Tema 2. Lenguajes de programación en el lado del cliente.

Desarrollo de Tecnologías Web: Tecnologías en el Cliente (DTWTC)

Master Universitario en Ingeniería de Servicios y Aplicaciones Web

Ramón Vte. Cirilo (Ramon.V.Cirilo@uv.es)

Índice

- DOM – Document Object Model
- BOM – Browser Object Model
- JavaScript
  - Gestión de documentos HTML5 a través del DOM
  - Gestión de la visualización
  - Gestión de formularios
- Java, JavaScript y XML
- La tecnología AJAX

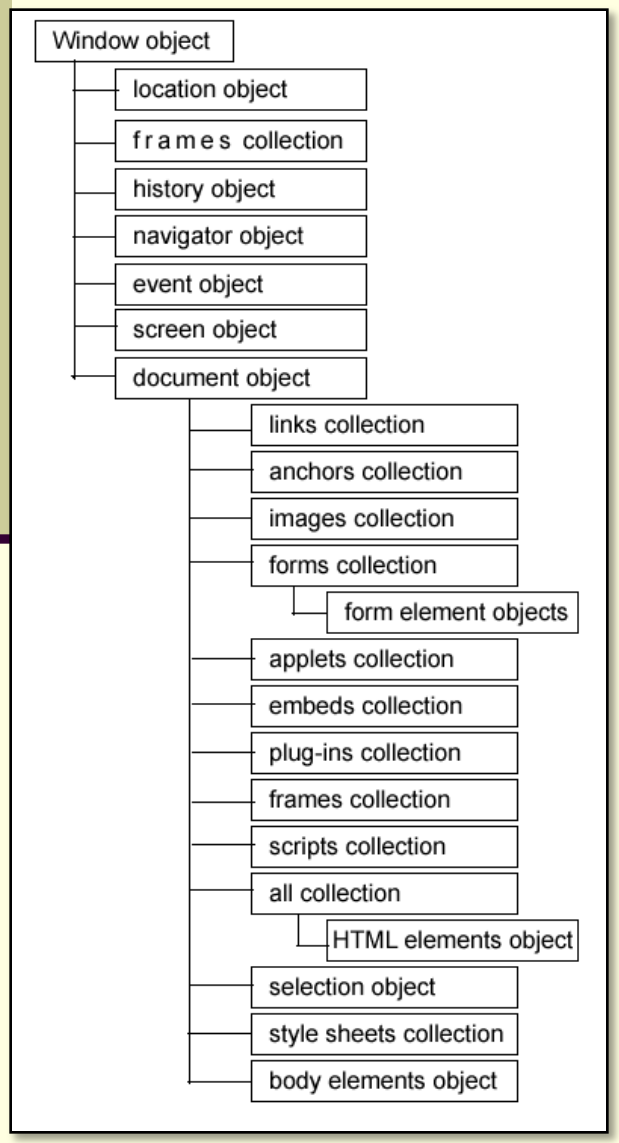
Object Model

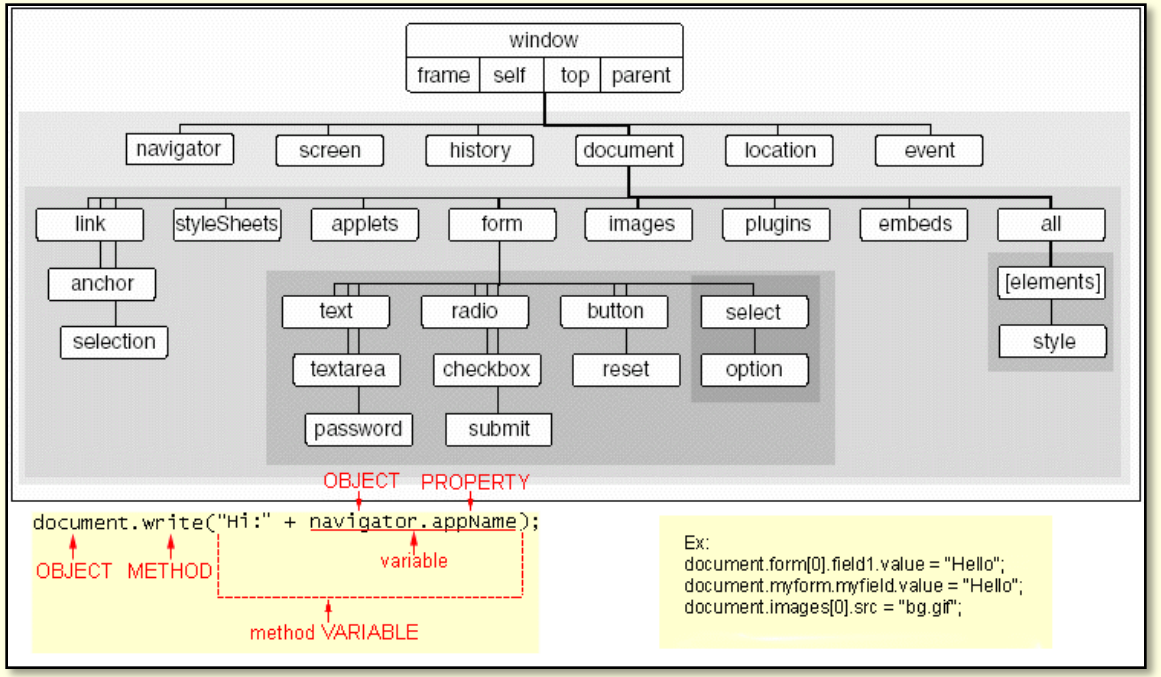
- Un modelo de objetos permite:
  - Conceptualización de contenidos
  - Representación de estructura no cíclica
  - Capacidad sobre el contenido de:
    - Recorrido
    - Modificación
    - Borrado
- Para su uso se precisa
  - Lenguaje específico
  - API para la navegación

HTML Object Model

Objetos Javascript	Objetos Navegador	Objetos HTML	
ARRAY BOOLEAN DATE MATH NUMBER STRING REGEXP PROPIEDADES GLOBALES ✓Infinity ✓NaN ✓undefined METODOS GLOBALES ✓decodeURI() ✓decodeURIComponent() ✓encodeURI() ✓encodeURIComponent() ✓escape() ✓eval() ✓isFinite() ✓isNaN() ✓Number() ✓parseFloat() ✓parseInt() ✓String() ✓unescape()	WINDOW NAVIGATOR SCREEN HISTORY LOCATION	DOCUMENT HTMLELEMENT ANCHOR AREA BASE BODY BUTTON EVENT FORM FRAME FRAMESET IFRAME IMAGE INPUT_BUTTON INPUT_CHECKBOX INPUT_FILE INPUT_HIDDEN INPUT_PASSWORD INPUT_RADIO INPUT_RESET INPUT_SUBMIT INPUT_TEXT	LINK META OBJECT OPTION SELECT STYLE TABLE TABLECELL TABLEROW TEXTAREA

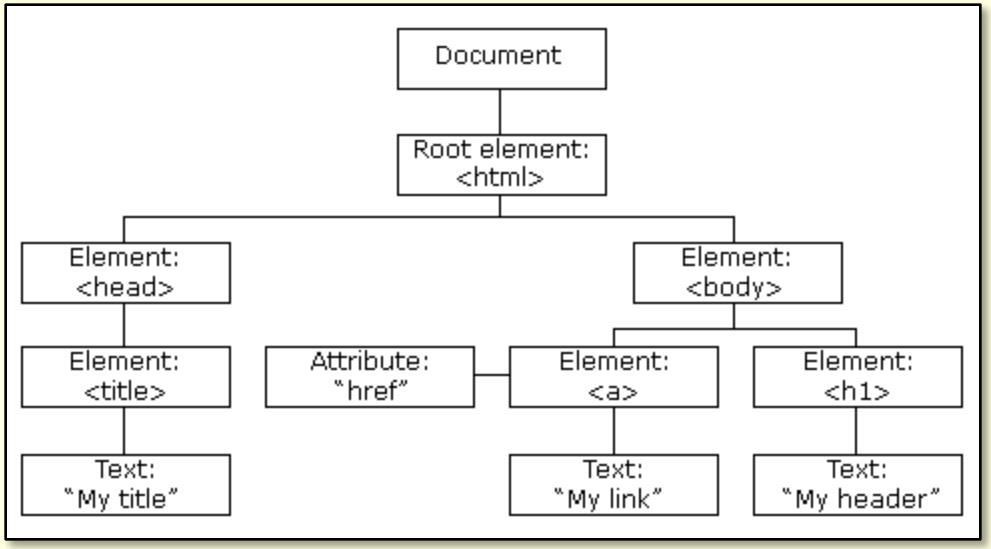
HTML Object Model

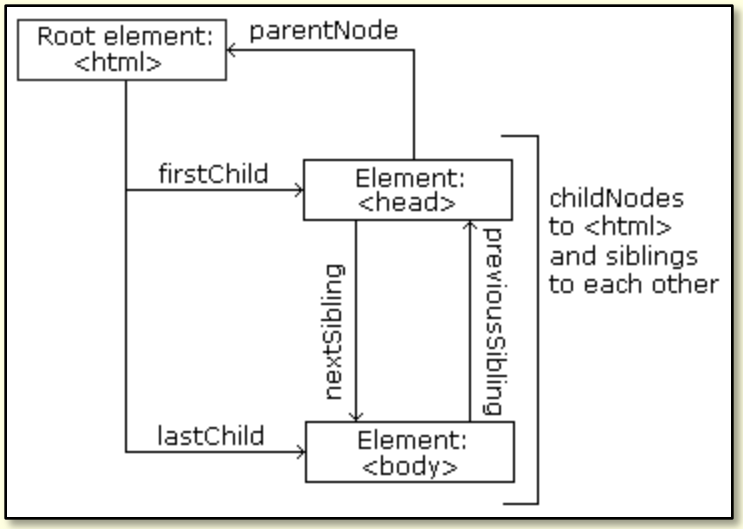




HTML DOM

- Ejemplo de árbol HTML DOM





HTML DOM

- Objetos principales
  - Document

Colecciones	Propiedades	Métodos
anchors[]	cookie	close()
forms[]	documentMode	getElementById()
images[]	domain	getElementsByName()
links[]	lastModified	open()
	readyState	write()
	referrer	writeln()
	title	
	URL	

HTML DOM

- Objetos principales
  - Event

Handlers		Mouse/Keyboard attributes	Other attributes
onabort onblur onchange onclick ondblclick onerror onfocus onkeydown onkeypress onkeyup onload onmousedown	onmousemove onmouseout onmouseover onmouseup onreset onresize onselect onsubmit onunload	altKey button clientX clientY ctrlKey metaKey relatedTarget screenX screenY shiftKey	bubbles cancelable currentTarget eventPhase target timeStamp type

HTML DOM

- Objetos principales
  - Element

Colecciones	Propiedades		Métodos	Eventos
attributes[] childNodes[]	accessKey className clientHeight clientWidth dir disabled firstChild height id innerHTML lang lastChild length nextSibling nodeName nodeType nodeValue	offsetHeight offsetLeft offsetParent offsetTop offsetWidth ownerDocument parentNode previousSibling scrollHeight scrollLeft scrollTop scrollWidth style tabIndex tagname title width	appendChild() blur() click() cloneNode() focus() getAttribute() getElementsByTagName() hasChildNodes() insertBefore() item() normalize() removeAttribute() removeChild() replaceChild() setAttribute() toString()	onblur onclick ondblclick onfocus onkeydown onkeypress onkeyup onmousedown onmousemove onmouseout onmouseover onmouseup onresize

