {

   Padre : function (attPadre) {
      this.atributoPadre=attPadre;
   },

   Hijo : function (attHijo, attPadre) {
        paquete.Padre.call(this,attPadre);
        this.atributoHijo=attHijo;
   }
}
paquete.Hijo.prototype = new paquete.Padre;

a = new paquete.Hijo(2,3);
alert (a.atributoPadre);
alert (a.atributoHijo);
```

Clases

Subpaquetes

- De esta forma, lo único global será "paquete", que podría ser, por ejemplo, el nombre de nuestro proyecto/librería.
- Podemos también añadir los niveles que queramos y agrupar como más nos guste.

Clases

Subpaquetes

```
paquete = {

    Padre : function (attPadre) {
        this.atributoPadre=attPadre;
    },

    subpaquete : {
        Hijo : function (attHijo, attPadre) {
            paquete.Padre.call(this,attPadre);
            this.atributoHijo=attHijo;
        }
    }
}
paquete.subpaquete.Hijo.prototype = new paquete.Padre;

a = new paquete.subpaquete.Hijo(2,3);
alert (a.atributoPadre);
alert (a.atributoHijo);
```

Clases

- **Ambito**
  - **Javascript** no posee ámbitos (**public, private, protected**), como otros lenguajes de programación.
  - Lo único que se puede hacer es simular de nuevo, en este caso solamente la visibilidad **private**.
  - La forma será definiendo variables locales a la función constructora.

```
function Persona(nif, nombre){
        var _nif = nif;
        var _nombre = nombre;
        var procesar = function(parametro) {
                Console.log(parametro);
        }
}
```

Clases

- **Ambito**
  - Aunque se ha conseguido que los atributos y el método sean privados, existe un problema, y es que para acceder a ellos, solo se puede hacer desde la función constructora por lo que los Getter y Setter se han de definir en la función constructora y no en el prototype.

```
function Persona(nif, nombre){
	var _nif = nif;
	this.getNif = function(){
		return _nif;
	}
}
```

Modelo de Objeto de Documento (DOM3)

- Conjunto de utilidades específicamente diseñadas para manipular documentos XML.
- Por extensión, DOM también se puede utilizar para manipular documentos XHTML y HTML.
- DOM transforma el código XML en una serie de nodos interconectados en forma de árbol.
- DOM está disponible en la mayoría de lenguajes de programación comúnmente empleados.

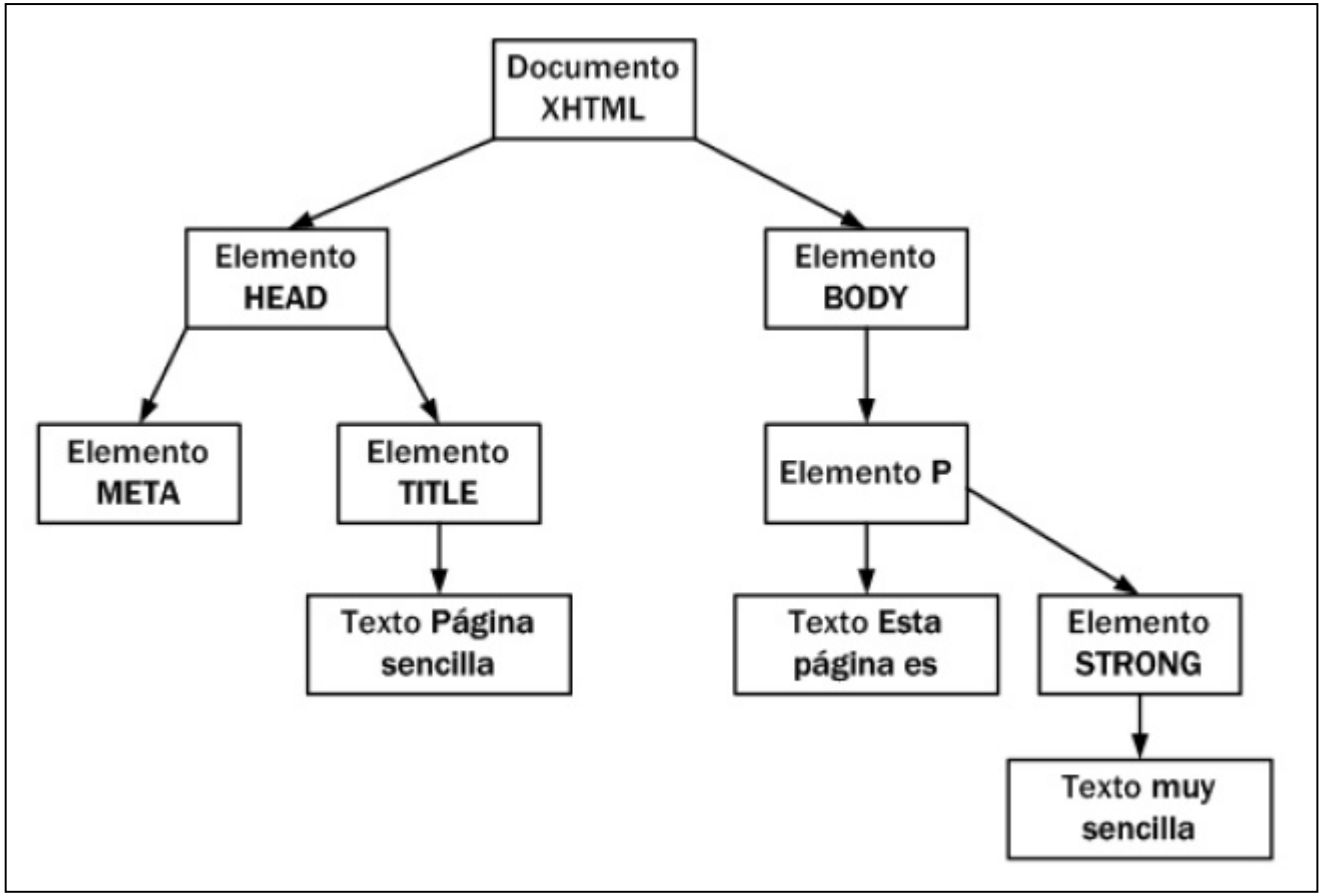
Modelo de Objeto de Documento (DOM3)

- Dado el siguiente código:

