UD 4.2 INTEGRACIÓ DE JAVASCRIPT EN EL NAVEGADOR

Índex1. ENTORN DEL NAVEGADOR, ESPECIFICACIONS 3

1.1. DOM (Model de Objectes del Document) 4

1.2. BOM (Model d'Objectes del Navegador)....4

2. RECORRENT EL DOM....6

2.1. En la part superior: documentElement i body 6

<HTML> = document.documentElement 6
<body> = document.body 6
<head> = document.head 6

2.2. Fills: childNodes, firstChild, lastChild 7

2.3. Germans i el pare 8

2.4. Navegació només per elements....9

3. BUSCAR: GETELEMENT*, QUERYSELECTOR* 11

3.1. document.getElementById o només id 11

3.2. querySelectorAll....12

3.3. querySelector....13

3.4. closest....13

3.5. getElementsBy*....14

4. PROPIETATS DEL NODE: TIPUS, ETIQUETA I CONTINGUT 16

4.1. Classes de node DOM 16

4.2. innerHTML: els continguts 16

4.3. outerHTML: HTML complet de l'element 17

4.4. textContent: text pur 17

4.5. La propietat “hidden” 18

5. MODIFICANT EL DOCUMENT 19

5.1. Exemple: mostrar un missatge 19

5.2. Creant el missatge 20

5.3. Mètodes d'inserció 20

5.4. Eliminació de nodes....21

6. ESTILS I CLASSES 22

6.1. className i classList 22

6.2. style d'un element....24

6.3. Reescriure tot usant style.cssText 25
6.4. Compte amb les unitats CSS 25
7. GRANDÀRIA D'ELEMENTS I DESPLAÇAMENT 26
7.1. Element de mostra 26
7.2. Geometria 27
7.3. offsetParent, offsetLeft/Top 27
7.4. offsetWidth/Height 29
7.5. clientTop/Left 30
7.6. clientWidth/Height 31
7.7. scrollWidth/Height 33
7.8. scrollLeft/scrollTop 34
8. GRANDÀRIA DE FINESTRA I DESPLAÇAMENT 35
8.1. Ample/alt de la finestra 35
8.2. Ample/Alt del document 35
8.3. Obtindre el desplaçament actual 36
8.4. Desplaçament: scrollTo, scrollBy, scrollIntoView 37

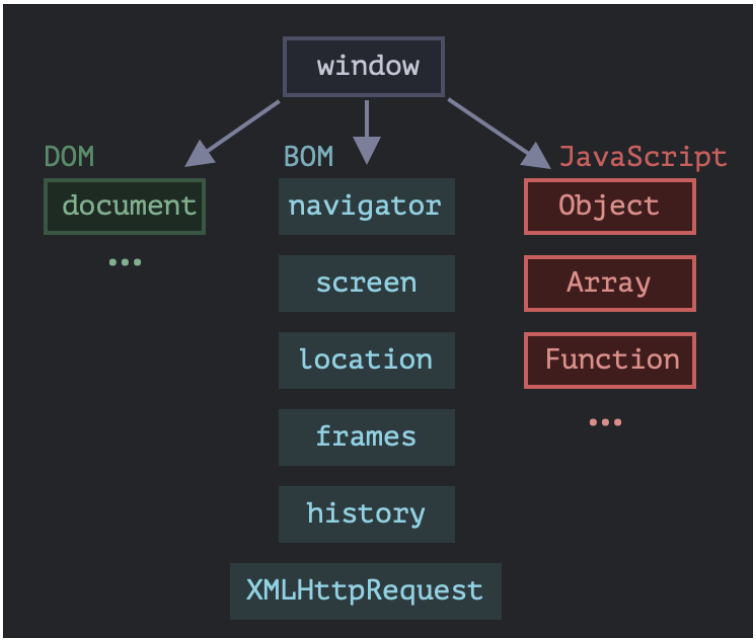
1. ENTORN DEL NAVEGADOR, ESPECIFICACIONS

El llenguatge JavaScript va ser creat inicialment per als navegadors web. Des de llavors, ha evolucionat en un llenguatge amb molts usos i plataformes.

Una plataforma pot ser un navegador, un servidor web o un altre host ("amfitrió"); fins i tot una màquina de café "intel·ligent", si pot executar JavaScript. Cadascun d'ells proporciona una funcionalitat específica de la plataforma. L'especificació de JavaScript crida a això entorn d'host.

Un entorn host proporciona els seus propisobjectes i funcions addicionals al nucli del llenguatge. Els navegadors web proporcionen un mitjà per a controlar les pàgines web. Node.js proporciona característiques del costat del servidor, etc.

Ací tens una vista general del que tenim quan JavaScript s'executa en un navegador web:



Hi ha un objecte "arrel" anomenat window. Té dos rols:

- Primer, és un objecte global per al codi JavaScript. Tot s'executa dins d'este objecte.
- Segon, representa la "finestra del navegador" i proporciona mètodes per a controlar-la.

Per exemple, podem usar-ho com a objecte global:

```
function sayHi() {
 alert("Hola");
}

// Les funcions globals són mètodes de l'objecte global:
window.sayHi();
```

I podem usar-ho com una finestra del navegador. Per a veure l'altura de la finestra:

```
alert(window.innerHeight); // altura interior de la finestra
```

1.1. DOM (Model de Objectes del Document)

Document Object Model, o DOM, representa tot el contingut de la pàgina com a objectes que poden ser modificats.

L'objecte document és el punt d'entrada a la pàgina. Amb ell podem canviar o crear qualsevol cosa en la pàgina.

Per exemple:

```
// canviar el color de fons a roig
document.body.style.background = "xarxa";
// desfer el canvi després d'1 segon
setTimeout(() => document.body.style.background = "", 1000);
```

Ací usem document.body.style, però hi ha molts, molts més. Les propietats i mètodes es descriuen en l'especificació: DOM Living Standard.

1.2. BOM (Model d'Objectes del Navegador)

El Model d'Objectes del Navegador (Browser Object Model, BOM) són objectes addicionals proporcionats pel navegador (entorn host) per a treballar amb tot excepte el document.

Per exemple:

- L'objecte navigator proporciona informació sobre el navegador i el sistema operatiu. Hi ha moltes propietats, però les dues més conegudes són: navigator.userAgent: sobre el navegador actual, i navigator.platform: sobre la plataforma (ajuda a distingir Windows/Linux/Mac, etc.).

- L'objecte location ens permet llegir la URL actual i pot redirigir el navegador a una nova.