HTML DOM

- Para avanzar más:
  - Referencia

    http://www.w3schools.com/jsref/default.asp

  - Tutorial

    http://www.w3schools.com/htmldom/default.asp

  - Ejemplos y práctica

    http://www.w3schools.com/htmldom/dom_examples.asp

HTML BOM

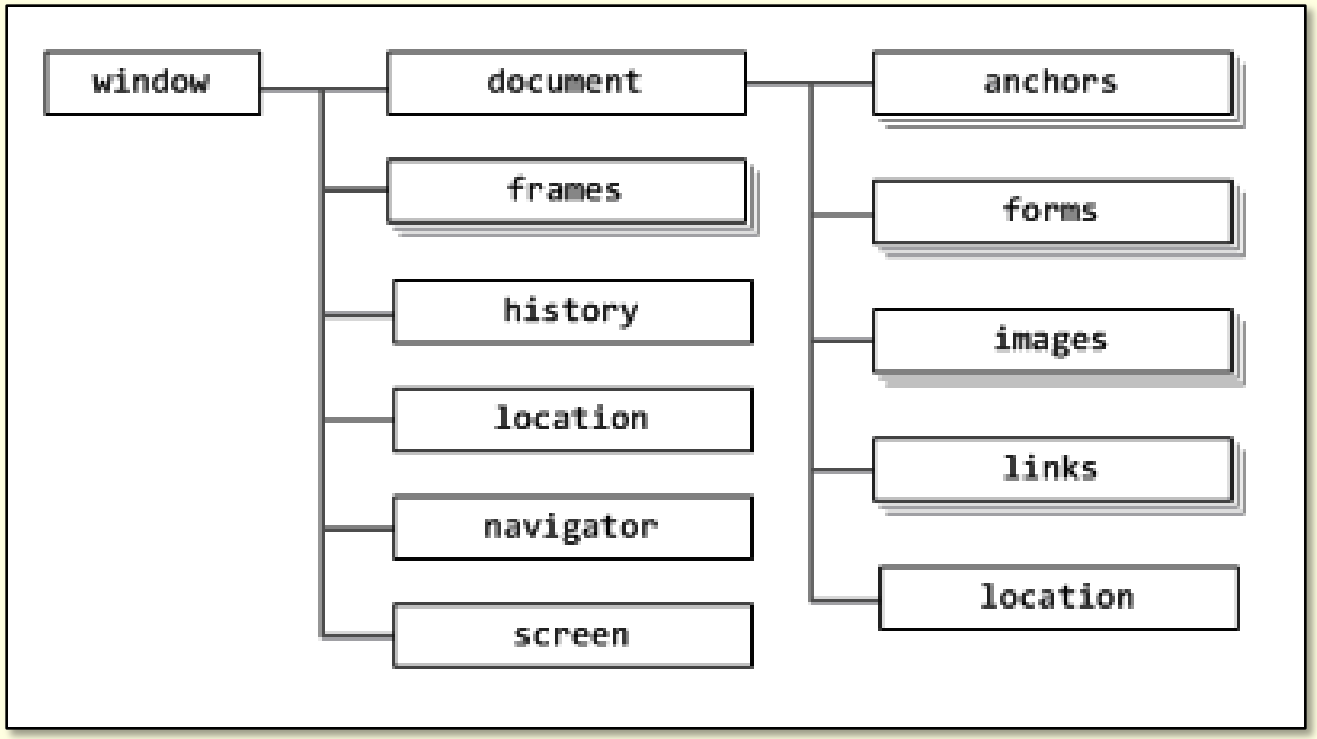
- El BOM (Browser Object Model) de HTML es un modelo de objetos para el navegador sobre el que se representan las páginas HTML
- El BOM, al igual que el DOM, tiene estructura de árbol
- El elemento padre del árbol es el objeto Window del que dependen el resto: Document, Frames, History, Location, Navigator y Screen.

HTML BOM

- El BOM es posible realizar algunas acciones como:
  - Crear, mover, redimensionar y cerrar ventanas de navegador.
  - Obtener información sobre el propio navegador.
  - Trabajar con las propiedades de la página actual y de la pantalla del usuario.
  - Gestionar las cookies.
  - …

HTML BOM

- Jerarquía del BOM



HTML BOM. Objetos principales

- Window

Propiedades		Métodos	
closed	outerWidth	alert()	print()
defaultStatus	pageXOffset	blur()	prompt()
document	pageYOffset	clearInterval()	resizeBy()
frames	parent	clearTimeout()	resizeTo()
history	screenLeft	close()	scrollBy()
innerHeight	screenTop	confirm()	scrollTo()
innerWidth	screenX	focus()	setInterval()
length	screenY	moveBy()	setTimeout()
location	self	moveTo()	
name	status	open()	
opener	top		
outerHeight			

Nota: No existe un estándar público que se pueda aplicar al objeto Window. Aún así, todos los principales navegadores lo soportan.

HTML BOM. Objetos principales

- Location (Es una propiedad tanto del objeto window como del objeto document)

Propiedad	Descripción
hash	El contenido de la URL que se encuentra después del signo # (para los enlaces de las anclas)
host	El nombre del servidor
hostname	La mayoría de las veces coincide con host (en ocasiones, se eliminan las www del principio)
href	La URL completa de la página actual
pathname	Todo el contenido que se encuentra después del host
port	Si se especifica en la URL, el puerto accedido.
protocol	El protocolo empleado por la URL (lo que se encuentra antes de las dos barras)
search	Todo el contenido que se encuentra tras el símbolo ?, es decir, la consulta o "query string"
assign ()	Carga un documento
reload ()	Recarga el documento actual
replace ()	Sustituye el documento actual por otro documento

HTML BOM. Objetos principales

- Navigator

Propiedad	Descripción
appCodeName	Codename del browser
appName	Nombre del browser
appVersion	Versión del browser
cookieEnabled	Determina si las cookies están habilitadas
onLine	Devuelve True si el browser está online, y False en caso contrario
platform	Plataforma en la que se ha compilado el browser
userAgent	User-Agent que envía el browser al servidor
javaEnabled ()	Especifica si el browser tiene habilitado el lenguaje Java

Nota: Es de los menos estandarizados, aunque alguna de sus propiedades son comunes en todos los navegadores

HTML BOM. Objetos principales

History

Propiedad	Descripción
length	Número de URLs en la lista del histórico
go ()	Carga un URL de la lista del histórico
back ()	Carga el URL anterior
forward ()	Carga el URL siguiente

Screen

Propiedad	Descripción
availHeight	Altura de la pantalla (excluyendo la barra de tareas)
availWidth	Anchura de la pantalla (excluyendo la barra de tareas)
colorDepth	Profundidad de bit de la paleta de colores para mostrar imágenes
height	Altura total de la pantalla
pixelDepth	Resolución de color en bits por pixel de la pantalla
width	Anchura total de la pantalla

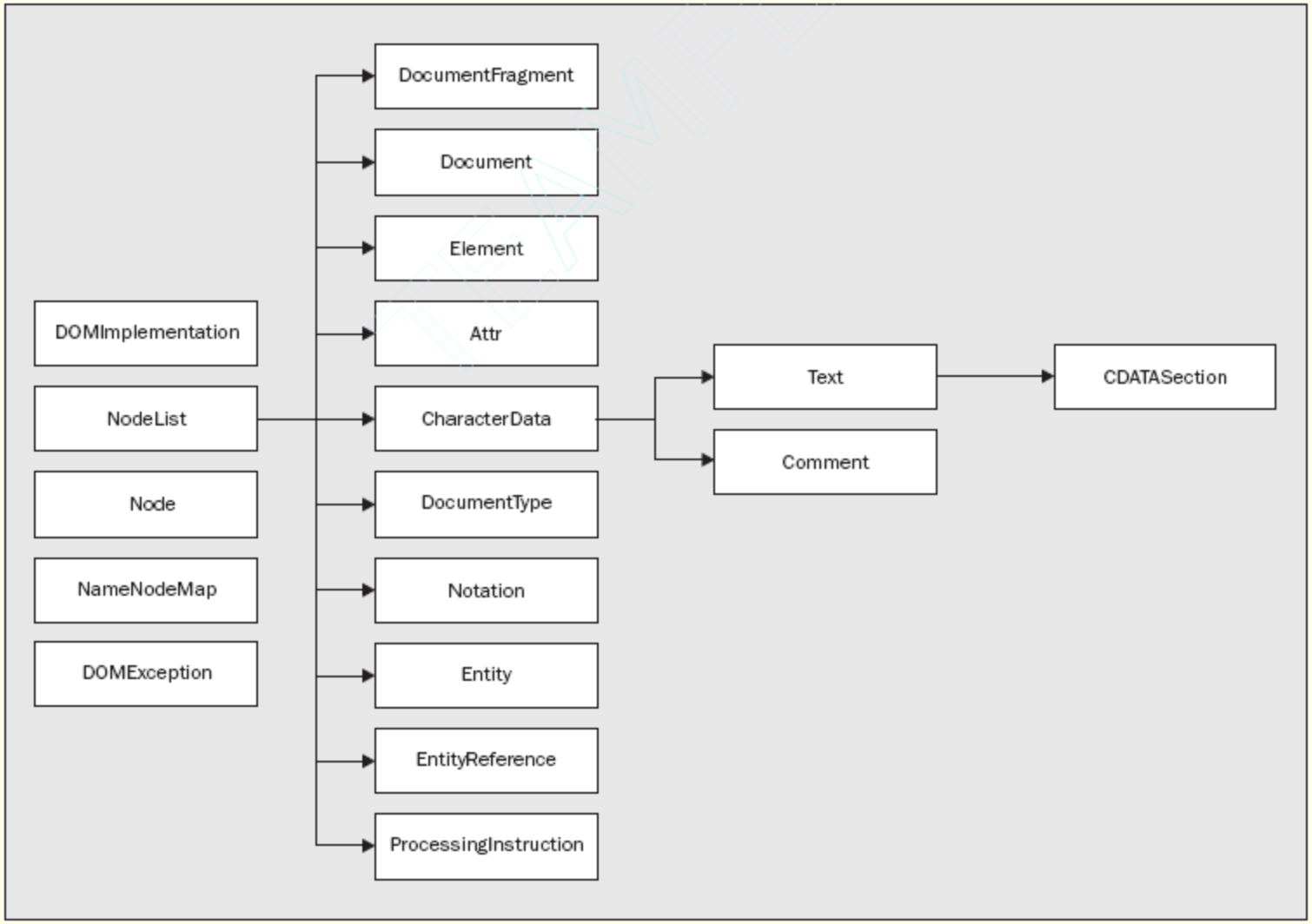
XML DOM

- El modelo de objetos XML tiene, de nuevo, estructura de árbol
- La especificación fue desarrollada por el W3C
- Su propósito es proporcionar un interface para:
  - Crear documentos XML
  - Navegar por documentos XML
  - Añadir, modificar o borrar partes del documento cuando están cargados en memoria

XML DOM

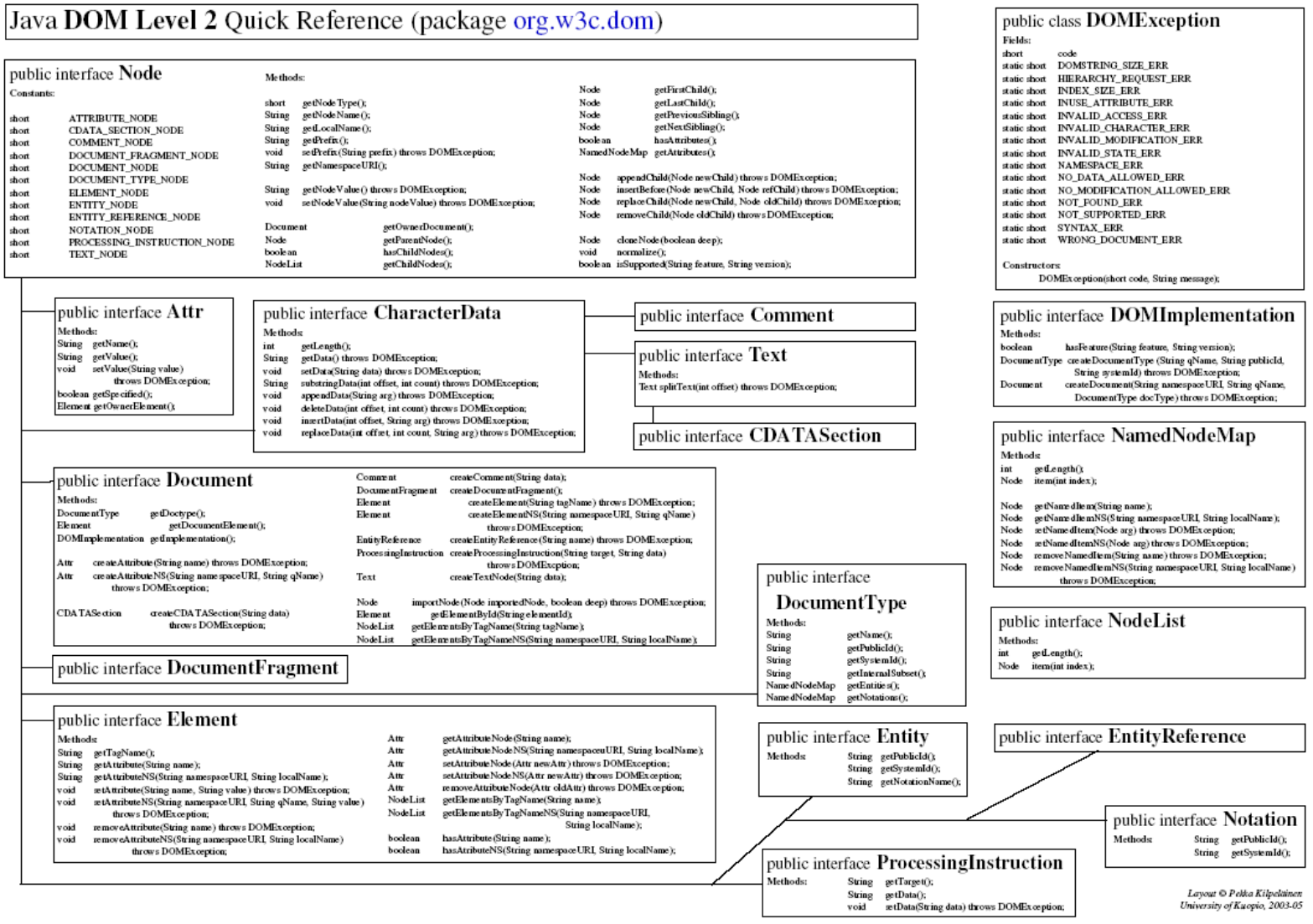
- Interfaces
  - Node: tipo de datos básico del DOM
  - Element: etiquetas del documento XML
  - Attr: atributos de los elementos
  - Text: el contenido de un elemento o atributo
  - Document: representa el documento XML
- Métodos
  - Document.getDocumentElement()
  - Node.getFirstChild(), Node.getLastChild()
  - Node.getNextSibling(), Node.getPreviousSibling()
  - Node.getAttribute(attrName)
  - Document.getElementsByTagName(tagName)