```
<!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>
```

Modelo de Objeto de Documento (DOM3)

- El navegador al aplicar DOM obtiene



Modelo de Objeto de Documento (DOM3)

- Para utilizar DOM es imprescindible que la página web se haya cargado por completo.
- DOM permiten añadir, eliminar, modificar y remplazar cualquier nodo.

Modelo de Objeto de Documento (DOM3)

- **Tipos de Nodos**
  - **Document**: nodo raíz de todos los documentos HTML y XML.
  - **DocumentType**: nodo que contiene el DTD.
  - **Element**: contenido definido por un par de etiquetas (<etiqueta>...</etiqueta>) o una etiqueta abreviada (<etiqueta/>). Es el único nodo que puede tener tanto nodos hijos como atributos.
  - **Attr**: representa el par nombre-de-atributo/valor.

Modelo de Objeto de Documento (DOM3)

- **Tipos de Nodos**
  - **Text**: texto que se encuentra entre una etiqueta de apertura y una de cierre. También almacena el contenido de una sección de tipo CDATA.
  - **CDataSection**: es el nodo que representa una sección de tipo <![CDATA[ ]]>.
  - **Comment**: representa un comentario.

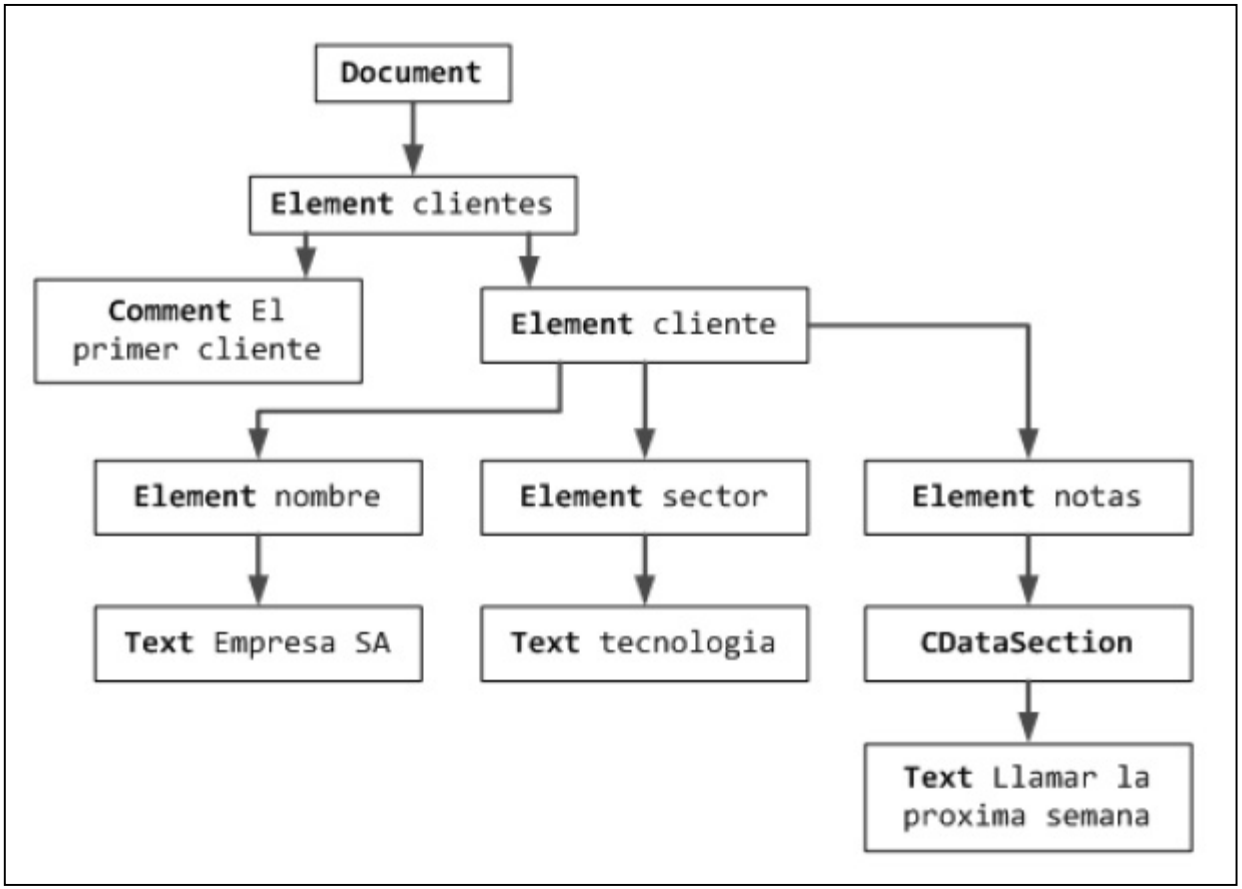
Modelo de Objeto de Documento (DOM3)

- Dado el siguiente código:

```
<?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>
```

Modelo de Objeto de Documento (DOM3)

- Su árbol DOM asociado



Modelo de Objeto de Documento (DOM3)

- **La interfaz Node**
  - JavaScript crea el objeto ***Node*** para definir las propiedades y métodos necesarios para procesar y manipular los documentos.
  - Este objeto tiene una serie de constantes y funciones que facilitan su manipulación.

Modelo de Objeto de Documento (DOM3)

- **La interfaz Node (Constantes)**
  - Node.ELEMENT_NODE = 1
  - Node.ATTRIBUTE_NODE = 2
  - Node.TEXT_NODE = 3
  - Node.CDATA_SECTION_NODE = 4
  - Node.ENTITY_REFERENCE_NODE = 5
  - Node.ENTITY_NODE = 6
  - Node.PROCESSING_INSTRUCTION_NODE = 7

Modelo de Objeto de Documento (DOM3)

- **La interfaz Node (Constantes)**
  - Node.COMMENT_NODE = 8
  - Node.DOCUMENT_NODE = 9
  - Node.DOCUMENT_TYPE_NODE = 10
  - Node.DOCUMENT_FRAGMENT_NODE = 11
  - Node.NOTATION_NODE = 12

Modelo de Objeto de Documento (DOM3)

- **La interfaz Node (Funciones)**

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

Modelo de Objeto de Documento (DOM3)

- El uso de DOM, esta supeditado al navegador donde se ejecute, y a la versión de DOM que implemente.
- Navegadores como Firefox y Safari implementan DOM de nivel 1 y 2 (y parte del 3), otros navegadores como Internet Explorer (versión 7 y anteriores) ni siquiera son capaces de ofrecer una implementación completa de DOM nivel 1.

Modelo de Objeto de Documento (DOM3)

- En la especificación de DOM para HTML, el nodo raíz de todos los demás se define en el objeto ***HTMLDocument***. Además, se crean objetos de tipo ***HTMLElement*** por cada nodo de tipo ***Element*** del árbol DOM.

Modelo de Objeto de Documento (DOM3)

- **Atributos**
  - Los nodos de tipo ***Element*** contienen la propiedad ***attributes***, que permite acceder a todos los atributos de cada elemento. Aunque técnicamente es de tipo ***NamedNodeMap***, sus elementos se pueden acceder como un ***array***.

Modelo de Objeto de Documento (DOM3)

- **Atributos**
  - Los atributos de los Nodos HTML, se pueden acceder siguiendo DOM.

```
nodo.getAttribute(“src");
```

  - O de forma mas sencilla y directa con la especificación de DOM para HTML.

```
nodo.src;
```

  - Se recomienda la ultima, ya que IE no implementa correctamente ***setAttribute***().
  - La única excepción es el atributo ***class***. Dado que ***class*** está reservada se accede con ***className***.

Modelo de Objeto de Documento (DOM3)

- **Funciones**
  - **getNamedItem(nombre)**, devuelve el nodo cuya propiedad ***nodeName*** contenga el valor ***nombre***.
  - **removeNamedItem(nombre)**, elimina el nodo con ***nodeName*** igual a ***nombre***.
  - **setNamedItem(nodo)**, añade el nodo a la lista ***attributes***, indexándolo según su ***nodeName***.
  - **item(posicion)**, devuelve el nodo en la posición indicada por el valor numérico ***posicion***.
  - **getAttribute(nombre)**, es equivalente a attributes.getNamedItem(nombre).

Modelo de Objeto de Documento (DOM3)

- **Funciones**
  - **setAttribute(nombre**, **valor**) equivalente a attributes.getNamedItem(nombre).value = valor.
  - **removeAttribute(nombre**), equivalente a attributes.removeNamedItem(nombre).
  - **getElementsByTagName**(), obtiene todos los elementos de la página cuya etiqueta sea igual que el parámetro que se le pasa a la función. La función devuelve un NodeList, similar a un array e Nodos.
  - **getElementsByName**() obtiene todos los elementos de la página cuyo atributo ***name*** coincida con el parámetro que se le pasa a la función.

Modelo de Objeto de Documento (DOM3)

- **Funciones**
  - **getElementById**() devuelve el elemento cuyo atributo ***id*** coincide con el parámetro indicado en la función.
  - **createAttribute(nombre**) Crea un nodo de tipo atributo con el nombre indicado
  - **createCDataSection(texto**) Crea una sección CDATA con un nodo hijo de tipo texto que contiene el valor indicado
  - **createComment(texto**) Crea un nodo de tipo comentario que contiene el valor indicado
  - **createDocumentFragment**() Crea un nodo de tipo DocumentFragment

Modelo de Objeto de Documento (DOM3)

- **Funciones**
  - **createElement(nombre_etiqueta**) Crea un elemento del tipo indicado en el parámetro **nombre_etiqueta**
  - **createEntityReference(nombre**) Crea un nodo de tipo EntityReference
  - **createProcessingInstruction(objetivo**, **datos**) Crea un nodo de tipo ProcessingInstruction
  - **createTextNode(texto**) Crea un nodo de tipo texto con el valor indicado como parámetro
  - **removeChild(nodo**), elimina un objeto nodo, dentro del nodo padre sobre el que se invoca la función.

Modelo de Objeto de Documento (DOM3)

- **Funciones**
  - **replaceChild(nuevoP**, **anteriorP**), remplaza un nodo por otro, dentro del nodo padre sobre el que se invoca la función.
  - **appendChild(nodo**), inserta un nuevo nodo en la ultima posición del listado de nodos hijos, dentro del nodo padre sobre el que se invoca la función.
  - **insertBefore(nuevoP**, **anteriorP**), inserta un nuevo nodo ***nuevoP*** antes del nodo ***anteriorP***

Ejercicio

- A partir de la página web proporcionada y utilizando las funciones DOM, mostrar por pantalla la siguiente información:
  1. Número de enlaces de la página
  2. Dirección a la que enlaza el penúltimo enlace
  3. Numero de enlaces que enlazan a http://prueba
  4. Número de enlaces del tercer párrafo

Ejercicio

- La aplicación cuenta con tres cajas vacías y dos botones. Al presionar el botón de "Genera", se crean dos números aleatorios. Cada número aleatorio se guarda en un elemento <p>, que a su vez se guarda en una de las dos cajas superiores.
- Una vez generados los números, se presiona el botón "Comparar", que compara el valor de los dos párrafos anteriores y determina cual es el mayor.

Ejercicio

- El párrafo con el número más grande, se mueve a la última caja que se utiliza para almacenar el resultado de la operación.
- En el ejercicio se deben utilizar entre otras, las funciones getElementById(), createElement(), createTextNode(), appendChild(), replaceChild() y Math.random().

Modelo de Objeto de Documento (DOM3)

- **DOM y CSS**
  - El acceso a las propiedades CSS no es tan directo y sencillo como el acceso a los atributos HTML.
  - Los estilos CSS se pueden aplicar
    - Directamente sobre el elemento con el atributo ***style***.
    - Mediante una clase CSS.
  - Como se accede a los atributos CSS depende del navegador.
    - Con IE, se hace uso de la propiedad **currentStyle**.
    - Con otros la función **getComputedStyle**().

Modelo de Objeto de Documento (DOM3)

- **DOM y CSS**
  - La propiedad ***currentStyle*** requiere el nombre de las propiedades CSS según el formato de JavaScript (sin guiones medios),
  - La función ***getPropertyValue***() exige el uso del nombre original de la propiedad CSS.

Modelo de Objeto de Documento (DOM3)

- **DOM y CSS (Ejemplo)**

