

# JavaScript avançat: DOM.

IES Son Ferrer

# JS Avançat: DOM. Introducció.

---

XML va fer sorgir la necessitat de processar i manipular el contingut dels seus arxius mitjançant els llenguatges de programació tradicionals. Per aquest motiu, van sorgir algunes tècniques entre les quals es troba DOM.

**DOM** (***Document Object Model***) és una API de funcions per **manipular documents XML/XHTML/HTML**.

# JS Avançat: DOM. Introducció.

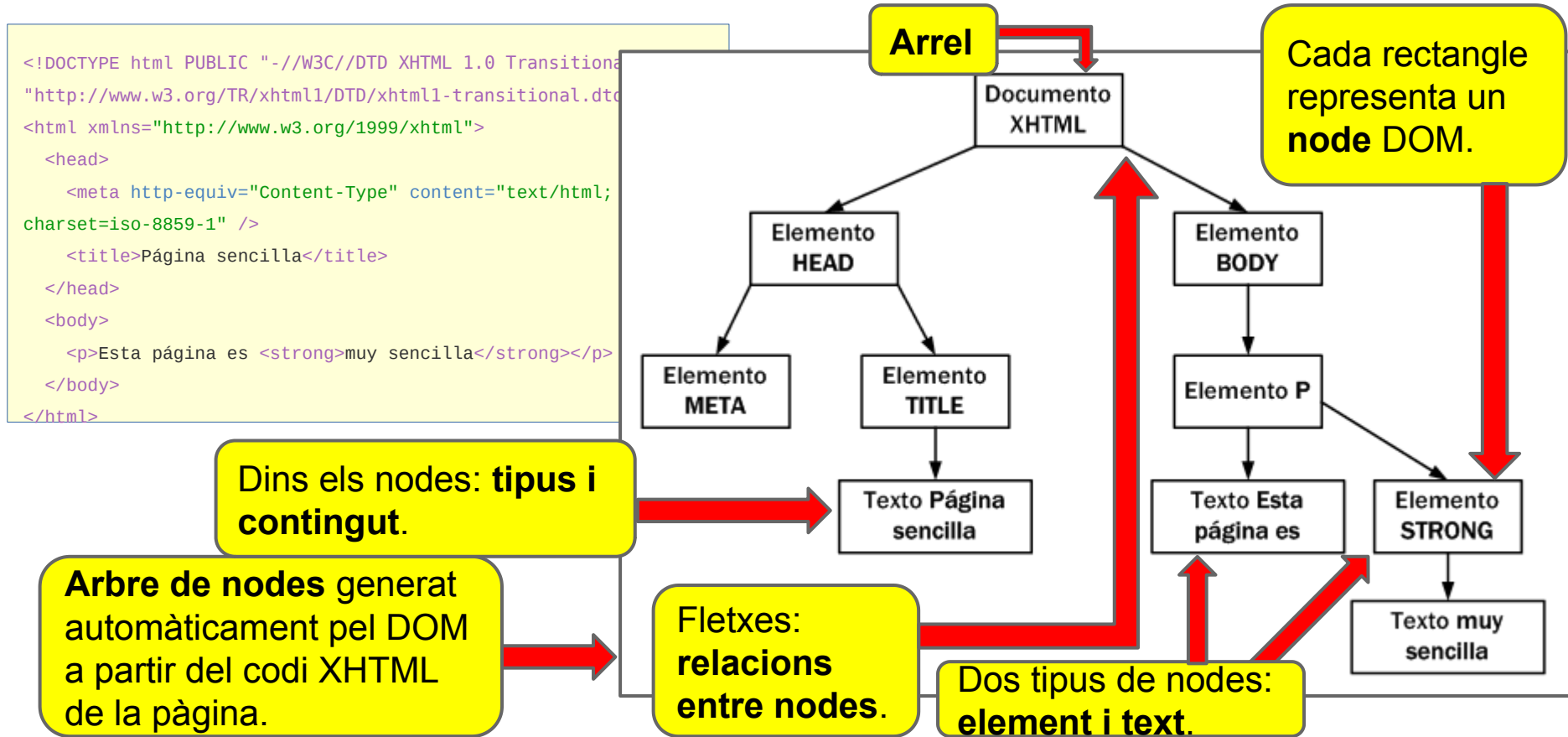
---

DOM transforma internament l'arxiu XML/XHTML/HTML original en una estructura més fàcil de manipular formada per una jerarquia de **nodes** interconnectats (=relacionats) en forma d'arbre (**arbre de nodes**).

Abans de poder utilitzar l'API de DOM és imprescindible que **la pàgina s'hagi carregat per complet** (sinó no està disponible l'arbre de nodes).

Les **funcions** que proporciona **DOM** permeten **afegir, eliminar, modificar i reemplaçar** qualsevol **node** de qualsevol document de forma senzilla.