XML DOM – Interfaces



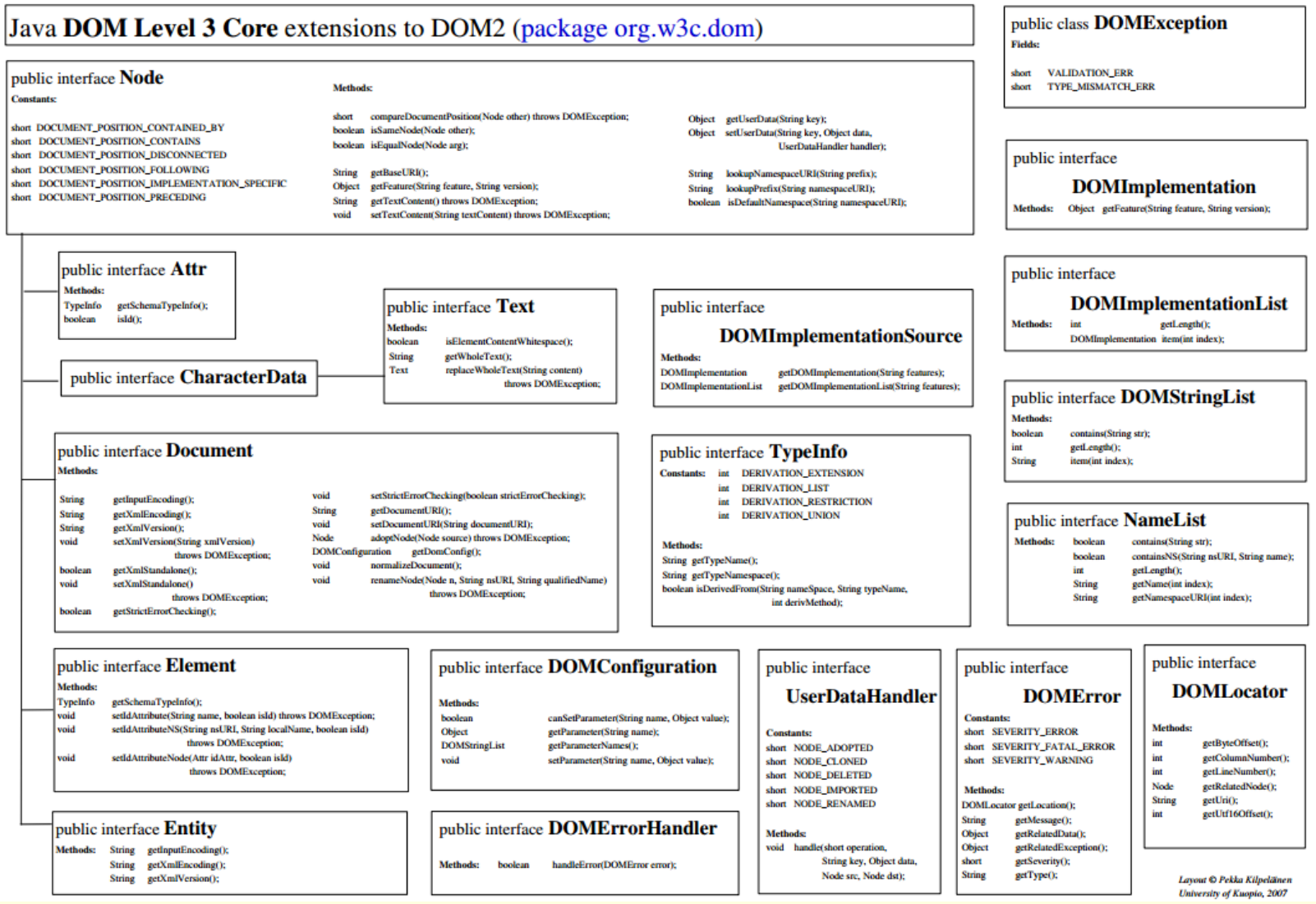
XML DOM - Interfaces

http://www.cs.uku.fi/~kilpelai/RDK10/refs/DOM-quickref.pdf



XML DOM - Interfaces

http://www.cs.uku.fi/~kilpelai/RDK10/refs/DOM3-quickref.pdf



XML DOM – Jerarquía de nodos

- Document -- Element (maximum of one), ProcessingInstruction, Comment, DocumentType (maximum of one)
- DocumentFragment -- Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference
- DocumentType -- no children
- EntityReference -- Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference
- Element -- Element, Text, Comment, ProcessingInstruction, CDATASection, EntityReference
- Attr -- Text, EntityReference
- ProcessingInstruction -- no children
- Comment -- no children
- Text -- no children
- CDATASection -- no children
- Entity -- Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference
- Notation -- no children

http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/core.html

XML DOM

- Para avanzar más:
  - Referencia

    http://www.w3schools.com/dom/dom_node.asp

  - Tutorial

    http://www.w3schools.com/dom/default.asp

  - Ejemplos y práctica

    http://www.w3schools.com/dom/dom_examples.asp

Gestión del DOM con JavaScript

- Existen dos formas para interaccionar con el DOM a través de Javascript
  - Uso directo del lenguaje
    - Pros: Posibilidad de uso completo de las funciones del lenguaje
    - Contras: Necesidad de considerar todos los navegadores y sus versiones
  - Uso de librerías específicas
    - Pros: Consideran las variantes de uso de distintos navegadores
    - Contras: Implementan sólo una parte reducida del lenguaje

Gestión del DOM con JavaScript

- JavaScript permite usar el interface DOM para trabajar con:
  - El documento HTML
  - El modo de visualización de los contenidos a través de CSS
  - Los eventos generados en cada parte del documento
- En particular, es importante conocer como trabajar con los formularios HTML y sus componentes, puesto que son la principal fuente de información en aplicaciones Web.

Gestión del DOM con JavaScript

- Nos centraremos en la aproximación basada en el uso directo de JavaScript.
- El acceso al HTML DOM se hace usando directamente el objeto document.

```
document.writeln('<pre>Codigo de prueba: i++</pre>');

document.getElementById('254').innerHTML='';
```

- El acceso al HTML BOM se hace usando directamente el objeto window.

```
confirm('¿Se encuentra ud. en: ' + window.location.href + '?');

v = window.open('','','width=200,height=100');
v.close();
```

Gestión del DOM con JavaScript

- Acceso directo a elementos del DOM
  - getElementById (id)
  - getElementsByClassName (class)
  - getElementsByName (name)
  - getElementsByTagName (tag)

Gestión del DOM con JavaScript

- Acceso por navegación a elementos del DOM
  - <element>.childNodes
  - <element>.firstChild
  - <element>.hasChildNodes()
  - <element>.lastChild
  - <element>.nextSibling
  - <element>.parentNode
  - <element>.previousSibling

Gestión del DOM con JavaScript

- Trabajando con los elementos HTML
  - Atributos importantes
    - <element>.className
    - <element>.disabled
    - <element>.hidden
    - <element>.id
    - <element>.tagName
    - <element>.attributes
    - <element>.innerHTML
    - <element>.outerHTML

Gestión del DOM con JavaScript

- Trabajando con los elementos HTML
  - Funciones importantes (1)
    - <element>.add (class)
    - <element>.contains (class)
    - <element>.remove (class)
    - <element>.toggle (class)
    - <element>.getAttribute (name)
    - <element>.hasAttribute (name)
    - <element>.removeAttribute (name)
    - <element>.setAttribute (name, value)

Gestión del DOM con JavaScript

- Trabajando con los elementos HTML
  - Funciones importantes (2)
    - <element>.appendChild (html_element)
    - <element>.cloneNode (boolean)
    - <element>.insertBefore (new, child)
    - <element>.removeChild (element)
    - <element>.replaceChild (old, new)
    - <element>.createElement (element)
    - <element>.createTextNode (text)

Gestión del DOM con JavaScript

- Desplazamiento dentro del documento
  - A través del objeto window

```
window.scrollBy(0,50); // horizontal, vertical

window.scrollTo(200,500); // horizontal, vertical
```

  - A través del objeto location

```
<script>
    document.getElementById("mi_id").onclick = function() {
    document.location.hash = "mapa";
    }
</script>
```

Gestión del DOM con JavaScript

- Ejemplos de uso

EJEMPLOS DE USO

```
document.getElementById("intro");
document.getElementsByTagName("p");
```

```
txt = document.getElementById("intro").innerHTML;
document.getElementById("p1").innerHTML = "New text!";
```

```
txt = document.getElementById("intro").childNodes[0].nodeValue;
```

```
document.getElementById('main').getElementsByTagName("p");
```

```
x = document.getElementsByTagName("p");
for (i = 0; i < x.length; i++)
{
    document.write(x[i].innerHTML);
    document.write("<br />");
}
```

```
document.body.style.backgroundColor = "lavender";
document.getElementById("p1").style.color = "blue";
document.getElementById("p1").style.fontFamily = "Arial";
```

Control de formularios con JavaScript

- HTML5 tiene sus propios mecanismos de control para los formularios, p. ej.:
  - Definición de valores máximos y mínimos
  - Definición de expresiones regulares para ciertos campos
  - Limitación del tamaño de un campo
  - Control de cambios sobre elementos
  - …
- JavaScript, a través del API del DOM, permite manejar un abanico más amplio de funciones de control.

Control de formularios con JavaScript

- Controles propios de HTML5
  - [autocomplete: “ON” | “OFF”]
    - “Recuerda” datos introducidos con anterioridad
    - Se aplica a <form> o a <input>
  - [pattern = regexp]
  - [autofocus]
  - [disabled]
    - Se puede aplicar a elementos o a un <fieldset>
  - [readonly]
  - [required]
  - [min], [max], [step], [maxlength]

Control de formularios con JavaScript

- Controles propios de HTML5
  - Eventos de control
    - <form> [oninput]
    - <form> [onsubmit]
    - <element> [onselect]
    - <element> [onchange]
    - <element> [onfocus], [onblur], [onfocusin], [onfocusout]
    - <element> [oncontextmenu]
    - [onkeydown], [onkeypress], [onkeyup]
    - [onclick], [ondoubleclick], [onmousewheel], [onscroll], [onmousedown], [onmouseup], [onmouseover], [onmouseout]

Control de formularios con JavaScript

- Controles del DOM de HTML5
  - document.activeElement
  - <element>.checked
  - <element>.disabled | .hidden
  - <element>.id | .attributes | .classname
  - alert(), confirm(), prompt()

Control de formularios con JavaScript

- Gestión de eventos: Manejadores de eventos
  - Atributos que forman parte de los elementos

```
<input type="text"
  onfocus="if(typeof focused == 'undefined' ||
    focused == 0)
    alert('Texto en mayúsculas'); focused = 1;"
  onblur="this.value=this.value.toUpperCase(); focused = 0;"/>
```

  - Atributos que invocan a funciones

```
<script>
var focused = 0;
function foco(obj) {
    if (focused != 1) {alert('Texto en mayúsculas'); focused = 1;}
}
function sinfoco(obj) {
    obj.value = obj.value.toUpperCase(); focused = 0;
}
</script>
...
<input type="text" onfocus="foco(this);" onblur="sinfoco(this);"/>
```

Control de formularios con JavaScript

- Gestión de eventos: Manejadores de eventos
  - Asociar un manejador a un elemento (1)

```
<script>

var focused = 0;

document.getElementById('mi_id').onfocus = foco;
document.getElementById('mi_id').onblur = sinfoco;

function foco(e) {
    if (focused != 1) {alert('Texto en mayúsculas'); focused = 1;}
}

function sinfoco(e) {
    e.target.value = e.target.value.toUpperCase(); focused = 0;
}

</script>
...
<input type="text" id="mi_id"/>
```

Control de formularios con JavaScript

- Gestión de eventos: Manejadores de eventos
  - Asociar un manejador a un elemento (2)

```
<script>

var focused = 0;

document.getElementById('mi_id').addEventListener("focus",foco);
document.getElementById('mi_id').addEventListener("blur",sinfoco);

function foco(e) {
    if (focused != 1) {alert('Texto en mayúsculas'); focused = 1;}
}

function sinfoco(e) {
    e.target.value = e.target.value.toUpperCase(); focused = 0;
}

</script>
...
<input type="text" id="mi_id"/>
```

Control de formularios con JavaScript

- Propiedades de los eventos

Propiedad	Descripción
type	Nombre del evento
target	Elemento que dispara el evento
currentTarget	Elemento al que se le asigna el evento
eventPhase	La fase en el ciclo de vida del evento (CAPTURING_PHASE, AT_TARGET, BUBBLING_PHASE – *captura, ejecución, propagación*)
bubbles	True si el evento se propaga en el documento.
cancelable	True si el evento tiene una acción por defecto que puede ser cancelada
timeStamp	Fecha y hora en la que ocurrió el evento
stopPropagation()	Detiene el flujo de propagación del evento a través de los padres del árbol de elementos.
stopImmediatePropagation()	Detiene el flujo de propagación del evento a través del árbol de elementos, incluyendo eventos sobre el mismo elemento que no se hayan disparado.
preventDefault()	Evita que el browser realice la acción por defecto asociada al evento.
defaultPrevented	True si preventDefault() ha sido invocado.