Ací veiem com podem usar l'objecte location:

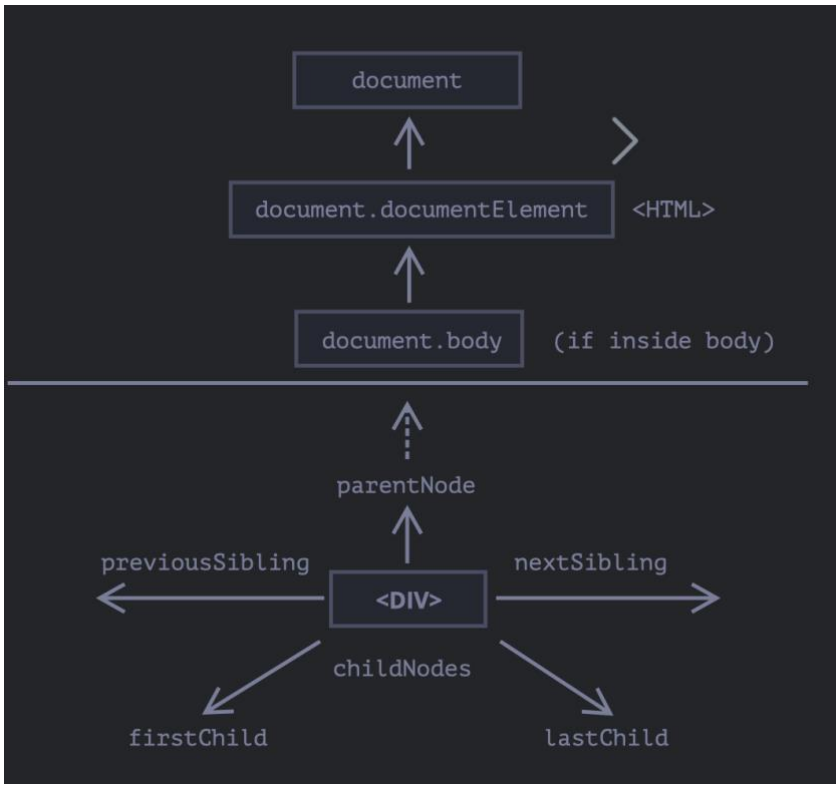
```
alert(location.href); // mostra la URL actual
if (confirm("Anar a wikipedia?")) {
 location.href = "https://wikipedia.org"; // redirigir el navegador a una altra URL
}
```

2. RECORRENT EL DOM

El DOM ens permet fer qualsevol cosa amb els seus elements i continguts, però el primer que hem de fer és arribar a l'objecte corresponent del DOM.

Totes les operacions en el DOM comencen amb l'objecte document. Este és el principal "punt d'entrada" al DOM. Des d'ací podrem accedir a qualsevol node.

Esta imatge representa els enllaços que ens permeten viatjar a través dels nodes del DOM:



Els analitzarem amb més detall.

2.1. En la part superior: documentElement i body

Els tres nodes superiors estan disponibles com a propietats de document:

<HTML> = document.documentElement

El node superior del document és document.documentElement. Este és el node del DOM per a l'etiqueta <HTML>.

<body> = document.body

Un altre node molt utilitzat és l'element <body> – document.body.

<head> = document.head

L'etiqueta <head> està disponible com document.head.

2.2.Fills: childNodes, firstChild, lastChild

Existixen dos termes que utilitzarem d'ara en avant:

- Nodes fills (childNodes) – elements que són fills directes, és a dir els seus descendents immediats. Per exemple, <head> i <body> són fills de l'element <HTML>.
- Descendents – tots els elements niats d'un element donat, incloent-hi els fills, els seus fills i així successivament.

La col·lecció childNodes enumera tots els nodes fills, inclosos els nodes de text.

L'exemple inferior mostra tots els fills de document.body:

```
<HTML>
<body>
<div>Begin</div>

<ul>
<li>Information</li>
</ul>

<div>End</div>

<script>
for (let i = 0; i < document.body.childNodes.length; i++) {
alert( document.body.childNodes[i] ); // Text, DIV, Text, UL, ..., SCRIPT
}
</script>
...més coses...
</body>
</HTML>
```

Per favor observa un interessant detall ací. Si executem l'exemple anterior, l'últim element que es mostra és <script>. De fet, el document té més coses davall, però en el moment d'execució del script el navegador encara no l'ha llegit, per la qual cosa el script no el veu.

Les propietats firstChild i lastChild donen accés ràpid al primer i a l'últim fill.

Són sol dreceres. Si existiren nodes fills, la resposta següent seria sempre vertadera:

```
elem.childNodes[0] === elem.firstChild
elem.childNodes[elem.childNodes.length - 1] === elem.lastChild
```

També hi ha una funció especial elem.hasChildNodes() per a comprovar si hi ha alguns nodes fills.

2.3.Germans i el pare

Els germans són nodes que són fills del mateix pare.

Per exemple, ací <head> i <body> són germans:

```
<HTML>
 <head>...</head><body>...</body>
</HTML>
```

- <body> es diu que és el germà “següent” o a la “dreta” d'head <>,
- <head> es diu que és el germà “anterior” o a la “esquerra” de <body>.

El germà següent està en la propietat nextSibling i l'anterior – en previousSibling.

El pare està disponible en parentNode.

Per exemple:

```
 // el pare de <body> és <HTML>
alert( document.body.parentNode === document.documentElement ); // vertader

// després d'head <> va <body>
alert( document.head.nextSibling ); // HTMLBodyElement

// abans de <body> va <head>
alert( document.body.previousSibling ); // HTMLHeadElement
```