```
// Código HTML
      <p id="parrafo">...</p>
// Regla CSS
      #parrafo { color: #008000; }
// Código JavaScript para Internet Explorer
      var parrafo = document.getElementById("parrafo");
      var color = parrafo.currentStyle['color'];
// Código JavaScript para otros navegadores
      var parrafo = document.getElementById("parrafo");
      var color =document.defaultView.getComputedStyle(
            parrafo, '').getPropertyValue('color');
```

Modelo de Objeto de Documento (DOM3)

- **DOM y Tablas HTML**
  - DOM para HTML incluye varias propiedades y métodos para crear tablas, filas y columnas de forma sencilla.
  - Propiedades y métodos de <table>

Propiedad/Método	Descripción
`rows`	Devuelve un array con las filas de la tabla
`tBodies`	Devuelve un array con todos los `<tbody>` de la tabla
`insertRow(posicion)`	Inserta una nueva fila en la posición indicada dentro del array de filas de la tabla
`deleteRow(posicion)`	Elimina la fila de la posición indicada

Modelo de Objeto de Documento (DOM3)

- **DOM y Tablas HTML**
  - Propiedades y métodos de <tbody>

Propiedad/Método	Descripción
rows	Devuelve un array con las filas del <tbody> seleccionado
insertRow(posicion)	Inserta una nueva fila en la posición indicada dentro del array de filas del <tbody>
deleteRow(posicion)	Elimina la fila de la posición indicada

  - Pr

Propiedad/Método	Descripción
cells	Devuelve un array con las columnas de la fila seleccionada
insertCell(posicion)	Inserta una nueva columna en la posición indicada dentro del array de columnas de la fila
deleteCell(posicion)	Elimina la columna de la posición indicada

Modelo de eventos

- El modelo de eventos de javascript, se basa en la capacidad que tienen los elementos HTML de lanzar eventos, y en la posibilidad que tiene el programador de asignar funciones javascript que responden, se ejecutan, cuando se producen los eventos, estas funciones, son llamadas **handler** o **listener**.

Modelo de eventos

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

Modelo de eventos

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>

Modelo de eventos

- Los manejadores de los eventos, se pueden definir de dos formas
  - Declarativa en las etiquetas HTML.

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

  - Programática, en funciones javascript.

```
<input id="pinchable" type="button" value="Pinchame y verás" />

Pinchable.onclick = muestraMensaje;

function muestraMensaje() { alert('Gracias por pinchar'); }
```

Modelo de eventos

- Dentro de la propiedad que permite definir el manejador de un evento en una etiqueta HTML, se puede emplear la palabra reservada **this**, para hacer referencia al elemento HTML que genero el evento.

```
<div onmouseover="this.style.borderColor='black';"></div>
```

- **Ojo**, si se emplea un método de un objeto como handler porque this dentro de dicho método significa cosas distintas cuando se emplea como método y cuando se emplea como handler

Modelo de eventos

- Ojo que cuando se emplea la programática, se ha de haber cargado por completo el árbol de elementos DOM, antes de poder realizar el registro de los **handler**. Para ello, se puede definir un handler para este evento

```
window.onload = function() { }
```

Modelo de eventos

- El método **addEventListener**, es la otra vía para registrar **handler** en los objetos **javascript**.
- Este método acepta tres parámetros.
  - **EventType**: Tipo evento
  - **Listener**: Función de **listener**, que recibe como parámetro el evento, y que ha de tratarlo.
  - **UseCapture**: Booleano que indica si se quiere propagar el evento hacia arriba en el árbol de nodos. (normalmente se establece a false).

Modelo de eventos

- Para IE6, 7 y 8, el API para registrar los eventos es distinto, se hace con la función **attachEvent**()

```
window.attachEvent('onstorage', function(event){});
```

Modelo de eventos

- Dentro de los **handler** se puede obtener información del evento.
- Para ello se tiene un objeto evento, que se obtiene de forma dispar, dependiendo del navegador
  - A través de **window.event** en IE.
  - Y como parámetro del **handler** para el resto.

```
function manejadorEventos(elEvento) {
      var evento = elEvento || window.event;
}
```

Modelo de eventos

- Para los eventos de teclas llega diferente información dependiendo de la tecla
- Evento **keydown** y **keyup**: Mismo comportamiento en todos los navegadores:
  - **keyCode**: código interno de la tecla
  - **charCode**: no definido

Modelo de eventos

- Evento **keypress**:
  - Internet Explorer:
    - **keyCode**: código del carácter de la tecla pulsada
    - **charCode**: no definido
  - Resto de navegadores:
    - **keyCode**: para las teclas normales, no definido. Para las teclas especiales, código interno de la tecla.
    - **charCode**: para las teclas normales, el código del carácter de la tecla que se ha pulsado. Para las teclas especiales, 0.

Modelo de eventos

- Para convertir el código de un carácter (no confundir con el código interno) al carácter que representa la tecla que se ha pulsado, se utiliza la función **String.fromCharCode()**.
- Ejemplo de función para obtener en cualquier navegador la tecla pulsada