Control de formularios con JavaScript

- Control de ejecución de eventos
  - El orden de ejecución de eventos que dispara un elemento del DOM es en orden decreciente de padre a hijo, para todos los nodos que capturen el evento.
  - Se puede controlar el orden y la propagación de eventos de dos formas:
    - target.addEventListener(tipo, listener[, useCapture]). useCapture puede ser true o false. Si es true, el padre captura y ejecuta antes que el hijo.
    - event.stopPropagation(). Evita que una vez capturado un evento, los padres o los hijos puedan capturarlo también.
    - event.preventDefault(). Evita que se ejecuten las acciones por defecto asociadas a eventos sobre elementos.

Control de formularios con JavaScript

- Gestión del evento Submit
  - El envío del formulario al servidor se hace a través de un elemento input de tipo Submit o invocando la función de javascript submit() sobre el objeto del formulario.
  - Se puede capturar el evento submit (onsubmit) y actuar sobre el para realizar validaciones
    - Se suelen crear funciones que se invocan al producirse el evento
    - La función debe devolver un valor TRUE o FALSE. En el primer caso, los datos se envían al servidor; en el segundo caso, no se realiza el envío.

Nota: Hacer un return false en un manejador de eventos es equivalente a un preventDefault y stopPropagation a la vez

Control de visualización con JavaScript

- El objetivo es controlar las hojas de estilo CSS asociadas a un documento con JavaScript
- En primer lugar, **document.stylesheets** contiene un conjunto de objetos de tipo CSSStyleSheet:

Propiedad	Descripción
cssRules	Reglas de la hoja de estilos
deleteRule(<pos>)	Elimina un regla de la hoja de estilos
insertRule(<rule>,<pos>)	Inserta una nueva regla
disabled	Permite consultar o cambiar el estado de habilitación
href	Devuelve el URL de la hoja de estilos
media	Indica los medios a los que aplica la hoja de estilos
ownerNode	Elemento HTML sobre el que se define el estilo
title	Titulo del estilo
type	Tipo del estilo

Control de visualización con JavaScript

- Gestión de los “media”

Propiedad	Descripción
appendMedium(<med>)	Agrega un medio
deleteMedium(<med>)	Elimina un medio
item(<pos>)	Devuelve un medio en una determinada posición
length	Longitud del media (cantidad de medios definidos)
mediaText	Devuelve el valor del texto del atributo media

Control de visualización con JavaScript

- Ejemplo

```
<style title="core styles">
p {
    border: medium double black;
    background-color: lightgray;
}

#block1 { color: white;}

table {border: thin solid black; border-collapse: collapse;
    margin: 5px; float: left;}

td {padding: 2px;}
</style>

<link rel="stylesheet" type="text/css" href="styles.css"/>

<style media="screen AND (min-width:500px), PRINT" type="text/css">
    #block2 {color:yellow; font-style:italic}
</style>
```

Control de visualización con JavaScript

- Ejemplo

Propiedad			
index	0	1	2
href	null	http://www.uv.es/styles.css	null
title	core styles	null	null
type	text/css	text/css	text/css
ownerNode	STYLE	LINK	STYLE

Datos	
Media count:	2
Media text:	screen and (min-width:500px), print
Media 0:	screen and (min-width:500px)
Media 1:	Print

Control de visualización con JavaScript

- Control de estilos individuales: CSSRuleList

Propiedad	Descripción
item(<pos>)	Devuelve un estilo en una determinada posición
length	Longitud de la lista (cantidad de estilos definidos)
cssText	Devuelve o actualiza el texto que define el estilo
parentStyleSheet	Hoja de estilo a la que pertenece
selectorText	Devuelve o actualiza el selector del texto que define el estilo
style	Obtiene un objeto **CSSStyleDeclaration** que representa los estilos

Control de visualización con JavaScript

- Control de estilos: CSSStyleDeclaration

Propiedad	Descripción
cssText	Devuelve o actualiza el texto que define el estilo
getPropertyCSSValue(<name>)	Obtiene la propiedad especificada
getPropertyPriority(<name>)	Obtiene la prioridad de la propiedad especificada
getPropertyValue(<name>)	Obtiene el valor especificado como un String
item(<pos>)	Devuelve un ítem en una determinada posición
length	Longitud de la lista (número de ítems)
parentRule	Regla del estilo
removeProperty(<name>)	Elimina la propiedad especificada
setProperty(<name>, <value>[,<priority>])	Establece el valor y la prioridad para la propiedad especificada
<style>	Atributo que permite modificar un parámetro de estilo particular en notación de objeto.atributo

Nota: Computed Styles son los estilos por defecto del navegador. Se puede acceder a ellos a través de document.defaultView.getComputedStyle (<element>)

Control de visualización con JavaScript

- Acceso directo a elementos del DOM
  - querySelector (selector). Devuelve el primer elemento que coincide con el selector indicado

```
//returns the element with ID="myheader"
document.querySelector('#myheader')

//returns the P with class="description" element
document.querySelector('p.description')

//returns the 1st image within container "#content"
document.querySelector('#content img:nth-of-type(1)')

//selects the checked radio button within "#myform"
document.querySelector('#myform input[type="radio"]:checked')

//returns either element "#biography" or "#gallery", depending on
which one is found first within document tree
document.querySelector('#biography, #gallery')
```

Control de visualización con JavaScript

- Acceso directo a elementos del DOM
  - querySelectorAll (selector). Devuelve todos los elementos que coinciden con el selector indicado

```
//returns all elements with class="mygroup"
document.querySelectorAll('.mygroup')

//returns the default selected option within each SELECT menu
document.querySelectorAll('option[selected="selected"]')

//returns the first cell within each table row of "mytable"
document.querySelectorAll('#mytable tr>td:nth-of-type(1)')

//returns both elements "#biography" and "#gallery" (inclusive)
document.querySelectorAll('#biography, #gallery')
```

Control de visualización con JavaScript

- Relación de propiedades CSS – DOM (1)

DOM	CSS	DOM	CSS
background	background	borderRightWidth	border-right-width
backgroundAttachment	background-attachment	borderSpacing	border-spacing
backgroundColor	background-color	borderStyle	border-style
backgroundImage	background-image	borderTop	border-top
backgroundPosition	background-position	borderTopColor	border-top-color
backgroundRepeat	background-repeat	borderTopStyle	border-top-style
border	border	borderTopWidth	border-top-width
borderBottom	border-bottom	borderWidth	border-width
borderBottomColor	border-bottom-color	captionSide	caption-side
borderBottomStyle	border-bottom-style	clear	clear
borderBottomWidth	border-bottom-width	color	color
borderCollapse	border-collapse	cssFloat	float
borderColor	border-color	cursor	cursor
borderLeft	border-left	direction	direction
borderLeftColor	border-left-color	display	display
borderLeftStyle	border-left-style	emptyCells	empty-cells
borderLeftWidth	border-left-width	font	font
borderRight	border-right	fontFamily	font-family
borderRightColor	border-right-color	fontSize	font-size
borderRightStyle	border-right-style	fontStyle	font-style

Control de visualización con JavaScript

- Relación de propiedades CSS – DOM (2)

DOM	CSS	DOM	CSS
fontVariant	font-variant	outlineStyle	outline-style
fontWeight	font-weight	outlineWidth	outline-width
height	height	overflow	overflow
letterSpacing	letter-spacing	padding	padding
lineHeight	line-height	paddingBottom	padding-bottom
listStyle	list-style	paddingLeft	padding-left
listStyleImage	list-style-image	paddingRight	padding-right
listStylePosition	list-style-position	paddingTop	padding-top
listStyleType	list-style-type	tableLayout	table-layout
margin	margin	textAlign	text-align
marginBottom	margin-bottom	textDecoration	text-decoration
marginLeft	margin-left	textIndent	text-indent
marginRight	margin-right	textShadow	text-shadow
marginTop	margin-top	textTransform	text-transform
maxHeight	max-height	visibility	visibility
maxWidth	max-width	whiteSpace	whitespace
minHeight	min-height	width	width
minWidth	min-width	wordSpacing	word-spacing
outline	outline	zIndex	z-index
outlineColor	outline-color		

Java, Javascript y XML

- El lenguaje XML es un estándar W3C para el intercambio de datos entre sistemas.
- XML se usa para definir lenguajes de marcas propios y extensibles.
- En el caso de las aplicaciones J2EE, se utiliza como formato de comunicación e intercambio entre componentes.
- Analizaremos las distintas formas de generar documentos XML y sobre todo leer documentos XML e interpretarlos en Java.

Java, Javascript y XML

- Resumen de XML
  - Documento XML = Doc. bien formado.
    - Un único elemento raíz
    - Todas las etiquetas deben cerrarse
    - Todos los atributos deben entrecomillarse
    - Sólo se pueden utilizar caracteres Unicode
  - Vista de árbol con nodos de varios tipos:
    - Root (/)
    - Element (<etiqueta var="val"/>
    - Text (El contenido de una etiqueta, por ejemplo)
    - Comment (<!-- Un comentario -->)
    - Processing instruction (<?cocoon-process … ?>

Java, Javascript y XML

- Ejemplo de XML y representación en árbol

```
<sentence>
   The &projectName; <![CDATA[<i>project</i>]]> is
   <?editor: red><bold>important</bold><?editor: normal>.
</sentence>
```

```
+ ELEMENT: sentence
   + TEXT: The
   + ENTITY REF: projectName
      + COMMENT: The latest name we're using
      + TEXT: Eagle
   + CDATA: <i>project</i>
   + TEXT: is
   + PI: editor: red
   + ELEMENT: bold
      + TEXT: important
   + PI: editor: normal
```

Java, Javascript y XML

- Escribir documentos XML en Java
  - El procedimiento de creación de documentos XML consiste en generar un stream con la propia estructura del documento que se desea
  - El algoritmo debe generar etiquetas y atributos, e intercalar los contenidos de forma adecuada dentro de la estructura
  - La obtención de una cadena o un stream con un documento XML es independiente de la creación de un objeto XML que satisfaga las características de los distintos modelos de objetos que pueden existir.

Java, Javascript y XML

- Ejemplo:
  - Obtener los 10 primeros elementos de la serie de Fibonacci en formato XML.

```
<?xml version="1.0"?>
<Fibonacci_Numbers>
<fibonacci>1</fibonacci>
<fibonacci>1</fibonacci>
<fibonacci>2</fibonacci>
<fibonacci>3</fibonacci>
<fibonacci>5</fibonacci>
<fibonacci>8</fibonacci>
<fibonacci>13</fibonacci>
<fibonacci>21</fibonacci>
<fibonacci>34</fibonacci>
<fibonacci>55</fibonacci>
</Fibonacci_Numbers>
```

Java, Javascript y XML

- Programa para el cálculo:

```
import java.math.BigInteger;

public class FibonacciNumbers {

  public static void main(String[] args) {

    BigInteger low = BigInteger.ONE;
    BigInteger high = BigInteger.ONE;

    for (int i = 1; i <= 10; i++) {
      System.out.println(low);
      BigInteger temp = high;
      high = high.add(low);
      low = temp;
    }
  }
}
```

Java, Javascript y XML

- Programa con resultado en XML

```
for (int i = 1; i <= 10; i++) {
    System.out.println(low);
    BigInteger temp = high;
    high = high.add(low);
    low = temp;
}
```

```
import java.math.BigInteger;

public class FibonacciXML {

    public static void main(String[] args) {
        BigInteger low = BigInteger.ONE;
        BigInteger high = BigInteger.ONE;

        System.out.println("<?xml version=\"1.0\"?>");
        System.out.println("<Fibonacci_Numbers>");
        for (int i = 0; i < 10; i++) {
            System.out.print(" <fibonacci>");
            System.out.print(low);
            System.out.println("</fibonacci>");
            BigInteger temp = high;
            high = high.add(low);
            low = temp;
        }
        System.out.println("</Fibonacci_Numbers>");
    }
}
```

Java, Javascript y XML

- Leer documentos XML y procesarlos en Java
  - La lectura y procesado de documentos XML en Java son más complejos que la escritura.
  - El problema radica en la necesidad de recorrer el documento, validarlo, y acceder a los elementos de datos en un orden determinado o bajo una demanda del programa concreta.
  - El proceso de lectura y validación de un documento XML se conoce como "parsing".
  - Afortunadamente, este proceso no debe ser programado (existen muchas librerías que implementan el parsing de documentos XML y permiten su control a través de un API).

Java, Javascript y XML

- Las tareas que realiza el parser son:
  - Traducir el documento a Unicode.
  - Ensamblar las partes del documento separadas en múltiples Entities.
  - Resolver referencias a caracteres.
  - Asimilar secciones CDATA.
  - Comprobar que el documento esté bien formado.
  - Mantener una lista de espacios de nombres y su ámbito para cada Element.
  - Validar el documento contra un DTD o un esquema.
  - Asociar Entities no parseadas con URLs particulares y notaciones.
  - Asignar tipos a atributos.