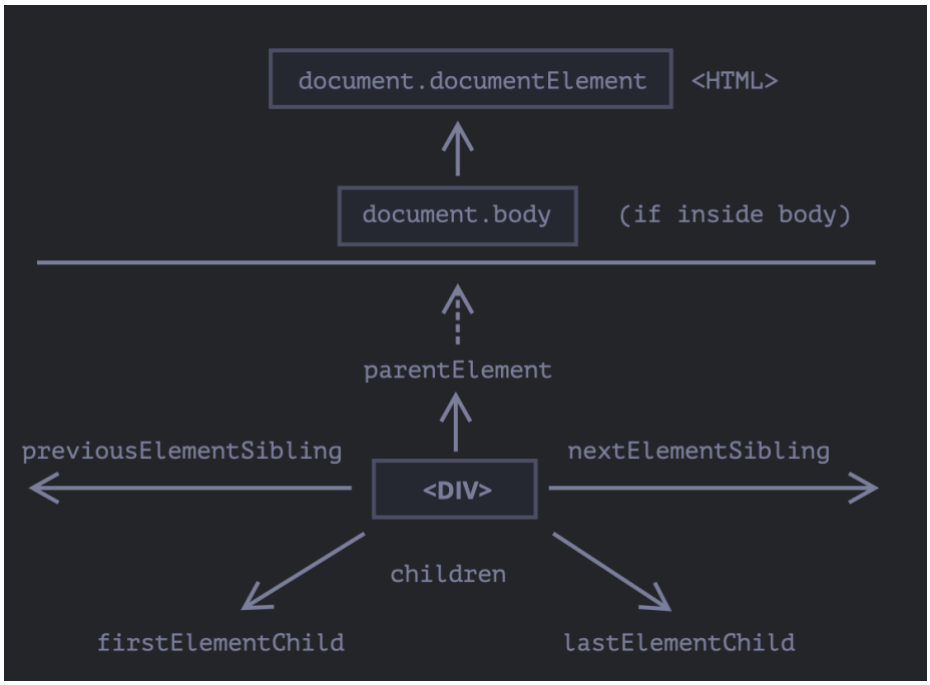
2.4. Navegació només per elements

Les propietats de navegació enumerades a baix es referixen a tots els nodes. Per exemple, en childNodes podem veure nodes de text, nodes elements; i si existixen, fins i tot els nodes de comentaris.

Però per a moltes tasques no volem els nodes de text o comentaris. Volem manipular el node que representa les etiquetes i formularis de l'estructura de la pàgina.

Així que veurem més enllaços de navegació que només tenen en compte els elements nodes:

Els enllaços són similars als de dalt, només que tenen dins la paraula Element:



- children – només eixos fills que tenen l'element node.
- firstElementChild, lastElementChild – el primer i l'últim element fill.
- previousElementSibling, nextElementSibling – elements veïns.
- parentElement – element pare.

Modificarem un dels exemples de dalt: reemplaça childNodes per children. Ara ensenya sol elements:

```
<HTML>
<body>
<div>Begin</div>
<ul>
<li>Information</li>
</ul>
<div>End</div>
<script>
for (let elem of document.body.children) {
alert(elem); // DIV, UL, DIV, SCRIPT
}
</script>
</body>
</HTML>
```

3. BUSCAR: GETELEMENT*, QUERYSELECTOR*

Les propietats de navegació del DOM són ideals quan els elements són a prop els uns dels altres. Però, i si no ho estan? Com obtindre un element arbitrari de la pàgina?

Per a estos casos existixen mètodes de cerca addicionals.

3.1. document.getElementById o només id

Si un element té l'atribut id, podem obtindre l'element usant el mètode document.getElementById(id), sense importar on es trobe.

Per exemple:

```
<div id="elem">
<div id="elem-content">Element</div>
</div>
<script>
// obtindre l'element
let elem = document.getElementById('elem');
// fer que el seu fons siga roig
elem.style.background = 'xarxa';
</script>
```

Existix a més una variable global nomenada per l'id que fa referència a l'element:

```
<div id="elem">
<div id="elem-content">Element</div>
</div>
<script>
// elem és una referència a l'element del DOM amb id="elem"
elem.style.background = 'xarxa';
// id="elem-content" té un guió en el seu interior, per la qual cosa no pot ser un nom de variable
// ...però podem accedir a ell usant claudàtors: window['elem-content']
</script>
```

...Això és llevat que declarem una variable de JavaScript amb el mateix nom, llavors esta té prioritat:

```
<div id="elem"></div>

<script>
let elem = 5; // ara elem és 5, no una referència a <div id="elem">

alert(elem); // 5
</script>
```

3.2.querySelectorAll

Sens dubte el mètode més versàtil, elem.querySelectorAll(css) retorna tots els elements dins d'elem que coincidixen amb el selector CSS dau.

Ací busquem tots els elements <li> que són els últims fills:

```
<ul>
<li>La</li>
<li>prova</li>
</ul>
<ul>
<li>ha</li>
<li>passat/li>
</ul>
<script>
let elements = document.querySelectorAll('ul > li:last-child');

for (let elem of elements) {
alert(elem.innerHTML); // "prova", "passat"
}
</script>
```

Este mètode és molt poderós, perquè es pot utilitzar qualsevol selector de CSS.

3.3. querySelector

L'anomenada a elem.querySelector(css) retorna el primer element per al selector CSS dau.

En altres paraules, el resultat és el mateix que elem.querySelectorAll(css)[0], però este últim cerca tots els elements i tria un, mentre que elem.querySelector només busca un. Així que és més ràpid i també més curt d'escriure.

3.4. closest

Els ancestres d'un element són: el pare, el pare del pare, el seu pare i així successivament. Tots els ancestres junts formen la cadena de pares des de l'element fins al cim.

El mètode elem.closest(css) cerca l'ancestre més pròxim que coincidix amb el selector CSS. El propi elem també s'inclou en la cerca.

En altres paraules, el mètode closest puja de l'element i comprova cadascun dels pares. Si coincidix amb el selector, llavors la cerca es deté i retorna este ancestre.

Per exemple:

```
<h1>Contingut</h1>

<div class="contents">
 <ul class="book">
 <li class="chapter">Capítol 1</li>
 <li class="chapter">Capítol 2</li>
 </ul>
</div>

<script>
 let chapter = document.querySelector('.chapter'); // LI

 alert(chapter.closest('.book')); // UL
 alert(chapter.closest('.contents')); // DIV

 alert(chapter.closest('h1')); // null (perquè h1 no és un ancestre)
</script>
```

3.5.getElementsBy*

També hi ha altres mètodes que permeten buscar nodes per una etiqueta, una classe, etc.

Hui dia, són en la seua majoria historia, ja que querySelector és més poderós i curt d'escriure.

- elem.getElementsByTagName(tag) busca elements amb l'etiqueta donada i retorna una col·lecció amb ells. El paràmetre tag també pot ser un asterisc "*" per a “qualsevol etiqueta”.
- elem.getElementsByClassName(className) retorna elements amb la classe.
- document.getElementsByName(name) retorna elements amb l'atribut name dau, en tot el document. Molt rarament usat.