# JS Avançat: DOM. Introducció.



# JS Avançat: DOM. Tipus de nodes.

---

En l'especificació completa de DOM hi ha 12 tipus de nodes, però habitualment utilitzarem només:

- **Document**, node arrel del que deriven tots els altres nodes de l'arbre.
- **DocumentType**, node que conté la representació del DTD emprat en la pàgina (indicat mitjançant **DOCTYPE**).
- **Element**, representa cadascuna de les etiquetes HTML (<etiqueta>...</etiqueta> o <etiqueta/>). Es tracta de l'únic node que pot tenir atributs i nodes fills.
- **Attr**, representar el parell **atribut=valor** contingut a les etiquetes HTML.
- **Text**, node que conté el text tancat per una etiqueta HTML (<etiqueta>Contingut</etiqueta>). També emmagatzema el contingut d'una secció CDATA.

# JS Avançat: DOM. Tipus de nodes.

---

En l'especificació completa de DOM hi ha 12 tipus de nodes, però habitualment utilitzarem només:

- **CDataSection**: és el node que representa una secció de tipus `<![CDATA[ ]]>`.  
CDATA ve de "*Character Data*", i no és més que una secció que ha de ser ignorada pel parser. És útil i s'usa quan tens caràcters invàlids pel XML com a contingut d'un node.
- **Comment**, representa els comentaris inclosos a la pàgina XHTML/XML/HTML.

Els altres tipus de nodes que no se fan servir habitualment:

**DocumentFragment, Entity, EntityReference, ProcessingInstruction i Notation.**

# JS Avançat: DOM. Tipus de nodes.

```
<?xml version="1.0"?>
```

```
<clientes>
```

```
<!-- El primer cliente -->
```

```
<cliente>
```

```
<nombre>Empresa SA</nombre>
```

```
<sector>Tecnologia</sector>
```

```
<notas><![CDATA[
```

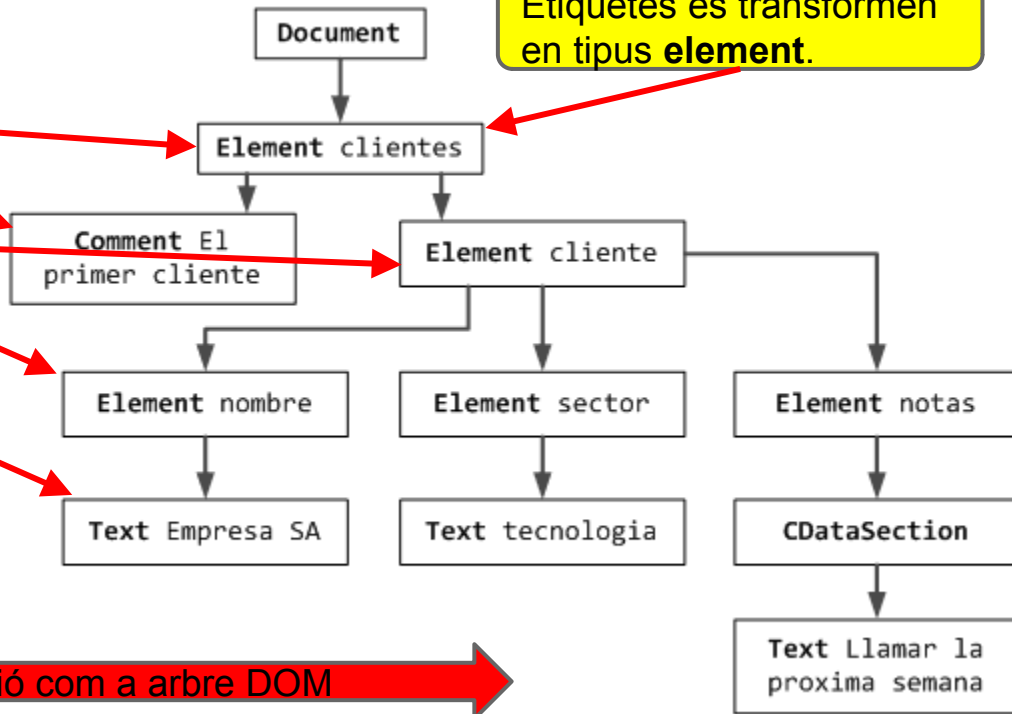
```
Llamar la proxima semana
```

```
]]></notas>
```

```
</cliente>
```

```
</clientes>
```

Etiquetes es transformen en tipus **element**.



Representació com a arbre DOM

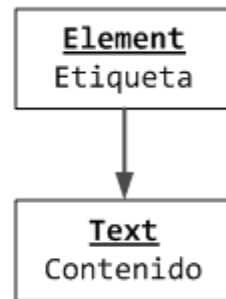
# JS Avançat: DOM. Tipus de nodes.

---

Cada **etiqueta simple** de tipus text es transforma en un parell de nodes:

- Un de **tipus Element** (que conté l'etiqueta en si).
- Un node fill d'aquest darrer, de **tipus Text**, que conté el contingut definit entre l'etiqueta d'obertura i la de tancament.