Java, Javascript y XML

- Hay 3 formas básicas de parsear un documento XML:
  - Crear explícitamente un procedimiento que lea el documento y maneje las etiquetas según la estructura y contenidos del documento.
  - Utilizar una librería que, tras leer el documento, construya una representación del mismo en forma de árbol y proporcione un API. Este es un parser tipo DOM.
  - Utilizar una librería que, a medida que recorre el documento y encuentra símbolos, avise al programa a través de ciertos eventos (encontrar un inicio de etiqueta, un fin de etiqueta, un atributo, etc.). Este es un parser basado en eventos.

Java, Javascript y XML

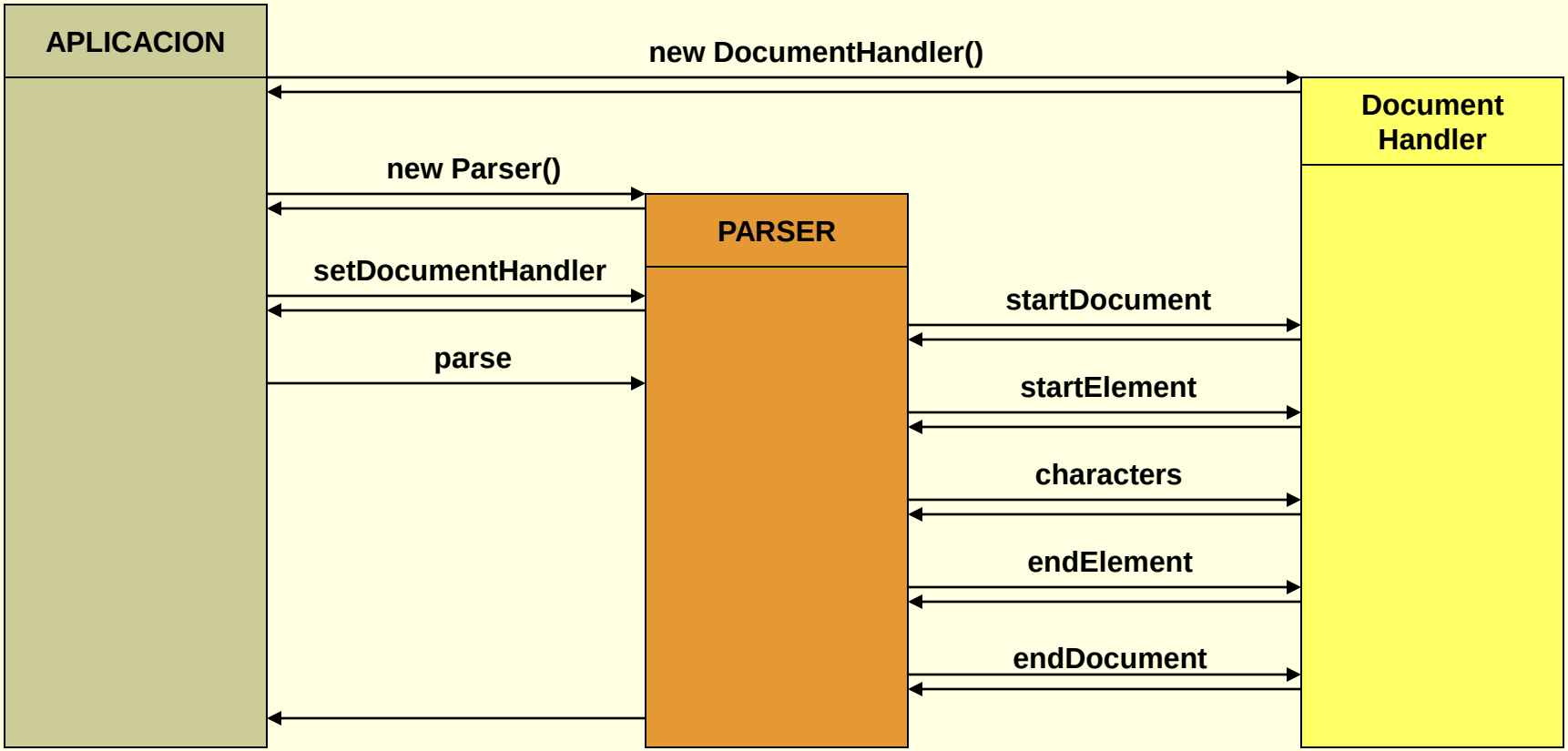
- Los parsers más habituales para Java son:
  - SAX (Simple API for XML)
    - El más completo y correcto. Basado en manejadores de eventos. Sólo permiten lectura.
  - DOM (Document Object Model)
    - Crea un árbol del documento. Es de lectura-escritura. Permite la creación de documentos.
  - JAXP
    - Aúna SAX y DOM. Parte estándar de Java 1.4.
  - JDOM
    - Java-native tree-based API. Uso simplificado.
  - JAXB
    - Permite una integración directa con Beans
- Cada parser publica su propio API

SAX

- Ventajas
  - Es simple
  - Es rápido
  - Puede analizar ficheros de cualquier tamaño por su característica de secuencialidad.
  - Es útil cuando se desea construir una estructura propia
  - Es útil cuando se necesita un pequeño subconjunto de los datos
- Inconvenientes
  - No hay posibilidad de acceso aleatorio (directo)
  - Búsquedas complejas difíciles de implementar
  - DTD no disponible en versión 1.0 (sí en la 2.0)
  - Información léxica no disponible (comentarios no disponibles, …)
  - Es de sólo lectura. No permite crear documentos.

SAX

- Estructura de procesamiento:



SAX

- Ejemplo simple 1

```
<?xml version="1.0"?>
<books>
  <book category="reference">
    <author>Nigel Rees</author>
    <title>Sayings of the Century</title>
    <price>8.95</price>
  </book>
  <book category="fiction">
    <author>Evelyn Waugh</author>
    <title>Sword of Honour</title>
    <price>12.99</price>
  </book>
  <book category="fiction">
    <author>Herman Melville</author>
    <title>Moby Dick</title>
    <price>8.99</price>
  </book>
</books>
```

```
import org.xml.sax.*;
import org.xml.sax.helpers.*;

public class cuentaLibrosHandler extends DefaultHandler
{
   public int cuenta = 0;

   public static void main (String args[]) throws Exception {
      (new cuentaLibrosHandler()).cuentaLibros(args[0]);
   }

   public void cuentaLibros(String fichero) throws Exception {
      XMLReader p = new org.apache.xerces.parsers.SAXParser();
      p.setContentHandler(new cuentaLibrosHandler());
      p.Parse(new InputSource(fichero));
   }

   public void startElement(String uri, String localName,
                  String qName, Attributes attributes)
                  throws SAXException {
      if (localName.equalsIgnoreCase("book")) cuenta++;
   }

   public void endDocument() throws SAXException {
      System.out.println("Hay " + cuenta + " libros.");
   }
}
```

SAX

- Ejemplo simple 2

```
<?xml version="1.0"?>
<books>
  <book category="reference">
    <author>Nigel Rees</author>
    <title>Sayings of the Century</title>
    <price>8.95</price>
  </book>
  <book category="fiction">
    <author>Evelyn Waugh</author>
    <title>Sword of Honour</title>
    <price>12.99</price>
  </book>
  <book category="fiction">
    <author>Herman Melville</author>
    <title>Moby Dick</title>
    <price>8.99</price>
  </book>
</books>
```

```
import javax.xml.parsers.*;
import org.xml.sax.*;
import org.xml.sax.helpers.*;

public class cuentaLibrosHandler extends DefaultHandler
{
   public int cuenta = 0;

   public static void main (String args[]) throws Exception {
      (new cuentaLibrosHandler()).cuentaLibros(args[0]);
   }

   public void cuentaLibros(String fichero) throws Exception {
      DefaultHandler handler = new cuentaLibrosHandler();
      SAXParserFactory factory = SAXParserFactory.newInstance();
      try {
         SAXParser parser = factory.newSAXParser();
         parser.parse(fichero, handler);
      } catch(Exception e) {
         String errorMessage =
         “Error parsing " + fichero + ": " + e;
         System.err.println(errorMessage);
         e.printStackTrace();
      }
   }

   public void startElement(String uri, String localName,
                  String qName, Attributes attributes)
                  throws SAXException {
      if (localName.equalsIgnoreCase(“book")) cuenta++;
   }

   public void endDocument() throws SAXException {
      System.out.println("Hay " + cuenta + " libros.");
   }
}
```

SAX

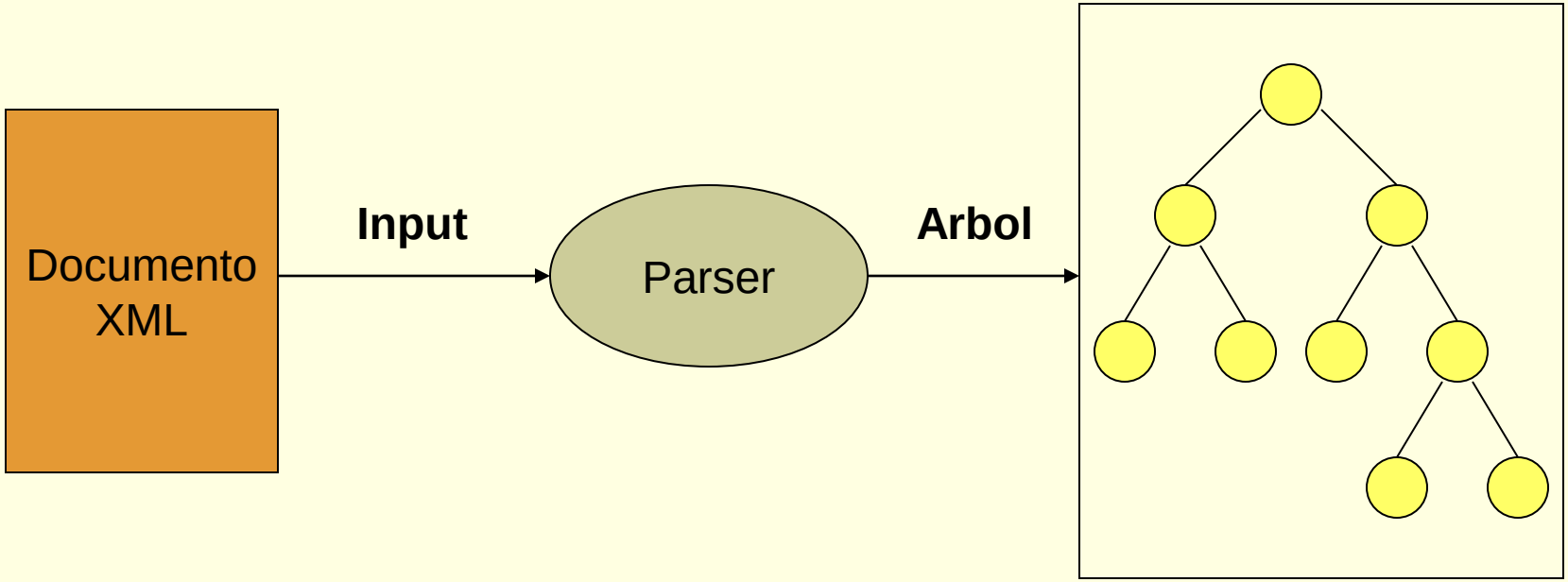
- DocumentHandler interface
  - Document events
    - startDocument()
    - endDocument()
  - Element events
    - startElement(String name, AttributeList attList)
    - endElement(String name)
  - Character Data
    - characters(char[] chars, int start, int len)
    - ignorableWhitespace(char[] chars, int start, int len)
  - Processing instructions
    - processingInstruction(String name, String data)

DOM

- Es el segundo mayor estándar API para parsers XML, y el primero en árboles.
- Basado en el objeto *Document* que contiene el documento XML original en forma de árbol.
- Se accede a los contenidos del documento invocando métodos del objeto *Document* o de los objetos que éste contiene.
- Ventajas
  - Es especialmente útil cuando se desea procesar el documento usando accesos aleatorios (directos).
  - Se trata de una API de lectura-escritura. Los documentos creados ya están bien formados.
  - Abstrae el contenido de la gramática.
  - Se asemeja a estructuras jerárquicas y de bases de datos.
- Inconvenientes
  - El consumo de memoria es mayor debido a que el árbol se almacena en memoria cuando se carga.