Per exemple:

```
// obtindre tots els divs del document
let divs = document.getElementsByTagName('div');
```

Per a trobar totes les etiquetes input dins d'una taula:

```
<table id="table">
<tr>
<td>La seua edat:</td>
<td>
<label>
<input type="radi" name="age" value="young" checked> menys de 18
</label>
<label>
<input type="radi" name="age" value="sènior"> més de 60
</label>
</td>
</tr>
</table>
<script>
let inputs = table.getElementsByTagName('input');
for (let input of inputs) {
alert( input.value + ': ' + input.checked );
}
</script>
```

4. PROPIETATS DEL NODE: TIPUS, ETIQUETA I CONTINGUT

4.1.Classes de node DOM

Els diferents nodes DOM poden tindre diferents propietats. Per exemple, un node d'element corresponent a l'etiqueta <a> té propietats relacionades amb l'enllaç, i el corresponent a <input> té propietats relacionades amb l'entrada i així successivament. Els nodes de text no són el mateix que els nodes d'elements. Però també hi ha propietats i mètodes comuns entre tots ells, perquè totes les classes de nodes DOM formen una única jerarquia.

4.2.innerHTML: els continguts

La propietat innerHTML permet obtindre l'HTML dins de l'element com un string.

També podem modificar-ho. Així que és una de les formes més poderoses de canviar la pàgina.

L'exemple mostra el contingut de document.body i després ho reemplaça per complet:

```
<body>
<p>Un paràgraf</p>
<div>Un div</div>

<script>
alert( document.body.innerHTML ); // llegir el contingut actual
document.body.innerHTML = 'El nou BODY!'; // reemplaçar
</script>

</body>
```

4.3.outerHTML: HTML complet de l'element

La propietat outerHTML conté l'HTML complet de l'element. Això és com innerHTML més l'element en si.

Heus ací un exemple:

```
<div id="elem">Hola <b>Món</b></div>
<script>
 alert(elem.outerHTML); // <div id="elem">Hola <b>Món</b></div>
</script>
```

4.4.textContent: text pur

El textContent proporciona accés al text dins de l'element: només text, menys totes les <tags>.

Per exemple:

```
<div id="news">
 <h1>Titular!</h1>
 <p>Els marcians ataquen a la gent!</p>
</div>

<script>
 // Titular! Els marcians ataquen a la gent!
 alert(news.textContent);
</script>
```

Com podem veure, només es retorna text, com si totes les <etiquetes> foren retallades, però el text en elles va romandre.

En la pràctica, rares vegades es necessita llegir este tipus de text.

Escriure en textContent és molt més útil, perquè permet escriure text de "manera segura".

Diguem que tenim un string arbitrari, per exemple, ingressat per un usuari, i volem mostrar-lo.

- Amb innerHTML ho tindrem inserit "com a HTML", amb totes les etiquetes HTML.
- Amb textContent ho tindrem inserit "com a text", tots els símbols es tracten literalment.

Compara els dos:

```
<div id="elem1"></div>
<div id="elem2"></div>

<script>
 let name = prompt("Quin és el teu nom?", "<b>Winnie-Pooh!</b>");

 elem1.innerHTML = name;
 elem2.textContent = name;
</script>
```

- El primer obté el nom “com a HTML”: totes les etiquetes es convertixen en etiquetes, per la qual cosa veiem el nom en negreta.
- El segon obté el nom “com a text”, així que literalment veiem Winnie-Pooh!.

En la majoria dels casos, esperem el text d'un usuari i volem tractar-lo com a text. No volem HTML inesperat en el nostre lloc. Una assignació a textContent fa exactament això.

4.5.La propietat “hidden”

L'atribut “hidden” i la propietat DOM especifiquen si l'element és visible o no.

Podem usar-ho en HTML o assignar-ho usant JavaScript, així:

```
<div>Tots dos divs a continuació estan ocults</div>
<div hidden>Amb l'atribut "hidden"</div>
<div id="elem">JavaScript va assignar la propietat "hidden"</div>
<script>
 elem.hidden = true;
</script>
```

Tècnicament, hidden funciona igual que style="display:none". Però és més curt d'escriure.

5. MODIFICANT EL DOCUMENT

La modificació del DOM és la clau per a crear pàgines “vives”, dinàmiques.

Ací veurem com crear nous elements “al vol” i modificar el contingut existent de la pàgina.

5.1. Exemple: mostrar un missatge

Fem una demostració usant un exemple. Afegirem un missatge que es veja més agradable que un alert.

Així és com es veurà:

```
<style>
.alert {
 padding: 15px;
 border: 1px solid #d6e9c6;
 border-radius: 4px;
 color: #3c763d;
 background-color: #dff0d8;
}
</style>

<div class="alert">
 <strong>Hola!</strong> Vosté ha llegit un important missatge.
</div>
```

Això va ser l'exemple HTML. Ara creiem el mateix div amb JavaScript (assumint que els estils ja estan en HTML/CSS).

5.2. Creant el missatge

Crear el div de missatge pren 3 passos:

```
// 1. Crear element <div>
let div = document.createElement('div');

// 2. Establir la seua classe a "alert"
div.className = "alert";

// 3. Agregar el contingut
div.innerHTML = "<strong>Hola!</strong> Vosté ha llegit un important missatge.";
```

Hem creat l'element. Però fins ara solament està en una variable anomenada div, no encara en la pàgina, i no la podem veure.

5.3. Mètodes d'inserció

Per a fer que el div aparega, necessitem inserir-ho en algun costat dins de document. Per exemple, en l'element <body>, referenciat per document.body.

Hi ha un mètode especial append per a això: document.body.append(div).

El codi complet:

```
<style>
.alert {
 padding: 15px;
 border: 1px solid #d6e9c6;
 border-radius: 4px;
 color: #3c763d;
 background-color: #dff0d8;
} </style>
<script>
 let div = document.createElement('div');
 div.className = "alert";
 div.innerHTML = "<strong>Hola!</strong> Vosté ha llegit un important.";
 document.body.append(div);
</script>
```

Ací usem el mètode append sobre document.body, però podem cridar append sobre qualsevol element per a posar un altre element dins d'ell. Per exemple, podem afegir alguna cosa a cridant div.append(anotherElement).

5.4. Eliminació de nodes

Per a llevar un node, tenim el mètode node.remove().

Fem que el nostre missatge desaparega després d'un segon:

```
<style>
.alert {
 padding: 15px;
 border: 1px solid #d6e9c6;
 border-radius: 4px;
 color: #3c763d;
 background-color: #dff0d8;
}
</style>

<script>
 let div = document.createElement('div');
 div.className = "alert";
 div.innerHTML = "<strong>Hola!</strong> Vosté ha llegit un important missatge.";

 document.body.append(div);
 setTimeout(() => div.remove(), 1000);
</script>
```

6. ESTILS I CLASSES

Abans d'aprofundir en com JavaScript maneja les classes i els estils, hi ha una regla important. Encara que és prou obvi, encara hem d'esmentar-ho.