```
function manejador(elEvento) {
	var evento = elEvento || window.event;
	var caracter = evento.charCode || evento.keyCode;

	alert("El carácter es: " + String.fromCharCode(caracter));
}
```

Modelo de eventos

- También se puede conocer si se pulsan teclas especiales
  - **altKey**: booleano que indica si estaba pulsada la tecla **Alt**.
  - **ctrlKey**: booleano que indica si estaba pulsada la tecla **Ctrl**.
  - **shiftKey**: booleano que indica si estaba pulsada la tecla **May**.

Modelo de eventos

- Cuando los eventos son de ratón, interesan otras propiedades.
  - **clientX** y **clientY**: Coordenadas respecto a la ventana del navegador.
  - **screenX** y **screenY**: Coordenadas respecto a la pantalla del ordenador.

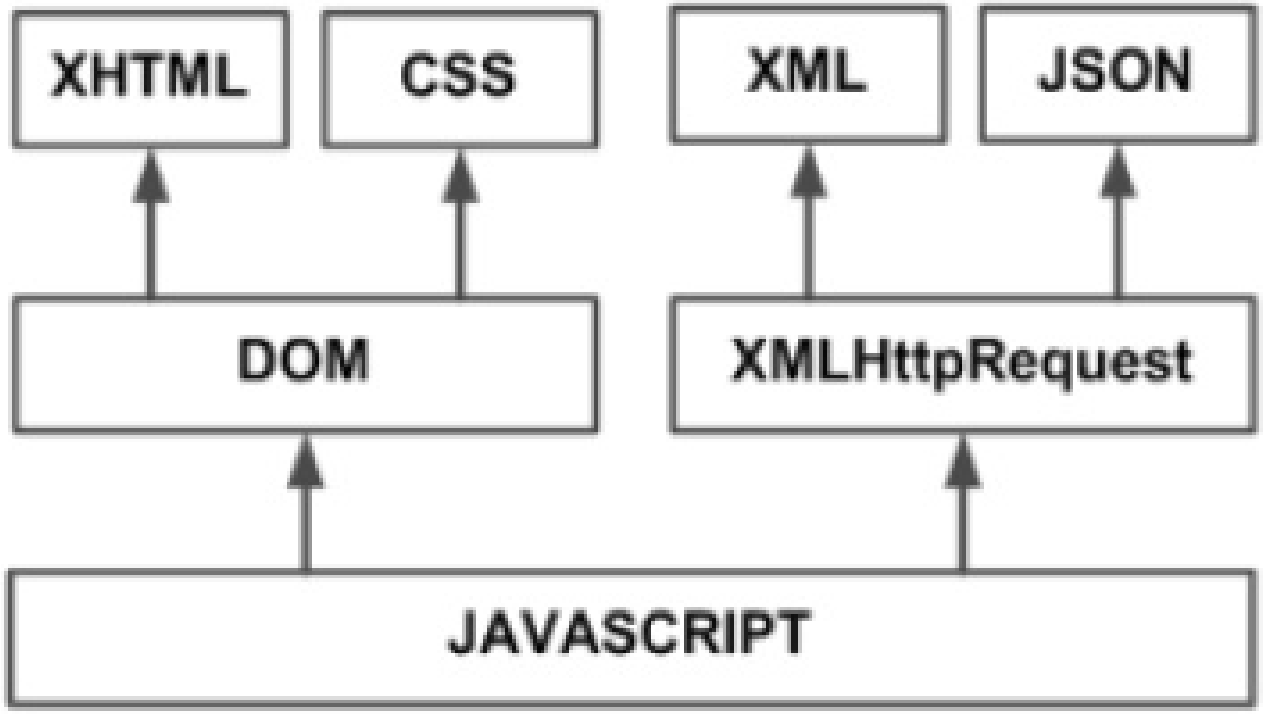
Introducción a AJAX

- El término AJAX es un acrónimo de Asynchronous JavaScript + XML, que se puede traducir como "JavaScript asíncrono + XML".

Introducción a AJAX

- Las tecnologías que forman AJAX son:
  - XHTML y CSS, para crear una presentación basada en estándares.
  - DOM, para la interacción y manipulación dinámica de la presentación.
  - XML, XSLT y JSON, para el intercambio y la manipulación de información.
  - XMLHttpRequest, para el intercambio asíncrono de información.
  - JavaScript, para unir todas las demás tecnologías.

Introducción a AJAX



Introducción a AJAX

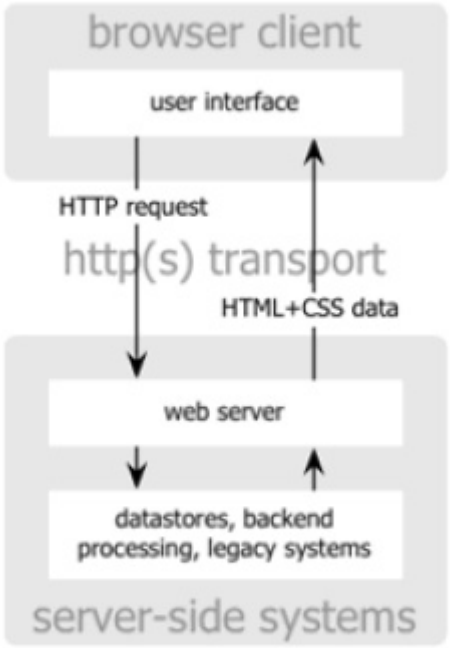
- En aplicaciones web tradicionales
  - Las acciones del usuario en la página (pinchar en un botón, seleccionar un valor de una lista, etc.) desencadenan llamadas al servidor.
  - Una vez procesada la petición del usuario, el servidor devuelve una nueva página HTML al navegador del usuario.

Introducción a AJAX

- Esta técnica funciona correctamente, pero no crea una buena sensación al usuario.
- Al realizar peticiones continuas al servidor, el usuario debe esperar a que se recargue la página con los cambios solicitados.
- Si la aplicación debe realizar peticiones continuas, su uso se convierte en algo molesto.
- Se pierde tiempo en recargar información ya cargada.

Introducción a AJAX

- Modelo tradicional de las aplicaciones web.



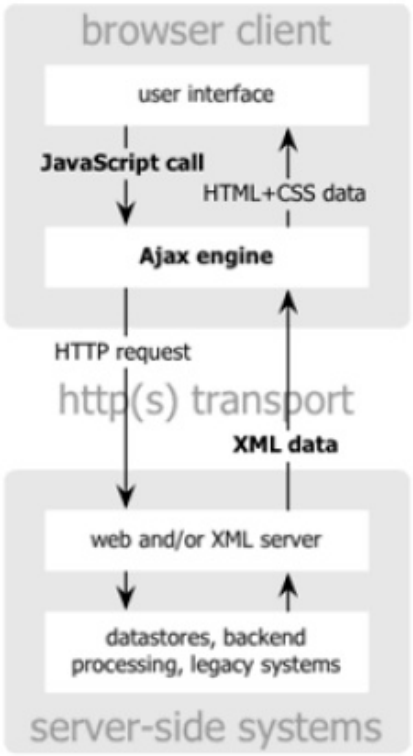
classic
web application model

Introducción a AJAX

- AJAX permite mejorar la interacción del usuario con la aplicación, evitando las recargas constantes de la página, ya que el intercambio de información con el servidor se produce en un segundo plano.
- El usuario nunca se encuentra con una ventana del navegador vacía esperando la respuesta del servidor.

Introducción a AJAX

- Nuevo modelo propuesto por AJAX.



Ajax
web application model

Utilización de XMLHttpRequest

- Todas las aplicaciones AJAX deben instanciar el objeto ***XMLHttpRequest***, dado que es el objeto que permite realizar comunicaciones con el servidor en segundo plano, sin recargar la página.
- La implementación del objeto ***XMLHttpRequest*** depende de cada navegador.

Utilización de XMLHttpRequest

- Los navegadores que siguen los estándares (Firefox, Safari, Opera, Internet Explorer 7 y 8) implementan el objeto ***XMLHttpRequest*** de forma nativa, por lo que se puede obtener a través del objeto ***window***.
- Los navegadores obsoletos (Internet Explorer 6 y anteriores) implementan el objeto ***XMLHttpRequest*** como un objeto de tipo ***ActiveX***.

Utilización de XMLHttpRequest

- Para adaptar una aplicación a todos los navegadores, se debería hacer