DOM

- Estructura de procesamiento:



DOM – Ejemplo: recorrer un árbol

- Programa en Java que, utilizando DOM, recorre un fichero XML y escribe el fichero con un formato indentado en la salida estándar.

```
import org.w3c.dom.*;
import org.apache.xerces.parsers.DOMParser;

public class IndentingParser
{
    static String displayStrings[] = new String[1000];
    static int numberDisplayLines = 0;

    public static void displayDocument(String uri) {...}

    public static void display(Node node, String indent) {...}

    public static void main(String args[]) {...}
}
```

DOM – Ejemplo: recorrer un árbol

```
public static void main(String args[])
{
    displayDocument(args[0]);

    for(int loopIndex = 0; loopIndex < numberDisplayLines; loopIndex++){
        System.out.println(displayStrings[loopIndex]);
    }
}
```

```
public static void displayDocument(String uri)
{
    try {
        DOMParser parser = new DOMParser();
        parser.parse(uri);
        Document document = parser.getDocument();

        display(document, "");
    } catch (Exception e) { e.printStackTrace(System.err); }
}
```

DOM – Ejemplo: recorrer un árbol

```
public static void display(Node node, String indent)
{
    if (node == null) { return; }

    int type = node.getNodeType();

    switch (type) {
        case Node.DOCUMENT_NODE: {...}
        case Node.ELEMENT_NODE: {...}
        case Node.CDATA_SECTION_NODE: {...}
        case Node.TEXT_NODE: {...}
        case Node.PROCESSING_INSTRUCTION_NODE: {...}
    }

    if (type == Node.ELEMENT_NODE) {
        displayStrings[numberDisplayLines] = indent.substring(0, indent.length() - 4);
        displayStrings[numberDisplayLines] += "</";
        displayStrings[numberDisplayLines] += node.getNodeName();
        displayStrings[numberDisplayLines] += ">";
        numberDisplayLines++;
        indent += "    ";
    }
}
```

DOM – Ejemplo: recorrer un árbol

```
case Node.DOCUMENT_NODE: {
      displayStrings[numberDisplayLines] = indent;
      displayStrings[numberDisplayLines] +=
        "<?xml version=\"1.0\" encoding=\""+
        "UTF-8" + "\"?>";
      numberDisplayLines++;
      display(((Document)node).getDocumentElement(), "");
      break;
}
```

```
case Node.TEXT_NODE: {
      displayStrings[numberDisplayLines] = indent;
      String newText = node.getNodeValue().trim();
      if(newText.indexOf("\n") < 0 && newText.length() > 0) {
            displayStrings[numberDisplayLines] += newText;
            numberDisplayLines++;
      }
      break;
}
```

DOM – Ejemplo: recorrer un árbol

```
case Node.CDATA_SECTION_NODE: {
      displayStrings[numberDisplayLines] = indent;
      displayStrings[numberDisplayLines] += "<![CDATA[";
      displayStrings[numberDisplayLines] += node.getNodeValue();
      displayStrings[numberDisplayLines] += "]]>";
      numberDisplayLines++;
      break;
}
```

```
case Node.PROCESSING_INSTRUCTION_NODE: {
      displayStrings[numberDisplayLines] = indent;
      displayStrings[numberDisplayLines] += "<?";
      displayStrings[numberDisplayLines] += node.getNodeName();
      String text = node.getNodeValue();
      if (text != null && text.length() > 0) {
            displayStrings[numberDisplayLines] += text;
      }
      displayStrings[numberDisplayLines] += "?>";
      numberDisplayLines++;
      break;
}
```

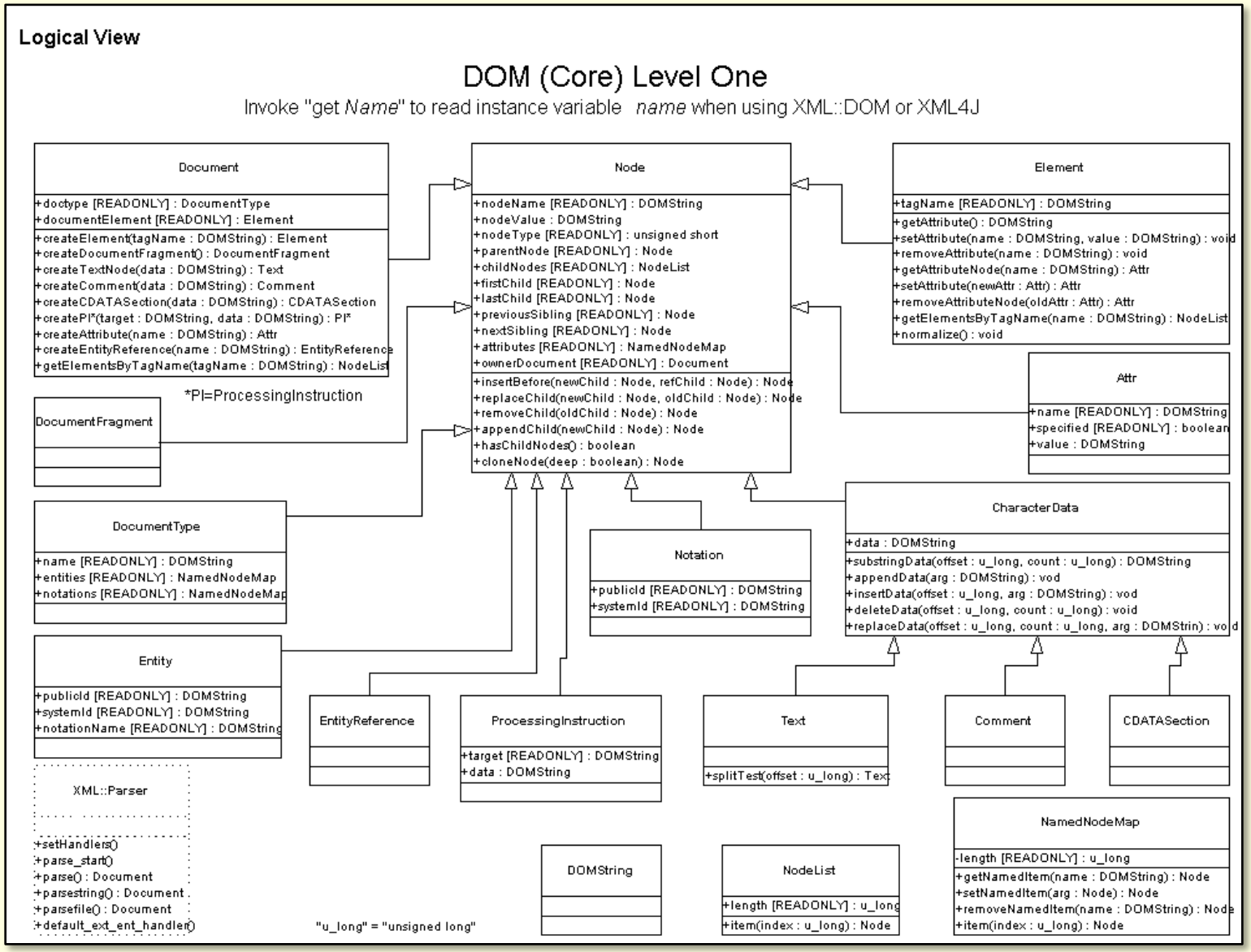
DOM – Ejemplo: recorrer un árbol

```
case Node.ELEMENT_NODE: {
        displayStrings[numberDisplayLines] = indent;
        displayStrings[numberDisplayLines] += "<“ + node.getNodeName();

        int length = (node.getAttributes() != null) ? node.getAttributes().getLength() : 0;
        Attr attributes[] = new Attr[length];
        for (int loopIndex = 0; loopIndex < length; loopIndex++)
                attributes[loopIndex] = (Attr)node.getAttributes().item(loopIndex);

        for (int loopIndex = 0; loopIndex < attributes.length; loopIndex++) {
                Attr attribute = attributes[loopIndex];
                displayStrings[numberDisplayLines] += " ";
                displayStrings[numberDisplayLines] += attribute.getNodeName();
                displayStrings[numberDisplayLines] += "=\"";
                displayStrings[numberDisplayLines] += attribute.getNodeValue();
                displayStrings[numberDisplayLines] += "\"";
        }
        displayStrings[numberDisplayLines] += ">";

        numberDisplayLines++;

        NodeList childNodes = node.getChildNodes();
        if (childNodes != null) {
                length = childNodes.getLength();
                indent += "    ";
                for (int loopIndex = 0; loopIndex < length; loopIndex++ )
                        display(childNodes.item(loopIndex), indent);
        }
        break;
}
```

JavaScript API para XML

- Para el desarrollo de aplicaciones web basadas en el uso de JavaScript se decidió crear un interfaz XML.
- Se definió un API para XML inicial (SAX). El proceso de parsing en SAX es:
  - Rápido y ligero
  - Secuencial
  - Unidireccional (no puede ir hacia atrás)
  - Basado en eventos
- Posteriormente surgió el DOM, que además de parsear permitía una representación en forma de árbol que podía ser navegable desde un lenguaje de programación.

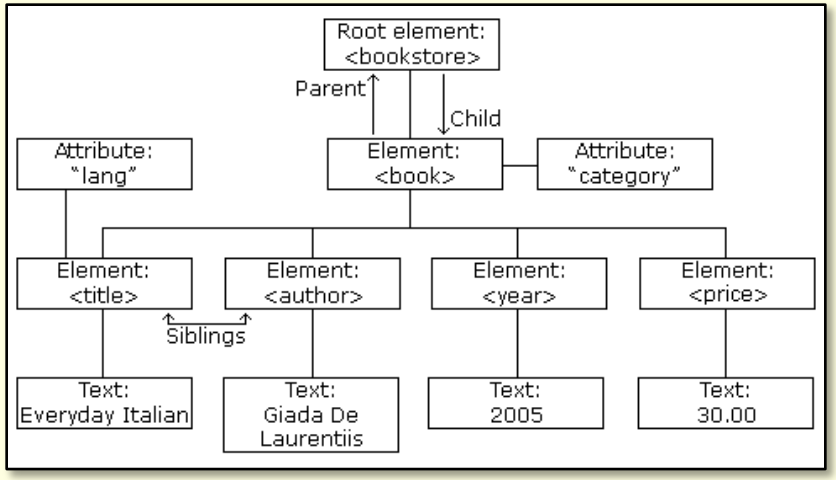
JavaScript API para XML (DOM)

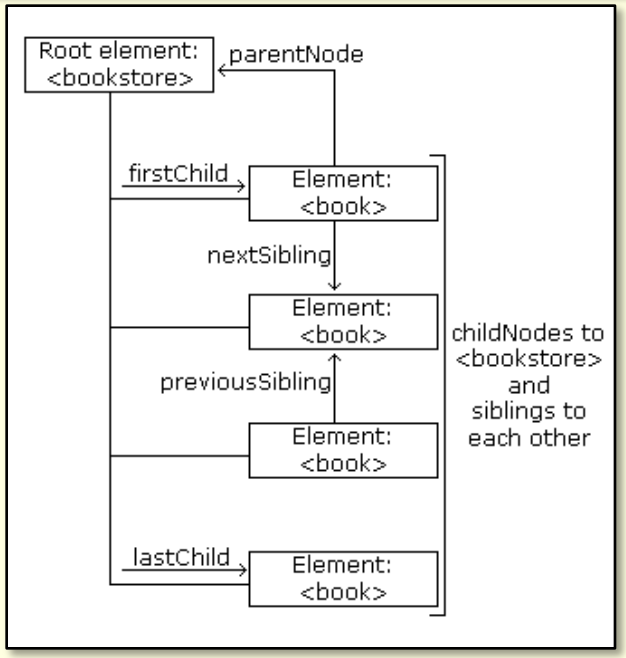


JavaScript API para XML (DOM)

- Ejemplo: bookstore

```
<bookstore>
   <book category="Novel">
      <title lang="English">
         Everyday Italian
      </title>
      <author>Giada de Laurentiis</author>
      <year>2005</year>
      <price>30.00</price>
   </book>
   <book category="...">
      ...
   </book>
   ...
</bookstore>
```





JavaScript API para XML (DOM)

- Nodos de la jerarquía (árbol)
  - **Document** — The very top-level node to which all other nodes are attached
  - **DocumentType** — The object representation of a DTD reference using the syntax <!DOCTYPE >, such as <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">. It cannot contain child nodes.
  - **DocumentFragment** — Can be used like a Document to hold other nodes
  - **Element** — Represents the contents of a start tag and end tag, such as <tag></tag> or <tag/>. This node type is the only one that can contain attributes as well as child nodes.
  - **Attr** — Represents an attribute name-value pair. This node type cannot have child nodes.
  - **Text** — Represents plain text in an XML document contained within start and end tags or inside of a CData Section. This node type cannot have child nodes.
  - **CDataSection** — The object representation of <![CDATA[ ]]>. This node type can have only text nodes as child nodes.
  - **Entity** — Represents an entity definition in a DTD, such as <!ENTITY foo "foo">. This node type cannot have child nodes.
  - **EntityReference** — Represents an entity reference, such as ". This node type cannot have child nodes.
  - **ProcessingInstruction** — Represents a PI. This node type cannot have child nodes.
  - **Comment** — Represents an XML comment. This node type cannot have child nodes.
  - **Notation** — Represents notation defined in a DTD. This is rarely used.