`<etiqueta>contenido</etiqueta>`



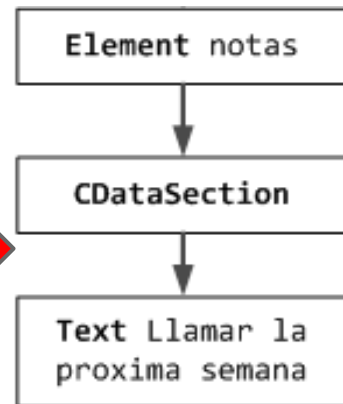


# JS Avançat: DOM. Tipus de nodes.

---

Si vos fixeu però, l'etiqueta `<notas>` es transforma en 3 nodes ja que conté una secció de tipus `CData` (que a la vegada es transforma en un node `Text` amb el contingut d'aquesta secció `CData`).

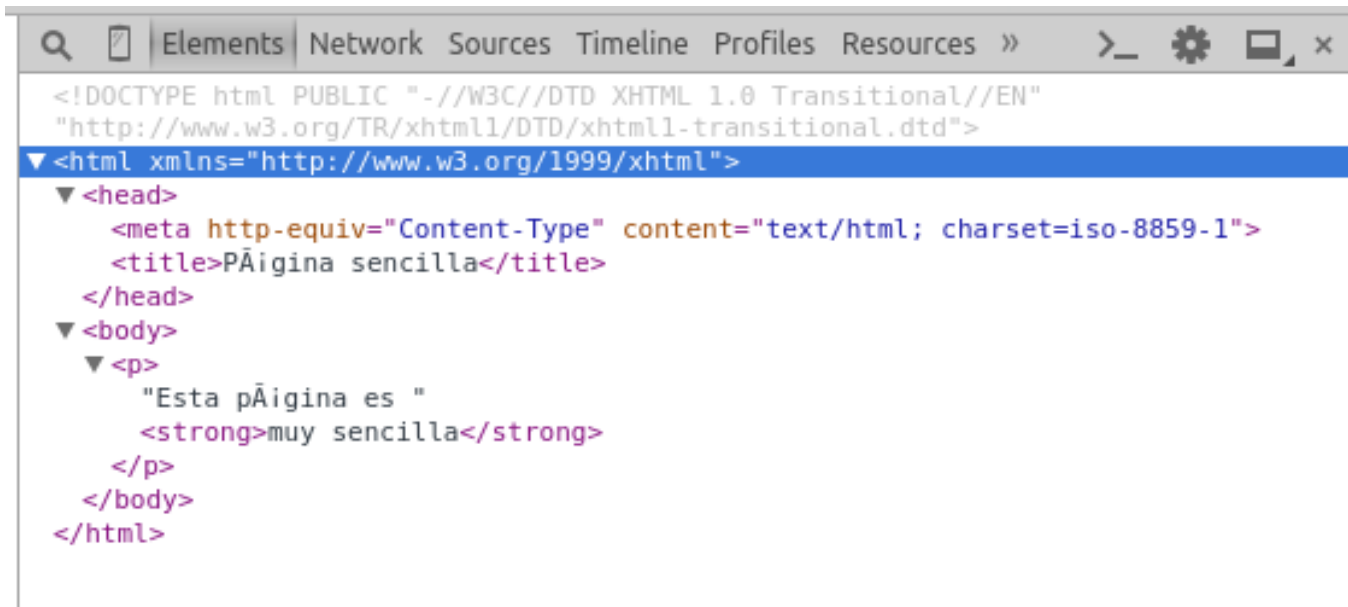
```
<notas><![CDATA[  
Llamar la proxima semana  
]]></notas>
```



# JS Avançat: DOM. Tipus de nodes.

---

Una bona idea és visualitzar el arbre de nodes creat pel DOM al navegador.  
Botó dret -> “Inspeccionar elemento” i es mostra Google Chrome Web Inspector (amb l’arbre de nodes de la pàgina!).



# JS Avançat: DOM. La interfície node.

---

JS defineix les següents constants per identificar els tipus de nodes:

`Node.ELEMENT_NODE = 1`

`Node.ATTRIBUTE_NODE = 2`

`Node.TEXT_NODE = 3`

`Node.CDATA_SECTION_NODE = 4`

`Node.ENTITY_REFERENCE_NODE = 5`

`Node.ENTITY_NODE = 6`

`Node.PROCESSING_INSTRUCTION_NODE =`

`7`

`Node.COMMENT_NODE = 8`

`Node.DOCUMENT_NODE = 9`

`Node.DOCUMENT_TYPE_NODE = 10`

`Node.DOCUMENT_FRAGMENT_NODE = 11`

`Node.NOTATION_NODE = 12`

# JS Avançat: DOM. La interfície node.

---

Ús pràctic per determinar els tipus de nodes:

```
alert(document.nodeType); // 9
alert(document.documentElement.nodeType); // 1
alert(document.nodeType == Node.DOCUMENT_NODE); // true
alert(document.documentElement.nodeType == Node.ELEMENT_NODE); //
true
```