En general, hi ha dues maneres de donar estil a un element:

1. Crear una classe css i agregar-la:
2. Escriure les propietats directament en style: .

JavaScript pot modificar tots dos, classes i les propietats de style.

Nosaltres hauríem de preferir les classes css en lloc de style. Este últim solo ha d'usar-se si les classes “no poden manejar-lo”.

Per exemple, style és acceptable si nosaltres calculem les coordenades d'un element dinàmicament i volem establir estes des de JavaScript, així:

```
let top = /* càlculs complexos */;
let left = /* càlculs complexos */;

elem.style.left = left; // ej. '123px', calculat en temps d'execució
elem.style.top = top; // ej. '456px'
```

Per a altres casos com convertir un text en roig, agregar una icona de fons. Escriure això en CSS i després agregar la classe (JavaScript pot fer això), és més flexible i més fàcil de mantindre.

6.1.className i classList

Canviar una classe és una de les accions més utilitzades.

En l'antiguitat, hi havia una limitació en JavaScript: una paraula reservada com "class" no podia ser una propietat d'un objecte. Eixa limitació no existix ara, però en eixe moment era impossible tindre una propietat "class", com elem.class.

Llavors per a classes de similars propietats, "className" va ser introduït: l'elem.className correspon a l'atribut "class".

Per exemple :

```
<body class="main page">
<script>
alert(document.body.className); // pàgina principal
</script>
</body>
```

Si assignem alguna cosa a elem.className, reemplaça tota la cadena de classes. A vegades és el que necessitem, però sovint volem agregar o eliminar una sola classe.

Hi ha una altra propietat per a això: elem.classList.

L'elem.classList és un objecte especial amb mètodes per a agregar, eliminar i alternar (add/remove/toggle) una sola classe.

Per exemple :

```
<body class="main page">
<script>
// agregar una classe
document.body.classList.add('article');

alert(document.body.className); // classe "article" de la pàgina principal
</script>
</body>
```

Llavors podem treballar amb tots dos: totes les classes com una cadena usant className o amb classes individuals usant classList. El que triem depén de les nostres necessitats.

Mètodes de classList:

- elem.classList.add/remove("class") – agrega o remou la classe.
- elem.classList.toggle("class") – agrega la classe si no existix, si no la remou.
- elem.classList.contains("class") – verifica si té la classe donada, retorna true/false.

A més, classList és iterable, llavors podem llistar totes les classes amb for..of, així:

```
<body class="main page">
<script>
for (let name of document.body.classList) {
alert(name); // main i després page
}
</script>
</body>
```

6.2.style d'un element

La propietat elem.style és un objecte que correspon a l'escrit en l'atribut "style". Establir elem.style.width="100px" funciona igual que sí que tinguérem en l'atribut style una cadena amb width:100px.

Per a propietats de diverses paraules s'usa camelCase:

```
background-color => elem.style.backgroundColor

z-index => elem.style.zIndex

border-left-width => elem.style.borderLeftWidth
```

Per exemple :

```
document.body.style.backgroundColor = prompt('background color?', 'green');
```

A vegades volem assignar una propietat d'estil i després remoure-la.

Per exemple, per a ocultar un element, podem establir elem.style.display = "none".

Després, més tard, és possible que vulguem remoure style.display com si no estiguera establit. En lloc de delete elem.style.display hauríem d'assignar-li una cadena buida: elem.style.display = "".

```
// si executem este codi, el <body> parpellejarà
document.body.style.display = "none"; // ocultar


setTimeout(() => document.body.style.display = "", 1000); // tornarà al normal
```

Si establim style.display com una cadena buida, llavors el navegador aplica classes i estils CSS incorporats normalment pel navegador, com si no existira tal style.display.

També hi ha un mètode especial per a això, elem.style.removeProperty('style property'). Així, podem llevar una propietat:

```
document.body.style.background = 'xarxa'; //establix background a roig


setTimeout(() => document.body.style.removeProperty('background'), 1000); //
llevar background després d'1 segon
```

6.3. Reescriure tot usant style.cssText

Normalment, podem usar style.* per a assignar propietats d'estil individuals. No podem establir tot l'estil com div.style="color: xarxa; width: 100px", perquè div.style és un objecte i és només de lectura.

Per a establir tot l'estil com una cadena, hi ha una propietat especial: style.cssText:

```
<div id="div">Button</div>
<script>
// podem establir estils especials amb banderes com "important"
div.style.cssText=`color: xarxa !important;
background-color: yellow;
width: 100px;
text-align: center;
`;
alert(div.style.cssText);
</script>
```

Esta propietat és rares vegades usada, perquè tal assignació remou tot els estils: no agrega estils sinó que els reemplaça íntegrament. Ocasionalment podria eliminar una cosa necessària. Però podem usar-ho de manera segura per a nous elements, quan sabem que no eliminarem un estil existent.

6.4. Compte amb les unitats CSS

No oblidar agregar les unitats CSS als valors.

Per exemple, nosaltres no hem d'establir elem.style.top a 10, sinó més aviat a 10px. En cas contrari no funcionaria:

```
<body>
<script>
document.body.style.margin = 20;
alert(document.body.style.margin); // '' (cadena buida, l'assignació és ignorada)
// ara agreguem la unitat CSS (px) i esta sí que funciona
document.body.style.margin = '20px';
alert(document.body.style.margin); // 20px
alert(document.body.style.marginTop); // 20px
alert(document.body.style.marginLeft); // 20px
</script>
</body>
```

7. GRANDÀRIA D'ELEMENTS I DESPLAÇAMENT

Hi ha moltes propietats en JavaScript que ens permeten llegir informació sobre l'ample, alt i altres característiques geomètriques dels elements.

Sovint necessitem d'elles quan movem o posicionem un element en JavaScript.