JavaScript API para XML (DOM)

- Propiedades y Métodos de los nodos

Propiedades	Métodos
baseURI	appendChild()
childNodes	cloneNode()
firstChild	compareDocumentPosition()
lastChild	getFeature(feature,version)
localName	getUserData(key)
namespaceURI	hasAttributes()
nextSibling	hasChildNodes()
nodeName	insertBefore()
nodeType	isDefaultNamespace(URI)
nodeValue	isEqualNode()
ownerDocument	isSameNode()
parentNode	isSupported(feature,version)
prefix	lookupNamespaceURI()
previousSibling	lookupPrefix()
textContent	normalize()
text	removeChild()
xml	replaceChild()
	setUserData(key,data,handler)

Javascript y XML (IE)

- Librería MSXML (DOM), basada en ActiveX
- Crear un objeto XML
  - var oXmlDom = new ActiveXObject("Microsoft.XmlDom");
- Rellenar un objeto XML
  - oXmlDom.loadXML("<root><child/></root>");
- Carga síncrona:
  - oXmlDom.async = false;
  - oXmlDom.load("test.xml");
- Carga asíncrona:
  - evento onreadystatechange
  - propiedad readyState (0:Null, 1:Cargando, 2:Cargado, 3:Procesando, 4:Ready)
- Nodo raíz:
  - oXmlDom.documentElement

Javascript y XML

- Ejemplo 1: IE – Cargar fichero xml

```
<html><head>
<script language="JavaScript">
var xmlDoc
function loadXMLDoc(dname) {
  if (window.ActiveXObject){ //IExplorer
    var xmlDoc=new ActiveXObject("Microsoft.XMLDOM");
  }
  xmlDoc.async="false"
  xmlDoc.load(dname)

  return xmlDoc
}
</script>
</head>

<body>
<form><input type="button" value="Get data" onclick="var mDoc = loadXMLDoc('test.xml');
document.getElementById('recibido').innerHTML = mDoc.documentElement.tagName;"/></form>

<hr>
<div id="recibido">Control de recepción del XML</div>
<hr>

</body></html>
```

Javascript y XML

- Ejemplo 2: FF – Cargar fichero xml

```
<html><head>
<script language="JavaScript">
var xmlDoc
function loadXMLDoc(dname) {
  if (document.implementation &&
  document.implementation.createDocument) {
  xmlDoc=document.implementation.createDocument("","",null);
  xmlDoc.async="false"
  xmlDoc.load(dname);
  xmlDoc.onload= readXML;
  }
}

function readXML() {
  document.getElementById('recibido').innerHTML =
    xmlDoc.documentElement.tagName;
}
</script>
</head>

<body>
<form><input type="button" value="Get data" onclick="loadXMLDoc('test.xml');"/></form>

<hr>
<div id="recibido">Control de recepción del XML</div>
<hr>

</body></html>
```

Javascript y XML

- Ejemplo 3: Generar xml desde texto

```
<html><head>
<script language="JavaScript">
var xmlDoc
function loadXMLString(txt) {
  if (window.DOMParser) {
    parser=new DOMParser();
    xmlDoc=parser.parseFromString(txt,"text/xml");
  }
  else {
    xmlDoc=new ActiveXObject("Microsoft.XMLDOM");
    xmlDoc.async="false";
    xmlDoc.loadXML(txt);
  }
  return xmlDoc;
}
</script>
</head>

<body>
<form><input type="button" value="Get data" onclick="var mDoc =
loadXMLString('<test>Prueba</test>'); document.getElementById('recibido').innerHTML =
mDoc.documentElement.tagName;"/></form>

<hr>
<div id="recibido">Control de recepción del XML</div>
<hr>

</body></html>
```

Javascript y XML

- Ejemplo 4: Obtener xml desde servidor

```
<html><head>
<script language="JavaScript">
function loadXMLDoc(dname) {
  if (window.XMLHttpRequest) {
    xhttp=new XMLHttpRequest();
  }
  else {
    xhttp=new ActiveXObject("Microsoft.XMLHTTP");
  }
  xhttp.open("GET",dname,false);
  xhttp.send("");
  return xhttp.responseXML;
}
</script>
</head>

<body>
<form><input type="button" value="Get data" onclick="var mDoc =
loadXMLDoc('http://www.w3schools.com/dom/books.xml');
document.getElementById('recibido').innerHTML = mDoc.documentElement.tagName;"/></form>

<hr>
<div id="recibido">Control de recepción del XML</div>
<hr>

</body></html>
```

Nota: Firefox no permite llamadas XMLHttpRequest a objetos que no estén en el mismo servidor que la página que invoca.

Javascript y XML

- Ejemplo 5: Cargar y procesar XML

```
<html><head>
<script language="JavaScript">
function loadXMLDoc(dname) {
  if (window.XMLHttpRequest) {
    xhttp=new XMLHttpRequest();
  } else {
    xhttp=new ActiveXObject("Microsoft.XMLHTTP");
  }
  xhttp.open("GET",dname,false);
  xhttp.send("");
  return xhttp.responseXML;
}
function processXML(mDoc) {
  document.getElementById('recibido').innerHTML =
  mDoc.getElementsByTagName("title")[0].childNodes[0].nodeValue + "<br />" +
  mDoc.getElementsByTagName("author")[0].childNodes[0].nodeValue + "<br />" +
  mDoc.getElementsByTagName("year")[0].childNodes[0].nodeValue;
}
</script>
</head>

<body>
<form><input type="button" value="Get data" onclick="var mDoc =
loadXMLDoc('http://www.w3schools.com/dom/books.xml'); processXML(mDoc);"/></form>

<hr>
<div id="recibido">Control de recepción del XML</div>
<hr>

</body></html>
```

Javascript – HTTP Requests

- Peticiones (requests) HTTP iniciadas desde Javascript
- Evitan recargas completas de páginas, frames ocultos, etc.
- Microsoft creó el objeto XMLHTTPRequest para implementarlos
  - var oRequest = new ActiveXObject("Microsoft.XMLHTTP");
- Especificación de la petición:
  - oRequest.open("get", "example.txt", false);
- Envío de la petición:
  - oRequest.send(null);
- Recepción de la respuesta:
  - oRequest.responseText
  - oRequest.responseXML.documentElement
- Trabajo en modo síncrono/asíncrono
  - Tercer parámetro del método open() [false→síncrono]
  - En modo asíncrono debería definirse la función onreadystatechange del mismo modo que en la carga de documentos XML asíncrona (readyState=4 → ok), que asegurá la recepción del contenido y permite su procesamiento, antes de invocar la función send().

AJAX

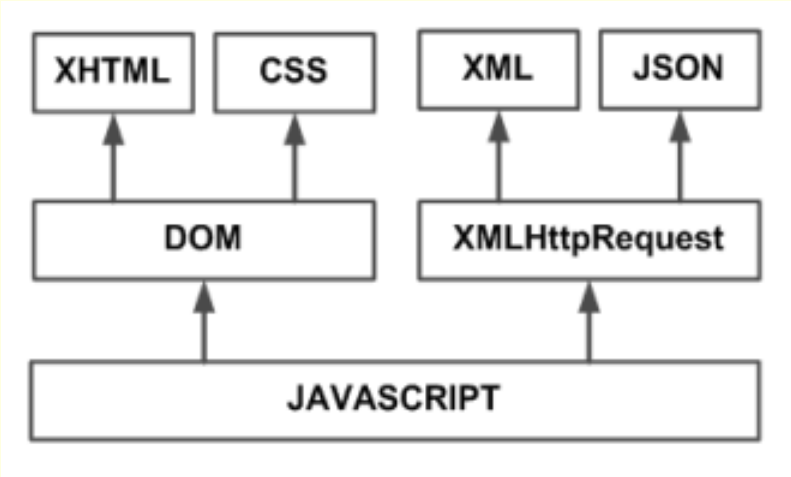
- AJAX = Asynchronous JavaScript + XML, que se puede traducir como "JavaScript asíncrono + XML".

- El término *AJAX* se acuñó por primera vez en el artículo "Ajax: A New Approach to Web Applications" publicado por Jesse James Garrett el 18 de Febrero de 2005.

- No es una tecnología en sí mismo: la unión de varias tecnologías que se desarrollan de forma autónoma y que se unen de formas nuevas y sorprendentes.

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.

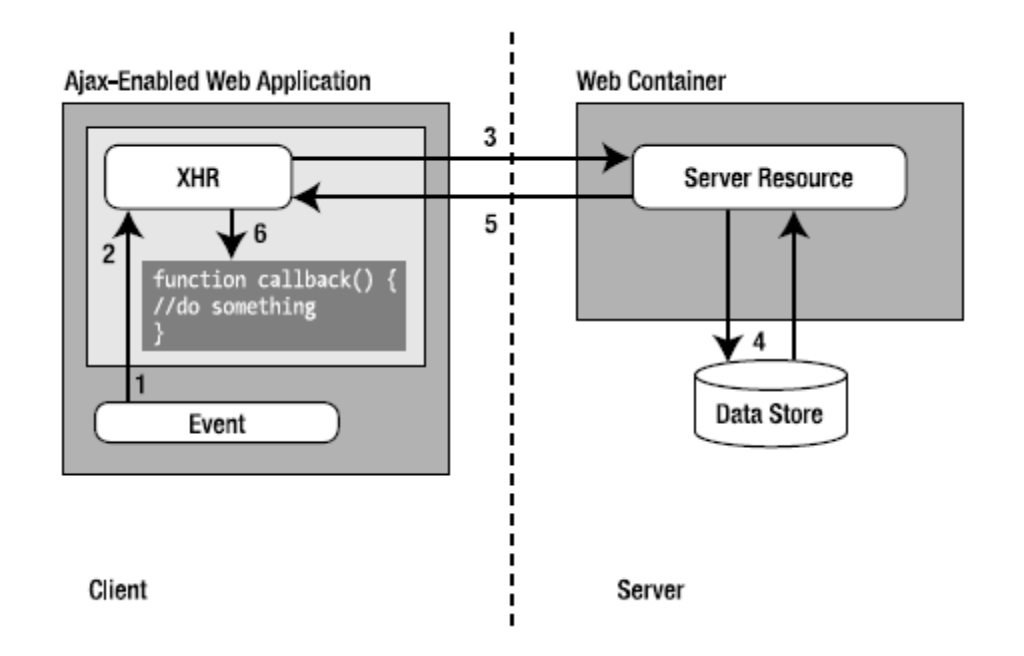


AJAX

- Principios de AJAX
  - Parte de la lógica de aplicación se puede desplazar al cliente ligero.
  - El servidor puede ofrece datos en lugar de contenido.
  - La interacción del usuario con la aplicación puede ser fluida y continua.
  - El uso de AJAX como patrón de programación requiere cierta disciplina.

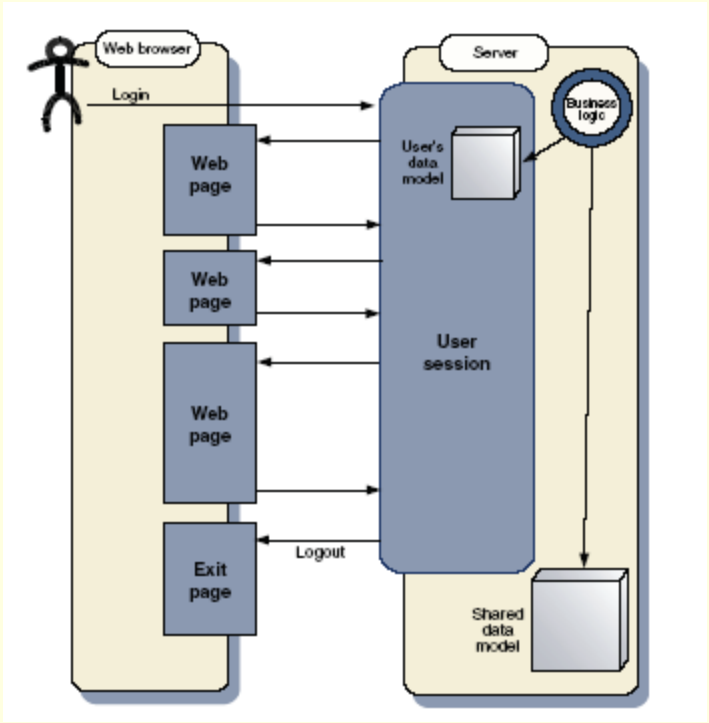
- Partes de AJAX (anteriores tecnologías)
  - Dynamic HTML
  - Remote Scripting
    - IFrames