# JS Avançat: DOM. La interfície node.

---

A més node té propietats i mètodes:

Propietat / Mètode	Valor retornat	Descripció
<code>nodeName</code>	<code>String</code>	El nom del node (no està definit per alguns tipus de node).
<code>nodeValue</code>	<code>String</code>	El valor del node (no està definit per alguns tipus de node).
<code>nodeType</code>	<code>Number</code>	Una de les 12 constants definides abans.
<code>ownerDocument</code>	<code>Document</code>	Referència del document al què pertany el node.
<code>firstChild</code>	<code>Node</code>	Referència al primer node de la llista de <code>childNodes</code>
<code>lastChild</code>	<code>Node</code>	Referència a l'últim node de la llista <code>childNodes</code>
<code>childNodes</code>	<code>NodeList</code>	Llista de tots els nodes fill del node actual.
<code>previousSibling</code>	<code>Node</code>	Referència del node germà anterior o <code>null</code> si aquest node és el primer germà.
<code>nextSibling</code>	<code>Node</code>	Referència del node germà següent o <code>null</code> si aquest node és l'últim germà.

# JS Avançat: DOM. La interfície node.

A més node té propietats i mètodes:

Propietat / Mètode	Valor retornat	Descripció
<code>parentNode()</code>	<code>Node</code>	Referència del node pare o <code>null</code> si és el node arrel.
<code>hasChildNodes()</code>	<code>Boolean</code>	Retorna <code>true</code> si el node actual té un o més nodes fill.
<code>attributes</code>	<code>NamedNodeMap</code>	S'empra amb nodes de tipus <code>Element</code> . Conté Objectes de tipus <code>Attr</code> que defineixen tots els atributs de l'element.
<code>appendChild(nodo)</code>	<code>Node</code>	Afegeix un nou node al final de de la llista <code>childNodes</code> .
<code>removeChild(nodo)</code>	<code>Node</code>	Elimina un node de la llista <code>childNodes</code> .
<code>replaceChild(nouNode, anteriorNode)</code>	<code>Node</code>	Reemplaça el node <code>anteriorNode</code> pel node <code>nouNode</code> .
<code>insertBefore(nouNode, anteriorNode)</code>	<code>Node</code>	Insereix el node <code>nouNode</code> abans que la posició del node <code>anteriorNode</code> dins de la llista <code>childNodes</code> .
<code>children</code>	<code>HTMLCollection</code>	Funciona igual que <code>childNodes</code> , però només obté els nodes fills que siguin elements (no text)

# JS Avançat: DOM. La interfície node.

---

Els mètodes i propietats incloses en la taula anterior són específics de XML, encara que poden aplicar-se a tots els llenguatges basats en XML, com per exemple XHTML. Per a les pàgines creades amb HTML, els navegadors fan com si HTML estigués basat en XML i ho tracten de la mateixa forma. No obstant això, s'han definit algunes extensions i particularitats específiques per XHTML i HTML.

# JS Avançat: DOM. HTML i DOM.

---

Desafortunadament, l'ús de DOM sempre està limitat per les possibilitats que ofereix cada navegador (depenent del navegador s'implementa el nivell 1, 2 o 3).

Quan s'utilitza DOM en pàgines HTML, el node arrel de tots els altres es defineix en l'objecte `HTMLDocument`. A més, es creen objectes de tipus `HTMLElement` per cada node de tipus `Element` de l'arbre DOM. Com es veurà en el següent capítol, l'objecte `document` és part del BOM (*Browser Object Model*), encara que també es considera que és equivalent de l'objecte `Document` del DOM dels documents XML. En qualsevol cas, l'objecte `document` també fa referència al node arrel de totes les pàgines HTML.



# JS Avançat: DOM. HTML i DOM

---

## ● Accés relatiu als nodes:

```
<html>
<head>
  <title>Aprenent DOM</title>
</head>
<body>
  <p>Aprenent DOM</p>
  <p>DOM és fàcil d'aprendre</p>
  <p>A més a més, DOM és molt potent</p>
</body>
</html>
```

### Obtenir arrel:

```
let objeto_html = document.documentElement;
```

En la variable `objeto_html` conté un objecte de tipus `HTMLElement` que representa l'element `<html>` de la pàgina. Segons l'arbre de node DOM de `<html>` derivaran: `<head>` i `<body>`.