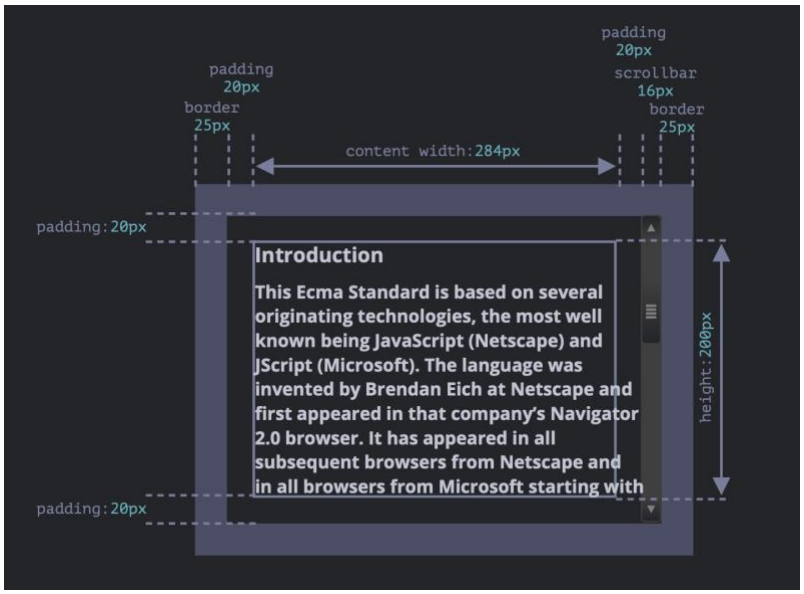
7.1. Element de mostra

Com un element de mostra per a demostrar les propietats, usarem el que s'indica a continuació:

```
<div id="example">
 </div>
<style>
 #example {
 width: 300px;
 height: 200px;
 border: 25px solid #E8C48F;
 padding: 20px;
 overflow: acte;
 }
</style>
```

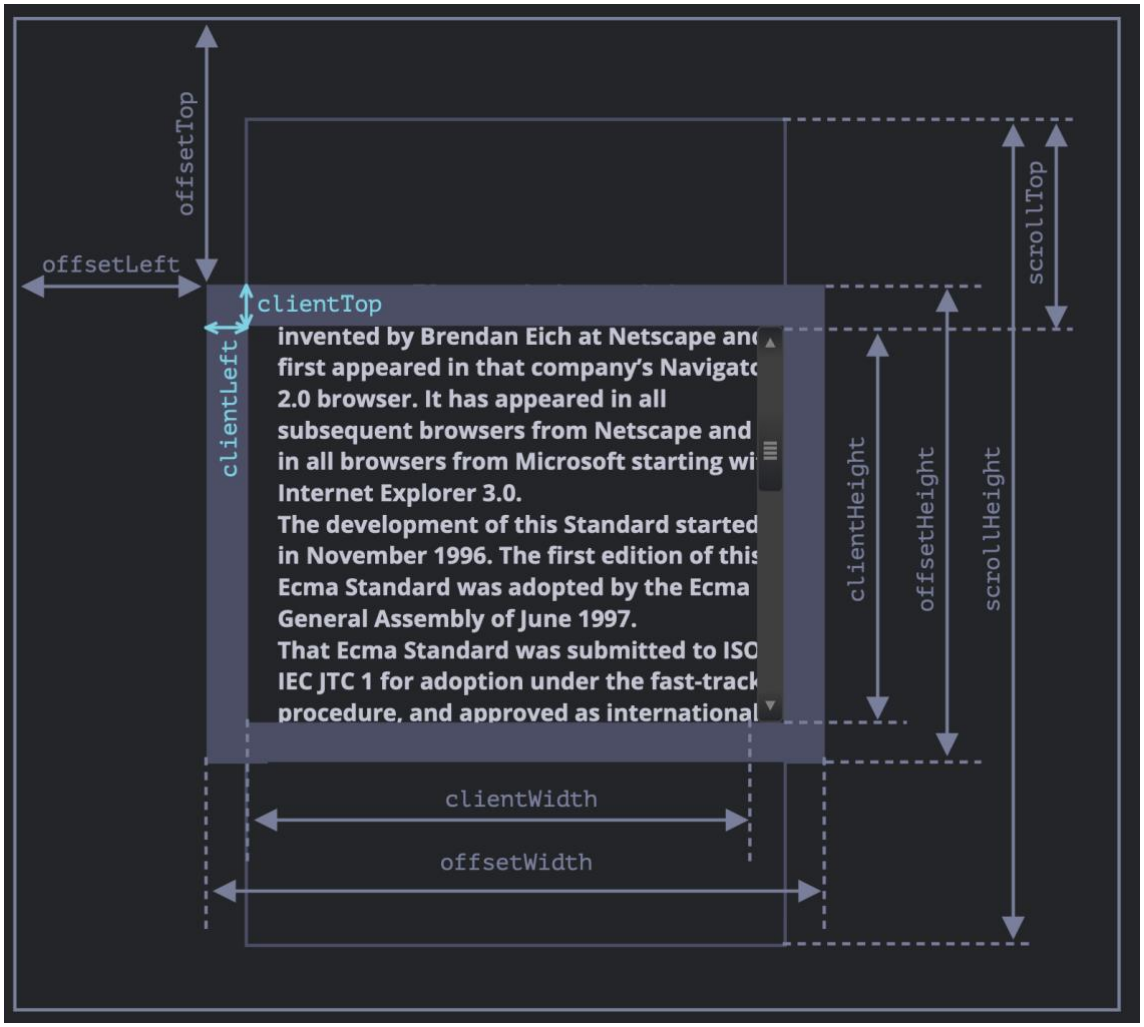
Este té vora, farciment i desplaçament. El conjunt complet de funcions. No hi ha marges perquè no són part de l'element en si, i no tenen propietats especials.

L'element té este aspecte:



7.2.Geometria

Ací està la imatge general amb propietats geomètriques:



Els valors d'estes propietats són tècnicament números, però estos números són "de píxels", així que estes són mesures de píxels.

Comencem a explotar les propietats, iniciant des de l'exterior de l'element.

7.3.offsetParent, offsetLeft/Top

Estes propietats són rarament necessàries, però encara són les propietats de geometria "més externes" així que començarem amb elles.

L'offsetParent és l'avantpassat més pròxim que usa el navegador per a calcular les coordenades durant el renderitzat.

Eixe és l'avantpassat més pròxim que és un dels següents:

- Posicionat per CSS (position és absolute, relative, fixed o sticky), o…
- <td>, <th>, or <table>, o…
- <body>.

Les propietats offsetLeft/offsetTop proporcionen coordenades x/i relatives a la cantonada superior esquerra d'offsetParent .