```
if(window.XMLHttpRequest) {
        // Navegadores que siguen los estándares
        peticion_http = new XMLHttpRequest();
} else if(window.ActiveXObject) {
        // Navegadores obsoletos
        peticion_http = new ActiveXObject("Microsoft.XMLHTTP");
}
```

Utilización de XMLHttpRequest

- **Propiedades de XMLHttpRequest**

Propiedad	Descripción
readyState	Valor numérico (entero) que almacena el estado de la petición
responseText	El contenido de la respuesta del servidor en forma de cadena de texto
responseXML	El contenido de la respuesta del servidor en formato XML. El objeto devuelto se puede procesar como un objeto DOM
status	El código de estado HTTP devuelto por el servidor (200 para una respuesta correcta, 404 para "No encontrado", 500 para un error de servidor, etc.)
statusText	El código de estado HTTP devuelto por el servidor en forma de cadena de texto: "OK", "Not Found", "Internal Server Error", etc.

Utilización de XMLHttpRequest

- **Posibles valores de readyState**

Valor	Descripción
0	No inicializado (objeto creado, pero no se ha invocado el método open)
1	Cargando (objeto creado, pero no se ha invocado el método send)
2	Cargado (se ha invocado el método send, pero el servidor aún no ha respondido)
3	Interactivo (se han recibido algunos datos, aunque no se puede emplear la propiedad responseText)
4	Completo (se han recibido todos los datos de la respuesta del servidor)

Utilización de XMLHttpRequest

- **Funciones de XMLHttpRequest**

Método	Descripción
`abort()`	Detiene la petición actual
`getAllResponseHeaders()`	Devuelve una cadena de texto con todas las cabeceras de la respuesta del servidor
`getResponseHeader("cabecera")`	Devuelve una cadena de texto con el contenido de la cabecera solicitada
`onreadystatechange`	Responsable de manejar los eventos que se producen. Se invoca cada vez que se produce un cambio en el estado de la petición HTTP. Normalmente es una referencia a una función JavaScript
`open("metodo", "url")`	Establece los parámetros de la petición que se realiza al servidor. Los parámetros necesarios son el método HTTP empleado y la URL destino (puede indicarse de forma absoluta o relativa)
`send(contenido)`	Realiza la petición HTTP al servidor
`setRequestHeader("cabecera", "valor")`	Permite establecer cabeceras personalizadas en la petición HTTP. Se debe invocar el método `open()` antes que `setRequestHeader()`

Utilización de XMLHttpRequest

- **Procedimiento**
  - Crear el objeto ***XMLHttpRequest***.
  - Definir como se ha de procesar la respuesta, en la función ***onreadystatechange***.
  - Crear la petición al servidor con la función ***open***, en la que se indica ***método HTTP***, ***url*** del servicio y si la petición es asincrona.
  - Enviar la petición con el método ***send***, enviando la información útil para el servicio.

Utilización de XMLHttpRequest

- Ejemplo

```
if(window.XMLHttpRequest) {
	peticion_http = new XMLHttpRequest();
} else if(window.ActiveXObject) {
	peticion_http = new ActiveXObject("Microsoft.XMLHTTP");
}

// Preparar la funcion de respuesta
peticion_http.onreadystatechange = muestraContenido;