**Obtenir head** (primer node fill de l'element `<html>`):

```
let objeto_head = objeto_html.firstChild;
```

```
let objeto_head = objeto_html.childNodes[0];
```

**Obtenir body** (darrer node fill de l'element `<html>`):

```
let objeto_body = objeto_html.lastChild;
```

```
let objeto_body = objeto_html.childNodes[1];
```

# JS Avançat: DOM. HTML i DOM

---

## ● Accés relatiu als nodes:

```
<html>
<head>
  <title>Aprenent DOM</title>
</head>
<body>
  <p>Aprenent DOM</p>
  <p>DOM és fàcil d'aprendre</p>
  <p>A més a més, DOM és molt potent</p>
</body>
</html>
```

Si vols conèixer el **nombre de fills d'un node**:

```
objeto_html.childNodes.length;
```

Per **accedir directament al body**:

```
let objeto_body = document.body;
```

# JS Avançat: DOM. HTML i DOM

---

## ● Accés relatiu als nodes:

```
<html>
<head>
  <title>Aprenent DOM</title>
</head>
<body>
  <p>Aprenent DOM</p>
  <p>DOM és fàcil d'aprendre</p>
  <p>A més a més, DOM és molt potent</p>
</body>
</html>
```

A més de les propietats anteriors, existeixen altres propietats com **previousSibling** i **parentNode** que es poden utilitzar per accedir a un node a partir d'un altre. Utilitzant aquestes propietats, es poden comprovar les següents igualtats:

```
objeto_head.parentNode == objeto_html
objeto_body.parentNode == objeto_html
objeto_body.previousSibling ==
objeto_head
objeto_head.nextSibling == objeto_body
objeto_head.ownerDocument == document
```

# JS Avançat: DOM. HTML i DOM.

---

- **Tipus de nodes:**

- El tipus de node s'obté de la propietat **nodeType**:

- `alert(document.nodeType); // 9 - definit en constant`

- `Node.DOCUMENT_NODE`

- `alert(document.documentElement.nodeType); //1-definit en const.`

- `Node.ELEMENT_NODE`

- Això en un bucle ens podria servir per determinar amb quin tipus de nodes estem fent feina:

- `if (document.documentElement.nodeType == Node.ELEMENT_NODE) {`

# JS Avançat: DOM. HTML i DOM. Atributs.

---

A més de a l'etiqueta HTML i el seu contingut de text, el DOM permet l'accés als atributs de cada etiqueta HTML. Els nodes de tipus `Element` contenen la propietat `attributes`, que permet accedir a tots els atributs de cada element com si vos un array (en realitat la propietat `attributes` és de tipus `NamedNodeMap`).

# JS Avançat: DOM. HTML i DOM. Atributs.

---

## Mètodes per fer feina amb atributs:

**getNamedItem(nom)**, retorna el node del que la propietat **nodeName** contengui el valor “nom”.

**removeNamedItem(nom)**, elimina el node que tengui la propietat **nodeName** que coincideixi amb el valor “nom”.

**setNamedItem(node)**, afegeix el node a la llista **attributes**, indexant-ho segons la seva propietat **nodeName**.

**item(posicio)**, retorna el node que es troba en la posició indicada pel valor numèric “posicio”.

NOTA: Tots aquests mètodes no retornen directament el valor de l'atribut, sinó que retornen un node de tipus **Attr**.

# JS Avançat: DOM. HTML i DOM. Atributs.

---

## Exemples d'ús:

```
<p id="introduccion" style="color: blue">Párrafo de prueba</p>
```

```
let p = document.getElementById("introduccion");  
let elId = p.attributes.getNamedItem("id").nodeValue; // elId = "introduccion"  
let elId = p.attributes.item(0).nodeValue;           // elId = "introduccion"  
p.attributes.getNamedItem("id").nodeValue = "preintroduccion";  
  
let atributo = document.createAttribute("lang");  
atributo.nodeValue = "es";  
p.attributes.setNamedItem(atributo);
```

# JS Avançat: DOM. HTML i DOM. Atributs.

---

Afortunadament DOM aporta altres mètodes que permeten l'accés i la modificació dels atributs de forma més directa:

**getAttribute(nombre)**, és equivalent a  
`attributes.getNamedItem(nom).`

**setAttribute(nom, valor)** equivalent a  
`attributes.getNamedItem(nom).value = valor.`

**removeAttribute(nom)**, equivalent a  
`attributes.removeNamedItem(nom).`



# JS Avançat: DOM. HTML i DOM. Atributs.

---

Exemples d'ús:

```
<p id="introduccion" style="color: blue">Párrafo de prueba</p>
```

```
let p = document.getElementById("introduccion");  
let elId = p.getAttribute("id"); // elId = "introduccion"  
p.setAttribute("id", "preintroduccion");
```

# JS Avançat: DOM. HTML i DOM. Accés directe als nodes.

---

Hem vist com accedint al node arrel de la pàgina podem anar accedint a la resta de nodes (del pare als seus fills i així successivament). Si tenim molt de nodes fins arribar al node que volem pot ser molt ineficient.