AJAX



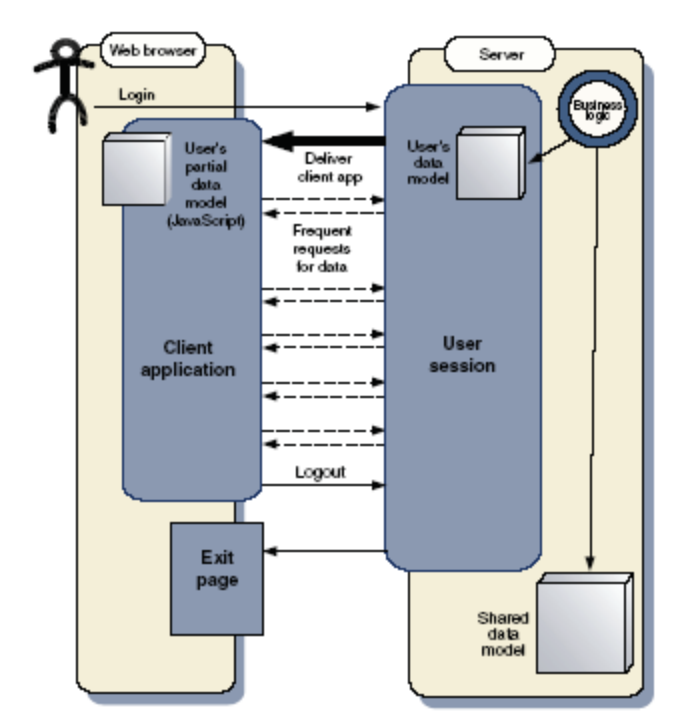
Funcionamiento de una aplicación AJAX

AJAX

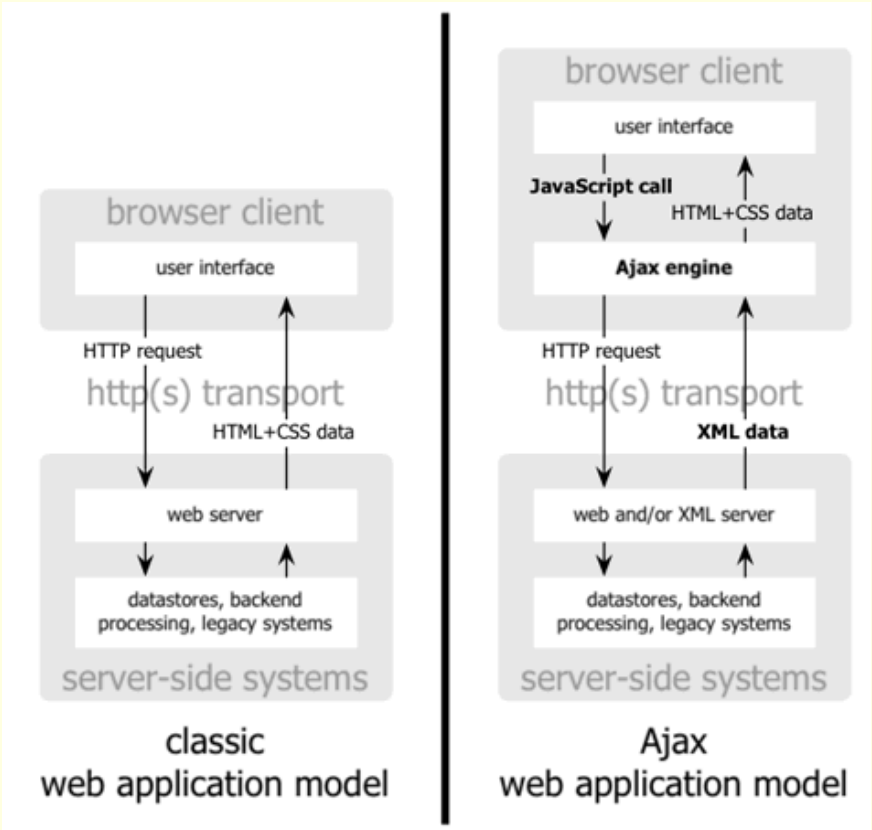


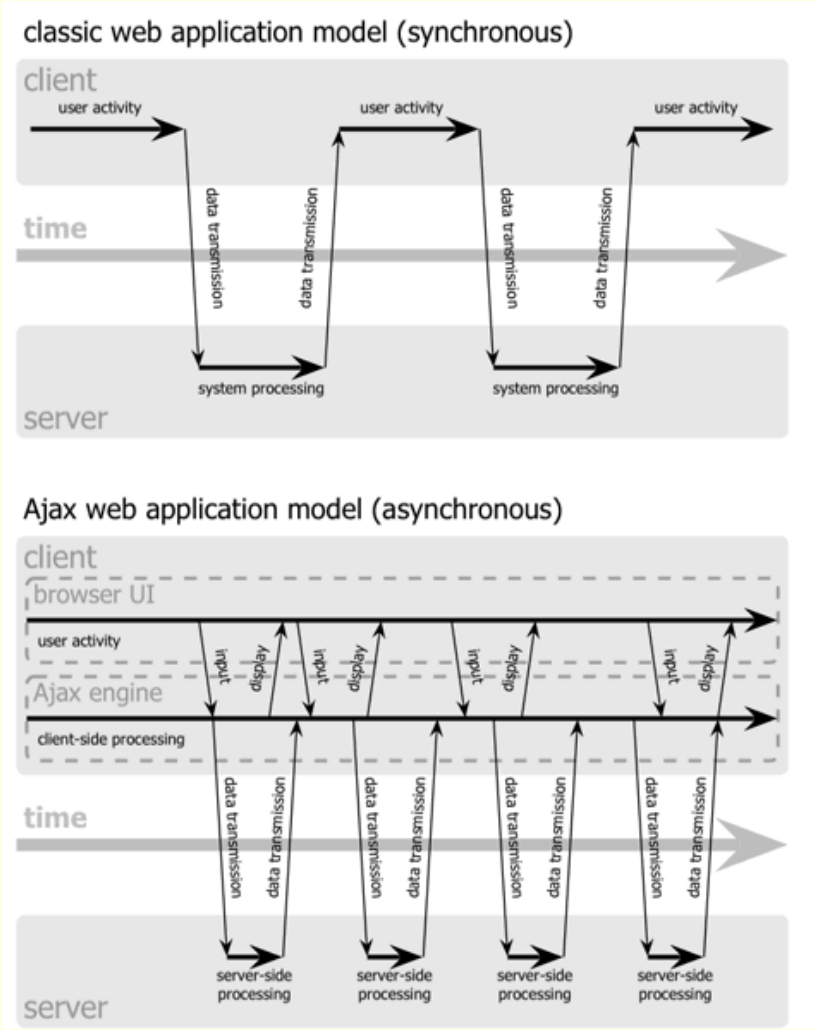
Aplicación Web Clásica

Aplicación Web con AJAX

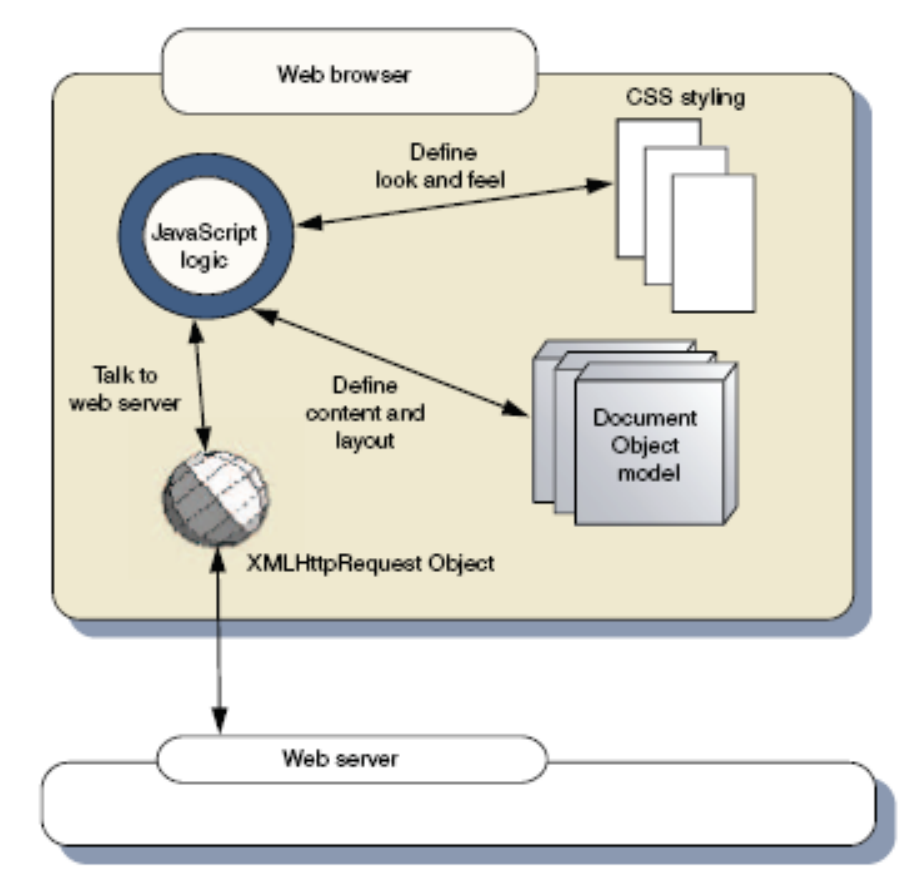


AJAX


browser client
user interface
HTTP request
http(s) transport
HTML+CSS data
web server
datastores, backend processing, legacy systems
server-side systems
classic web application model
browser client
user interface
JavaScript call
HTML+CSS data
Ajax engine
HTTP request
http(s) transport
XML data
web and/or XML server
datastores, backend processing, legacy systems
server-side systems
Ajax web application model

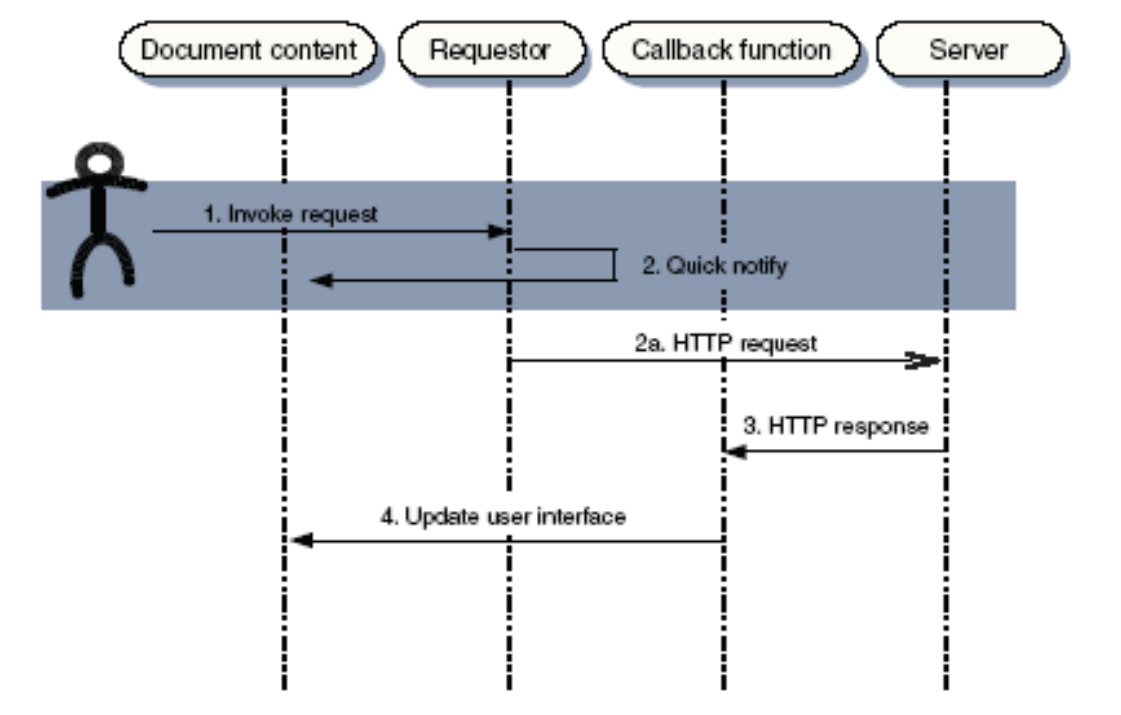

classic web application model (synchronous)
client
user activity
user activity
user activity
time
data transmission
data transmission
data transmission
data transmission
system processing
system processing
server
Ajax web application model (asynchronous)
client
browser UI
user activity
input
display
input
display
input
display
input
display
Ajax engine
client-side processing
time
data transmission
data transmission
data transmission
data transmission
data transmission
data transmission
data transmission
data transmission
server-side processing
server-side processing
server-side processing
server-side processing
server

AJAX



Tecnologías AJAX

AJAX



Secuencia de eventos en comunicación AJAX

Ejemplo AJAX (1)

```
<html><head>
<script language="JavaScript">
var xmlHttp

function getData() {
    xmlHttp=GetXmlHttpObject()
    if (xmlHttp==null) {
        alert ("Browser does not support HTTP Request")
        return
    }
    var url="http://www.bne.es/es/Inicio/index.html"
    xmlHttp.onreadystatechange=stateChanged
    xmlHttp.open("GET",url,true)
    xmlHttp.send(null)
}

function stateChanged()  {
    if (xmlHttp.readyState==4 || xmlHttp.readyState=="complete") {
        document.getElementById("recibidoHTML").innerHTML=xmlHttp.responseText
        document.getElementById("recibido").innerText=xmlHttp.responseText
    }
}
function GetXmlHttpObject() {
    var objXMLHttp=null
    if (window.XMLHttpRequest) {
        objXMLHttp=new XMLHttpRequest()
    }
    else if (window.ActiveXObject) {
        objXMLHttp=new ActiveXObject("Microsoft.XMLHTTP")
    }
    return objXMLHttp
}

</script>
</head>
```

Ejemplo AJAX (2)

```
<body>

<form>

    <input type="button" value="Get data" onclick="getData();"/>

</form>

<div id="recibido">Aquí recibiré el texto</div>

<hr> <hr> <hr>

<div id="recibidoHTML">Aquí recibiré el HTML</div>

</body></html>
```