// Realizar peticion HTTP
peticion_http.open('GET', 'http://localhost:8880/Ejemplo-
xmlHttpRequest/holamundo.txt', true);
peticion_http.send(null);
function muestraContenido() {
	if(peticion_http.readyState == 4) {
		if(peticion_http.status == 200) {
			alert(peticion_http.responseText);
		}
	}
}
```

Ejercicio

- A partir de la página web proporcionada, añadir el código JavaScript necesario para que:
  1. Al cargar la página, el cuadro de texto debe mostrar por defecto la URL de la propia página.
  2. Al pulsar el botón "Mostrar Contenidos", se debe descargar mediante peticiones AJAX el contenido correspondiente a la URL introducida por el usuario. El contenido de la respuesta recibida del servidor se debe mostrar en la zona de "Contenidos del archivo".
  3. En la zona "Estados de la petición" se debe mostrar en todo momento el estado en el que se encuentra la petición (No inicializada, cargando, completada, etc.)
  4. Mostrar el contenido de todas las cabeceras de la respuesta del servidor en la zona "Cabeceras HTTP de la respuesta del servidor".
  5. Mostrar el código y texto de estado de la respuesta del servidor en la zona "Código de estado".

Utilización de XMLHttpRequest

- **Envío de parámetros con la petición HTTP**
  - Se pueden realizar peticiones con los métodos HTTP, GET y POST.
  - La principal diferencia entre ambos métodos es que con POST los parámetros se envían en el cuerpo de la petición y con GET los parámetros se concatenan a la URL accedida.
  - El método GET tiene un límite de 512 bytes en la cantidad de datos que se pueden enviar. Si se intentan enviar más el servidor devuelve un error con código 414 (Request-URI Too Long).

Utilización de XMLHttpRequest

- **Envío de parámetros con la petición HTTP**
  - El objeto ***XMLHttpRequest*** no dispone de los campos que forman un formulario que se desea enviar al servidor, dicha información hay que generarla manualmente. Para ello es útil el método ***encodeURIComponent***.
  - En una petición ***POST***, hay que indicar que tipo de datos se están enviando para que el servidor no los descarte, esto se hace con la cabecera ***Content-Type***, y el método ***setRequestHeader***.

Utilización de XMLHttpRequest

- **Ejemplo**

```
if(window.XMLHttpRequest) {
        peticion_http = new XMLHttpRequest();
} else if(window.ActiveXObject) {
        peticion_http = new ActiveXObject("Microsoft.XMLHTTP");
}

// Preparar la funcion de respuesta
peticion_http.onreadystatechange = muestraContenido;

// Realizar peticion HTTP
peticion_http.open("POST", "http://localhost/ValidaDatos", true);

peticion_http.setRequestHeader("Content-Type",
                        "application/x-www-form-urlencoded");
var query_string = “dato1=" + encodeURIComponent(dato1.value) +
        "&dato2=“ + encodeURIComponent(dato2.value);
peticion_http.send(query_string);
```

Utilización de XMLHttpRequest

- JavaScript incluye unas funciones para trabajar con URL
  - **encodeURIComponent**(), codifica un trozo de una URL.
  - **decodeURIComponent**(), decodifica un trozo de URL.
  - **encodeURI**(), codifican una URL completa.
  - **decodeURI**(), decodifica una URL completa.
- Estas últimas no codifican los caracteres ; / ? : @ & = + $, #:

Ejercicio

- Comprobar si un nombre de usuario escogido está libre o ya lo utiliza otro usuario.
  1. Crear un script que valide contra el servidor con AJAX si el nombre está libre o no.
  2. El parámetro que contiene el nombre se llama login.
  3. La respuesta del servidor es "si" o "no“.
  4. A partir de la respuesta del servidor, mostrar un mensaje al usuario indicando el resultado de la comprobación.

Utilización de XMLHttpRequest

- **Procesado de respuestas XML**
  - Es posible obtener un XML como respuesta del servidor.
  - La respuesta del servidor se obtiene mediante la propiedad ***responseXML***, y se procesa empleando los métodos DOM de manejo de documentos XML/HTML.

Utilización de XMLHttpRequest

- **Ejemplo**

```
if (peticion_http.readyState == 4) {
        if (peticion_http.status == 200) {
                var documento_xml = peticion_http.responseXML;
                var root = documento_xml
                        .getElementsByTagName("respuesta")[0];
                var mensajes = root
                        .getElementsByTagName("mensaje")[0];
                var mensaje = mensajes.firstChild.nodeValue;
                var parametros = root
                        .getElementsByTagName("parametros")[0];
                var telefono = parametros
                .getElementsByTagName("telefono")[0]
                                .firstChild.nodeValue;
                var respuesta = mensaje + "\n" + "Fecha nacimiento = "
                        + fecha_nacimiento + "\n" + "Codigo postal = "
                        + codigo_postal + "\n" + "Telefono = "
                        + telefono;
                alert(respuesta);
        }
}
```

Ejercicio

- Partiendo del ejercicio anterior modificar el script para tener en cuenta el siguiente comportamiento.
  1. La respuesta del servidor es un documento XML con la siguiente estructura:
     - Si el nombre de usuario está libre:

```
<respuesta>
    <disponible>si</disponible>
</respuesta>
```

Ejercicio

- Si el nombre de usuario está ocupado:

```
<respuesta>
    <disponible>no</disponible>
    <alternativas>
        <login>...</login>
        <login>...</login>
        ...
        <login>...</login>
    </alternativas>
</respuesta>
```

3. Las alternativas se mostraran en una lista de elementos (
), con enlaces para cada uno de los nombres que al pinchar el nombre se copie en el cuadro de texto del login del usuario

Programación periódica

- En javascript, existen dos funciones que permiten ejecutar al cabo de un tiempo otras funciones.
  - **setInterval()**: Permite ejecutar de forma periódica la otra función indicando el lapso de tiempo.

```
setInterval(
        <callback>,
        <tiempo en milisegundos>,
        <parámetro1 para el callback>, <parámetro2 para el callback>, …
);
```

Programación periódica

- **setTimeout()**: Permite ejecutar una función una única vez al cabo de un tiempo.