Per això, tenim mètodes per **accedir de forma directa als nodes** desitjats:

`getElementsByTagName()`

`getElementsByName()`

`getElementById()`

# JS Avançat: DOM. HTML i DOM. Accés directe als nodes.

---

**getElementsByTagName()**: Obté tots els elements de la pàgina que tinguin la mateixa etiqueta que el paràmatre que se li passa a la funció.

```
let paragraphs = document.getElementsByTagName("p");
```

**Primer paràgraf:** paragraphs[0]

**Darrer paràgraf:** paragraphs[paragraphs.length - 1]

**Recòrrer elements:**

```
let paragraphs = document.getElementsByTagName("p");
for (let par of paragraphs) {
  let paragraf = par;
}
```

Es pot **aplicar de forma recursiva** als nodes retornats per la funció:

```
let paragraphs = document.getElementsByTagName("p");
let primerParagraf = paragraphs[0];
let enllaços = primerParagraf.getElementsByTagName("a");
```

# JS Avançat: DOM. HTML i DOM. Accés directe als nodes.

El mètode **getElementsByTagName()** el tenen tots els elements. Si en lloc d'utilitzar **document**, usam un element concret, llavors se cerquen les etiquetes dins aquest element.  
Exemple:

```
▼ <main id="principal">  
  <p>Un</p>  
  <p>Dos</p>  
  <p>Tres</p>  
</main>  
  <p>Quatre</p>  
  <p>Cinc</p>
```

```
> principal = document.getElementById('principal') // Obtenim el node de l'element main  
◀ ▶<main id="principal">...</main>  
  
> principal.getElementsByTagName('p')  
◀ ▶HTMLCollection(3) [p, p, p]  
  
> document.getElementsByTagName('p')  
◀ ▶HTMLCollection(5) [p, p, p, p, p]
```

# JS Avançat: DOM. HTML i DOM. Accés directe als nodes.

---

**getElementsByTagName()**: Obté tots els elements de la pàgina que tenguin l'atribut **name** igual que el paràmetre que se li passa a la funció.

L'atribut **name** sol ser únic pels elements HTML. En el cas de **radiobutton** és comú a tots els **radiobutton** que estan relacionats (aquí retornari una col·lecció d'elements).

```
<p name="prova">...</p>
<p name="especial">...</p>
<p>...</p>
```

```
let paragrafEspecial = document.getElementsByTagName("especial");
```

# JS Avançat: DOM. HTML i DOM. Accés directe als nodes.

---

**getElementById()**: Obté tots els elements de la pàgina que tenguin l'atribut **id** igual que el paràmatre que se li passa a la funció.

És la funció que més es fa servir pels desenvolupadors d'aplicacions web dinàmiques.

L'atribut **id** ha de ser únic per a cada element d'una mateixa pàgina.

```
<div id="cap">  
  <a href="/" id="logo">...</a>  
</div>  
let capçalera = document.getElementById("cap");
```

# JS Avançat: DOM. HTML i DOM. Crear, modificar i eliminar nodes.

---

Mètodes per creació de nous nodes	Descripció
<code>createAttribute(nom)</code>	Crear un altre node de tipus atribut amb el nom indicat
<code>createCDATASection(text)</code>	Crea una secció <b>CDATA</b> amb un node fill de tipus text que conté el valor indicat
<code>createComment(text)</code>	Crear un node de tipus comentari que conté el valor indicat.
<code>createDocumentFragment()</code>	Crear un node de tipus <b>DocumentFragment</b>
<code>createElement(nom_etiqueta)</code>	Crea un nou element del tipus indicat en el paràmetre <b>nom_etiqueta</b>
<code>createEntityReference(nom)</code>	Crea un node de tipus <b>EntityReference</b>
<code>createProcessingInstruction(Objectiu, Dades)</code>	Crear un node de tipus <b>ProcessingInstruction</b>
<code>createTextNode(text)</code>	Crear un node de tipus text amb el valor Indicad com a paràmetre

# JS Avançat: DOM. HTML i DOM. Crear, modificar i eliminar nodes.

---

## Creació d'elements HTML simples:

Un element HTML senzill, com un paràgraf, genera dos nodes:

Un primer node de tipus **Element** que representa l'etiqueta <p>.

Un segon node de tipus **Text** amb el contingut textual del paràgraf.

Per aquest motiu, crear i afegir un nou element consta de 4 passes:

1. Creació d'un node de tipus Element (representa l'element).
2. Creació d'un node de tipus Text (representa el contingut de l'element).
3. Afegir el node Text com un node fill del node Element.
4. Afegir el node Element a la pàgina, com a node fill del node corresponent al lloc on es vulgui insertar l'element.



# JS Avançat: DOM. HTML i DOM. Crear, modificar i eliminar nodes.

---

## Creació d'elements HTML simples:

Exemple:

```
// Crear node de tipus Element
let paragraf = document.createElement("p");
// Crear node de tipus Text
let contingut = document.createTextNode("Hello World!");
// Afegir el node Text com a fill del node Element
paragraf.appendChild(contingut);
// Afegir el node Element com a fill de la pàgina
document.body.appendChild(paragraf);
```

# JS Avançat: DOM. HTML i DOM. Crear, modificar i eliminar nodes.

---

## Creació d'elements HTML simples:

Si vos fixau, el procés de creació d'un element implica la utilització de tres funcions:

**createElement(etiqueta):** crea un node de tipus Element que representa a l'element HTML l'etiqueta del qual es passa com a paràmetre.

**createTextNode(contingut):** crea un node de tipus Text que emmagatzema el contingut textual dels elements HTML.

**nodoPare.appendChild(nodeFill):** afegeix un node com a fill d'un altre node. S'ha d'utilitzar almenys dues vegades amb els nodes habituals: en primer lloc s'afegeix el node Text com a fill del node Element i a continuació s'afegeix el node Element com a fill d'algun node de la pàgina.

# JS Avançat: DOM. HTML i DOM. Crear, modificar i eliminar nodes.

---

## Eliminació de nodes:

Eliminar un node és més fàcil i es fa amb la funció **removeChild()**:

```
let paragraf = document.getElementById("provisional");  
//let p = document.getElementsByTagName("p")[0];  
paragraf.parentNode.removeChild(paragraf);
```

`removeChild()` requereix com a **paràmetre el node a eliminar**. A més, aquesta funció ha de ser **invocada des de l'element pare** d'aquest node que es vol eliminar (mitjançant la propietat **nodeFill.parentNode**).

Quan s'elimina un node **s'eliminen automàticament tots els nodes fills** que tenguí.

# JS Avançat: DOM. HTML i DOM. Crear, modificar i eliminar nodes.

---

## Modificar nodes:

Modificar un node es fa amb **replaceChild()**:

```
let nouP = document.createElement("p");
let txt = document.createTextNode("Aquest paragraf substitueix al
    paragraf original");
nouP.appendChild(txt);
let anteriorP = document.body.getElementsByTagName("p")[0];
//let paragraf = document.getElementById("panterior");
anteriorP.parentNode.replaceChild(nouP, anteriorP)
```

`replaceChild()` requereix com a **paràmetres el node nou i el node a reemplaçar**. A més, aquesta funció ha de ser **invocada des de l'element pare** del node que es vol canviar (mitjançant la propietat **nodeFill.parentNode**).


# JS Avançat: DOM. HTML i DOM. Crear, modificar i eliminar nodes.

---

## Inserir nodes després d'un node:

```
<html>
<head><title>Exemple</title></
head>
  <body>
    <p>Primer paragraf</p>
  </body>
</html>
```

```
<html>
<head><title>Exemple</title></
head>
  <body>
    <p>Primer paragraf</p>
    <p>Segon paragraf</p>
  </body>
</html>
```



```
let nouP = document.createElement("p");
let txt = document.createTextNode("Segon paragraf");
nouP.appendChild(txt);
document.body.appendChild(nouP);
```


# JS Avançat: DOM. HTML i DOM. Crear, modificar i eliminar nodes.

---

## Inserir nodes abans d'un node:

```
<html>
<head><title>Exemple</title></
head>
  <body>
    <p>Primer paragraf</p>
  </body>
</html>
```

```
<html>
<head><title>Exemple</title></
head>
  <body>
    <p>Segon paragraf</p>
    <p>Primer paragraf</p>
  </body>
</html>
```



```
var nouP = document.createElement("p");
var txt = document.createTextNode("Segon paragraf");
nouP.appendChild(txt);
var anteriorP = document.getElementsByTagName("p")[0];
document.body.insertBefore(nouP, anteriorP);
```

## JS Avançat: DOM. HTML i DOM. Crear, modificar i eliminar nodes.

---

**MOLT IMPORTANT:** Per fer aquestes operacions amb el DOM tota la pàgina s'ha d'haver carregat. Per aconseguir això, és important utilitzar l'esdeveniment **onload()** per cridar a les funcions de Javascript, tal com es veurà més endavant en el capítol dels esdeveniments.

# JS Avançat: DOM. HTML i DOM. Atributs HTML i propietats CSS en DOM.

Existeixen diferències entre DOM per XML i HTML. El principal avantatge del **DOM per HTML** es que tots els atributs de tots els elements HTML es transformen en propietats dels nodes. D'aquesta forma es pot accedir de forma directa a qualsevol atribut d'HTML.

Considerem fragment HTML:

```

let imatge = document.getElementById("logo");
```

Per accedir als atributs:

```
let arxiu = imatge.getAttribute("src");
let vorera = imatge.getAttribute("border");
```

Per modificar els atributs:

```
imatge.setAttribute("src", "nuevo_logo.gif");
imatge.setAttribute("border", "1");
```



# JS Avançat: DOM. HTML i DOM. Atributs

## HTML i propietats CSS en DOM.

El que hem vist abans ha estat fent servir els mètodes “tradicionals” de DOM. L'avantatge de l'especificació DOM per HTML es que permet accedir i modificar tots els atributs dels elements de forma directa:

```
let imatge = document.getElementById("logo");  
// accedir als atributs  
let arxiu = imatge.src;  
let vorera = imatge.border;  
// modificar els atributs  
imatge.src = "nuevo_logo.gif";  
imatge.border = "1";
```