```
setTimeout(
        <callback>,
        <tiempo en milisegundos>,
        <parámetro1 para el callback>, <parámetro2 para el callback>, …
);
```

Expresiones regulares

- Las expresiones regulares son una serie de caracteres que forman un patrón, que representan a otro grupo de caracteres mayor, de tal forma que podemos comparar el patrón con otros conjuntos de caracteres para ver las coincidencias.
- Las expresiones regulares son una forma sencilla para
  - Manipular datos.
  - Realizar búsquedas.
  - Reemplazar strings.

Expresiones regulares

(patrón)

$1, ..., $9 del objeto **RegExp**.

- Para comprobar que hay coincidencia, pero no capturar, es decir no se pueden recuperar las cadenas encontradas con los atributos $1 ... $9 del objeto **RegExp**

(?:patrón)

Expresiones regulares

Carácter	Texto buscado
^	Principio de entrada o línea.
$	Fin de entrada o línea.
*	El carácter anterior 0 o más veces.
+	El carácter anterior 1 o más veces.
?	El carácter anterior una vez como máximo (es decir, indica que el carácter anterior es opcional).
.	Cualquier carácter individual, salvo el de salto de línea.
x|y	x o y.
{n}	Exactamente n apariciones del carácter anterior.
{n,m}	Como mínimo n y como máximo m apariciones del carácter anterior.
[abc]	Cualquiera de los caracteres entre corchetes. Especifique un rango de caracteres con un guión (por ejemplo, [a-f] es equivalente a [abcdef]).

Expresiones regulares

Carácter	Texto buscado
[^abc]	Cualquier carácter que no esté entre corchetes. Especifique un rango de caracteres con un guión (por ejemplo, [^a-f] es equivalente a [^abcdef]).
\b	Límite de palabra (como un espacio o un retorno de carro).
\B	Cualquiera que no sea un límite de palabra.
\d	Cualquier carácter de dígito. Equivalente a [0-9].
\D	Cualquier carácter que no sea de dígito. Equivalente a [^0-9].
\f	Salto de página.
\n	Salto de línea.
\r	Retorno de carro.

Expresiones regulares

Carácter	Texto buscado
\s	Cualquier carácter individual de espacio en blanco (espacios, tabulaciones, saltos de página o saltos de línea).
\S	Cualquier carácter individual que no sea un espacio en blanco.
\t	Tabulación.
\w	Cualquier carácter alfanumérico, incluido el de subrayado. Equivalente a [A-Za-z0-9_].
\W	Cualquier carácter que no sea alfanumérico. Equivalente a [^A-Za-z0-9_].

Expresiones regulares

- **Algunos ejemplos de expresiones**

Cualquier letra en minuscula	[a-z]
Entero	^(?:\+|-)?\d+$
Correo electrónico	/[\w-\.]{3,}@([\w-]{2,}\.)*([\w-]{2,}\.)[\w-]{2,4}/
URL	^(ht|f)tp(s?)\:\/\/[0-9a-zA-Z]([-.\w]*[0-9a-zA-Z])*(:(0-9)*)*(\/?)([a-zA-Z0-9\-\.\?\,\'\/\\\+&%\$#_]*)?$
Contraseña segura	(?!^[0-9]*$)(?!^[a-zA-Z]*$)^([a-zA-Z0-9]{8,10})$ (Entre 8 y 10 caracteres, por lo menos un digito y un alfanumérico, y no puede contener caracteres espaciales)
Fecha	^\d{1,2}\/\d{1,2}\/\d{2,4}$ (Por ejemplo 01/01/2007)
Hora	^(0[1-9]|1\d|2[0-3]):([0-5]\d):([0-5]\d)$ (Por ejemplo 10:45:23)
Número tarjeta de crédito	^((67\d{2})|(4\d{3})|(5[1-5]\d{2})|(6011))(-?\s?\d{4}){3}|(3[4,7])\ d{2}-?\s?\d{6}-?\s?\d{5}$
Número teléfono	^[0-9]{2,3}-? ?[0-9]{6,7}$
Código postal	^([1-9]{2}|[0-9][1-9]|[1-9][0-9])[0-9]{3}$
Certificado Identificación Fiscal	^(X(-|\.)?0?\d{7}(-|\.)?[A-Z]|[A-Z](-|\.)?\d{7}(-|\.)? [0-9A-Z]|\d{8}(-|\.)?[A-Z])$

Expresiones regulares

- En javascript para manejar expresiones regulares se tiene la tipología **RegExp**.

```
var reg = new RegExp(“patron”,”flags”);
```

- Aunque se pueden definir la expresiones de forma directa sin necesidad de hacer referencia al tipo **RegExp**, no es recomendable, ya que dificulta la comprensión.

```
var reg = /patron/flags
```

Expresiones regulares

- Para definir la expresión regular, se definen
  - **patrón**: La expresión
  - **flag**: Combinación de los siguientes valores
    - **g**: Indica que se realice una búsqueda global.
    - **i**: Indica que se ignoren mayúsculas o minúsculas.
    - **m**: Tratar caracteres de inicio y fin (^ y $) como inicio y fin de línea y no de texto.

Expresiones regulares

- Un ejemplo definido de las dos formas posibles

```
var re = new RegExp("\\w+");

var re = /\w+/;
```

Expresiones regulares

- Los métodos que proporciona este tipo son
  - **patrón.exec(cadena**): devuelve un array donde el elemento 0, tiene la primera correspondencia hallada en la cadena
  - **patrón.test(cadena**): devuelve el booleano que indica si la cadena cumple o no con la expresión.

Expresiones regulares

- Se aplican expresiones en métodos de la clase String.
  - **cadena.match(patron)**: devuelve el array con las coincidencias definidas por **patrón**, encontradas en **cadena** o null.
  - **cadena.replace(patron,cadena2)**: devuelve un string, donde partiendo de **cadena**, se han sustituido las coincidencias con el **patrón**, por **cadena2**.
  - **cadena.split(patron)**: devuelve un array, cuyos elementos son trozos de cadena, donde se han tomando como puntos de corte, la expresión indicada por el patrón.
  - **cadena.search(patron)**: devuelve la posición de la primera coincidencia