# JS Avançat: DOM. HTML i DOM. Atributs HTML i propietats CSS en DOM.

---

Una excepció al que hem dit en la transparència anterior és amb l'atribut `class`. Com que "class" és una paraula reservada per JS, s'accedeix a l'atribut amb el nom alternatiu `className`:

```
<p id="paragraf" class="normal">...</p>
let paragraf = document.getElementById("paragraf");
alert(paragraf.class);      // mostra "undefined"
alert(paragraf.className); // mostra "normal"
```

L'accés a les propietats CSS establertes mitjançant l'atribut `style` es realitza a través de la propietat `style` del node que representa l'element:

```
<p id="paragraf" style="color: #C00">...</p>
let paragraf = document.getElementById("paragraf");
let color = paragraf.style.color;
```

# JS Avançat: DOM. HTML i DOM. Atributs HTML i propietats CSS en DOM.

---

Un altre exemple d'accés a les propietats CSS fent servir l'atribut **style**:

```
let imatge = document.getElementById("imatge");  
alert(imatge.style.margin);  

```

NOTA: Pot passar que amb navegadors antics variï el valor que es retorna.

De vegades també canvia un poc el nom de la propietat:

```
let paragraf = document.getElementById("paragraf");  
alert(paragraf.style.fontWeight); // mostra "bold"  
<p id="paragraf" style="font-weight: bold;">...</p>
```

# JS Avançat: DOM. HTML i DOM. Atributs HTML i propietats CSS en DOM.

---

Aquestes modificacions en els noms de les propietats CSS es fan seguint camelCase:

font-weight -> fontWeight

line-height -> lineHeight

border-top-style -> borderTopStyle

list-style-image -> listStyleImage)

Recordeu que `class` també canvia per `className`. Altre ex.:

```
let paragraf = document.getElementById("paragraf");  
alert(paragraf.class); // mostra "undefined"  
alert(paragraf.className); // mostra "normal"  
<p id="paragraf" class="normal">...</p>
```

# JS Avançat: DOM. Manipulació del contingut dels elements

---

- La propietat **textContent** d'un element permet obtenir i modificar el text que conté aquest element. S'ha de tenir present que un element pot tenir dins més elements i aquest propietat obtindrà el text de TOTS ells. Exemple:

```
▼<p id="paragraf">  
  "La meva "  
  <strong>vida</strong>  
  " es lliga a tu com en la nit les "  
  <span style="color: red">flames</span>  
  " a la fosca"  
</p>
```

La meva **vida** es lliga a tu com en la nit les **flames** a la fosca

```
document.getElementById('paragraf').textContent  
"La meva vida es lliga a tu com en la nit les flames a la fosca"
```

# JS Avançat: DOM. Manipulació del contingut dels elements

---

Amb **textContent** també podem canviar el contingut, però només el text, no els elements interiors. Exemple:

```
document.getElementById('paragraf').textContent = "Te deix  
<strong>amor</strong> la mar com a penyora"
```

En el navegador veurem:

Te deix <strong>amor</strong> la mar com a penyora

# JS Avançat: DOM. Manipulació del contingut dels elements

---

- La propietat `innerHTML` és similar a l'anterior, però aquesta sí llegeix i manipula les etiquetes HTML. Exemple:

```
document.getElementById('paragraf').innerHTML  
"La meva <strong>vida</strong> es lliga a tu com en la nit les <span style  
="color: red">flames</span> a la fosca"
```

```
document.getElementById('paragraf').innerHTML = "Te deix  
<strong>amor</strong> la mar com a penyora"
```

En el navegador veurem:

Te deix **amor** la mar com a penyora

# Bibliografia

[http://librosweb.es/libro/ajax/capitulo\\_4.html](http://librosweb.es/libro/ajax/capitulo_4.html)

[http://librosweb.es/libro/javascript/capitulo\\_5.html](http://librosweb.es/libro/javascript/capitulo_5.html)

<http://www.maestrosdelweb.com/dom/>

[http://www.w3schools.com/js/js\\_htmlDOM.asp](http://www.w3schools.com/js/js_htmlDOM.asp)



# Mirar també a ...

- [https://www.w3schools.com/js/js\\_html\\_dom\\_elements.asp](https://www.w3schools.com/js/js_html_dom_elements.asp)
  - Cercar elements HTML per un nom de classe. Ex:
    - `var x = document.getElementsByClassName("intro");`
  - Cercar elements HTML amb selectors CSS. Ex:
    - `var x = document.querySelectorAll("p.intro");`
  - Cercar elements HTML amb les col·leccions d'objectes de HTML. Ex:
    - `var x = document.forms["frm1"];`

# Col·leccions d'objectes

- document.anchors
- document.body
- document.documentElement
- document.embeds
- document.forms
- document.head
- document.images
- document.links
- document.scripts
- document.title