En el següent exemple el més intern té <main> com offsetParent, i offsetLeft/offsetTop ho desplaça des de la seua cantonada superior esquerra (180):

```
<main style="position: relative" id="main">
<article>
<div id="example" style="position: absolute; left: 180px; top: 180px">...</div>
</article>
</main>
<script>
alert(example.offsetparent.id); // main
alert(example.offsetLeft); // 180 (nota: és un número, no un string "180px")
alert(example.offsetTop); // 180
</script>
```

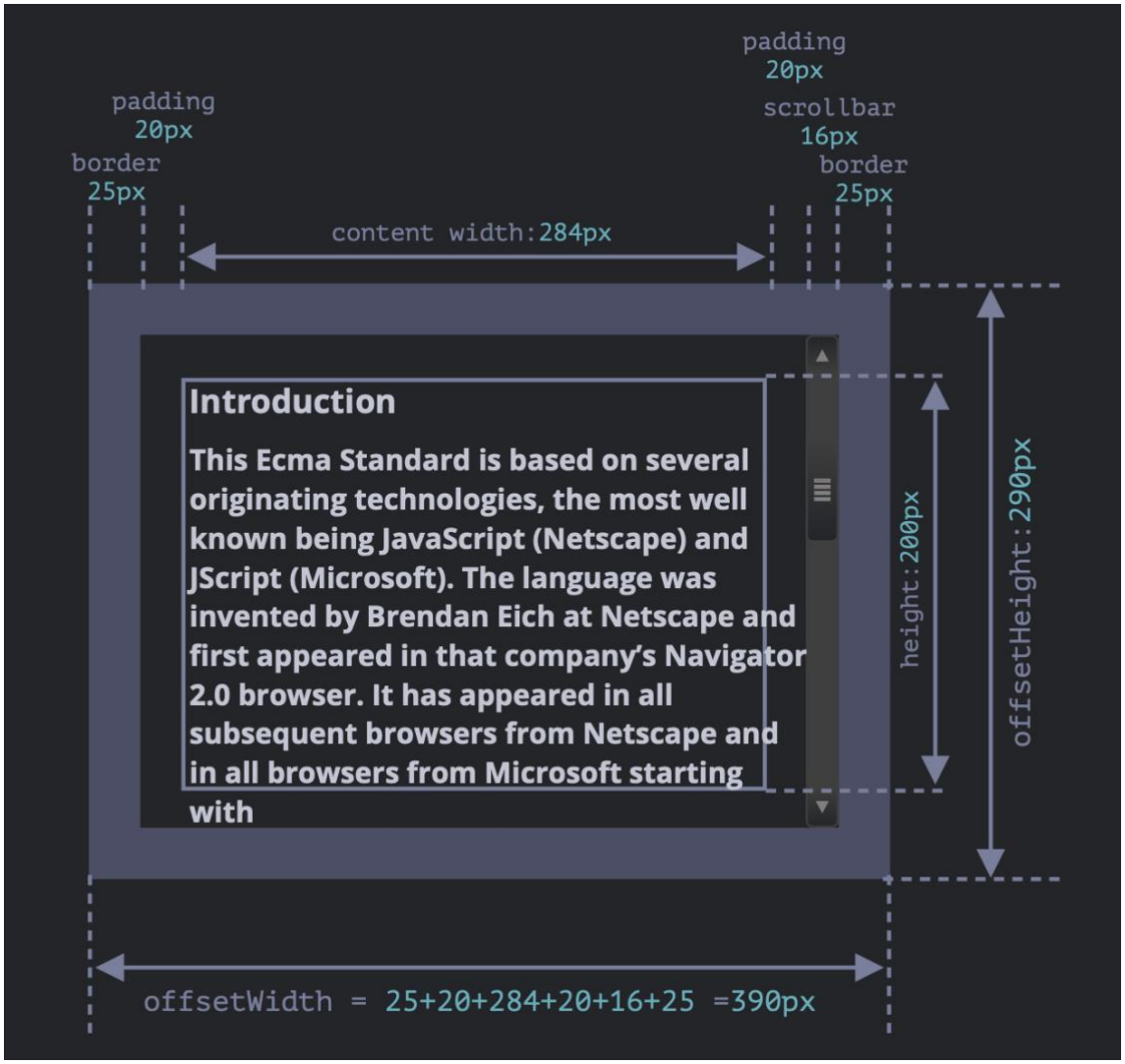
Hi ha diverses ocasions en la qual offsetParent és null:

- Per a elements no mostrats (display:none o no en el document).
- Per a <body> i <HTML>.
- Per a elements amb position:fixed.

7.4.offsetWidth/Height

Ara passem a l'element en si.

Estes dues propietats són les més simples. Proporcionen l'ample i alt "exterior" de l'element. O, en altres paraules, la seua grandària completa, inclosos les vores.



Per al nostre element de mostra:

- offsetWidth = 390 – l'ample exterior, pot ser calculat com CSS-width intern (300px) més acolchonados (2 * 20px) i vores (2 * 25px).
- offsetHeight = 290 – l'alt exterior.

7.5.clientTop/Left

Dins de l'element, tenim les vores.

Per a mesurar-los, estan les propietats clientTop i clientLeft.

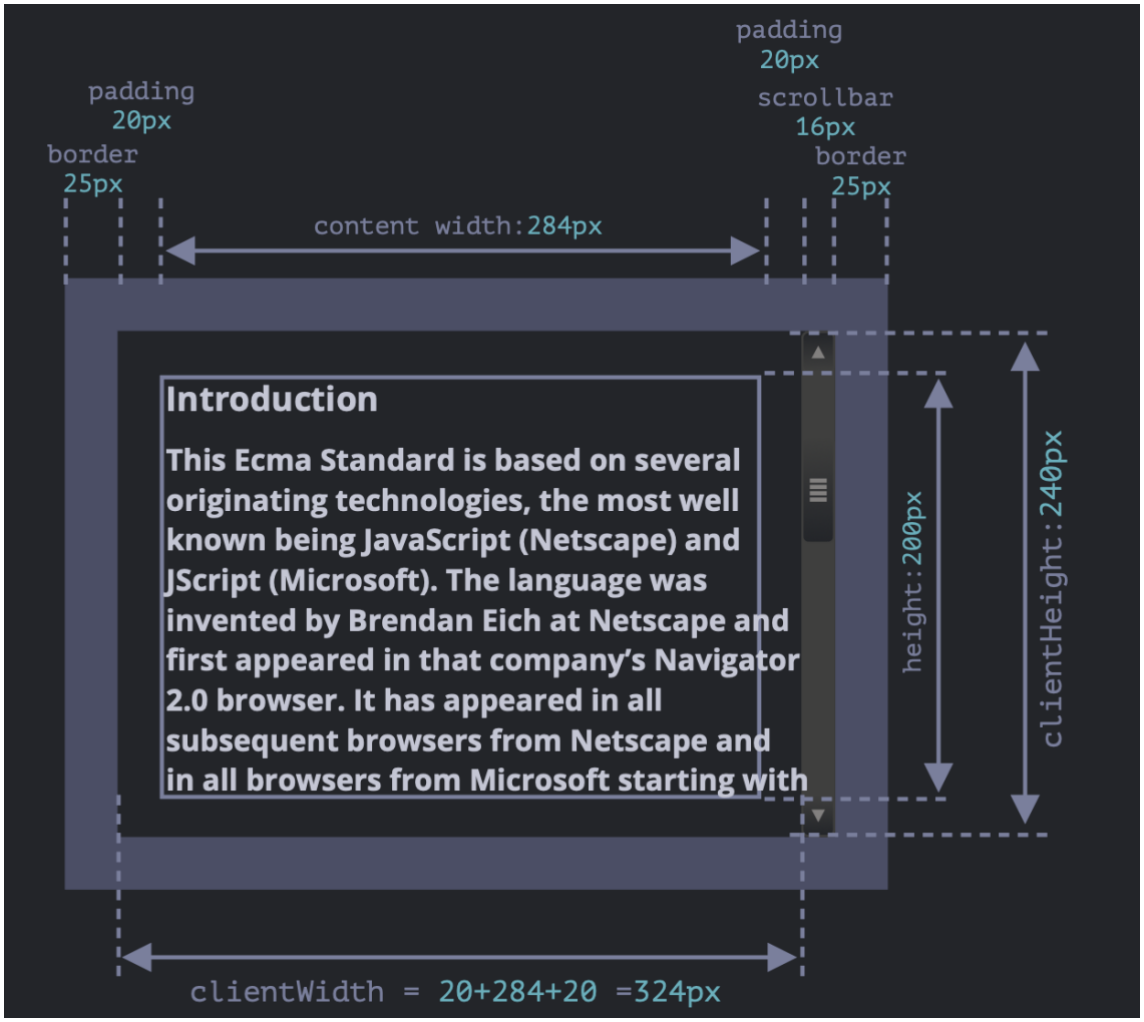
En el nostre exemple:

- clientLeft = 25 – ample de la vora esquerra
- clientTop = 25 – ample de la vora superior

7.6.clientWidth/Height

Esta propietat proporciona la grandària de l'àrea dins de les vores de l'element.

Inclouen l'ample del contingut juntament amb els farciments, però sense la barra de desplaçament:

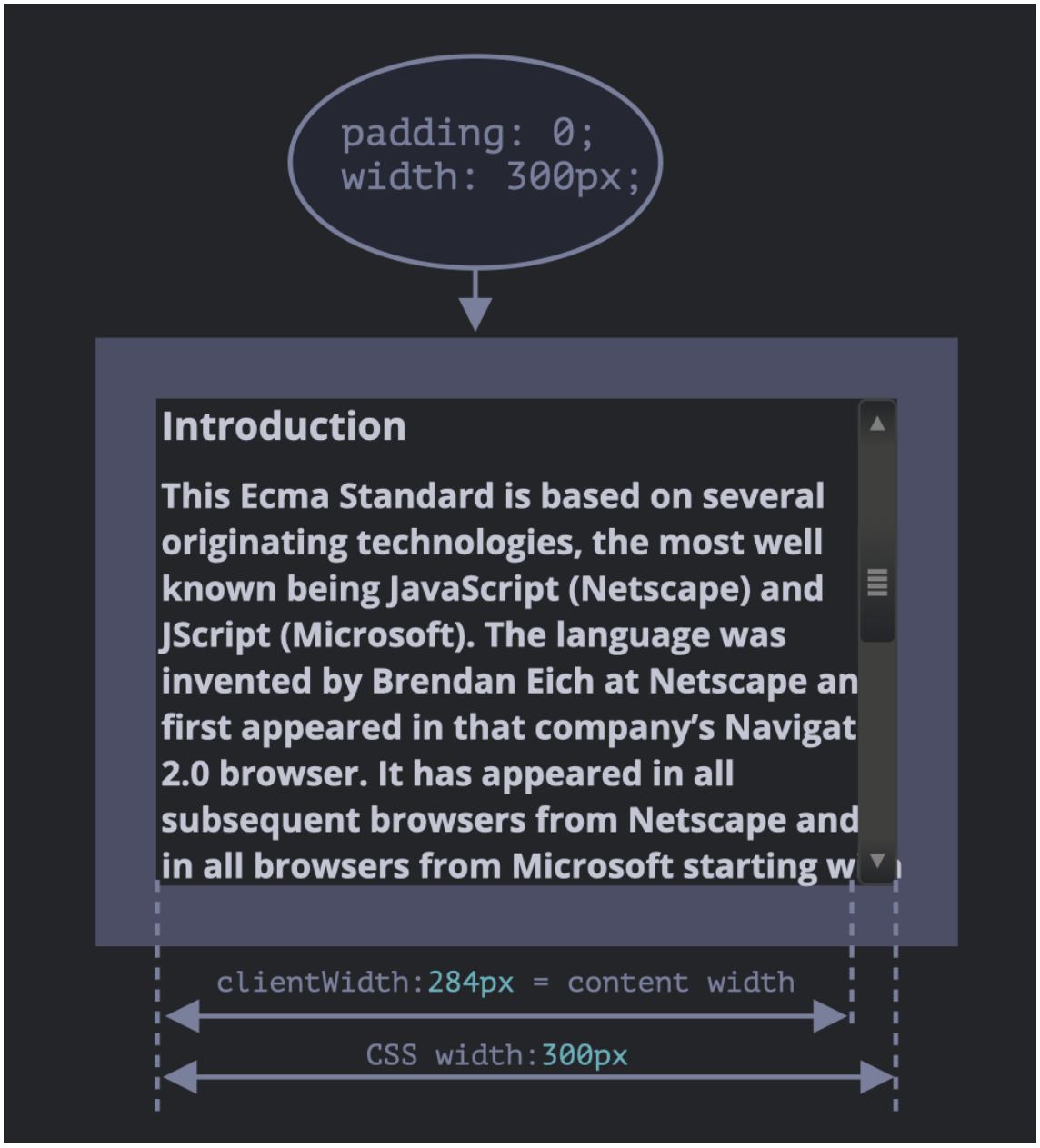


En la imatge de dalt, considerem primer clientHeight.

No hi ha una barra de desplaçament horitzontal, per la qual cosa és exactament la suma del que està dins de les vores: CSS-height 200px més el farciment superior i inferior (2 * 20px) totalitza 240px.

Ara clientWidth: ací l'ample del contingut no és 300px, sinó 284px, perquè els 16px són ocupats per la barra de desplaçament. Llavors la suma és 284px més els farciments d'esquerra i dreta, total 324px.

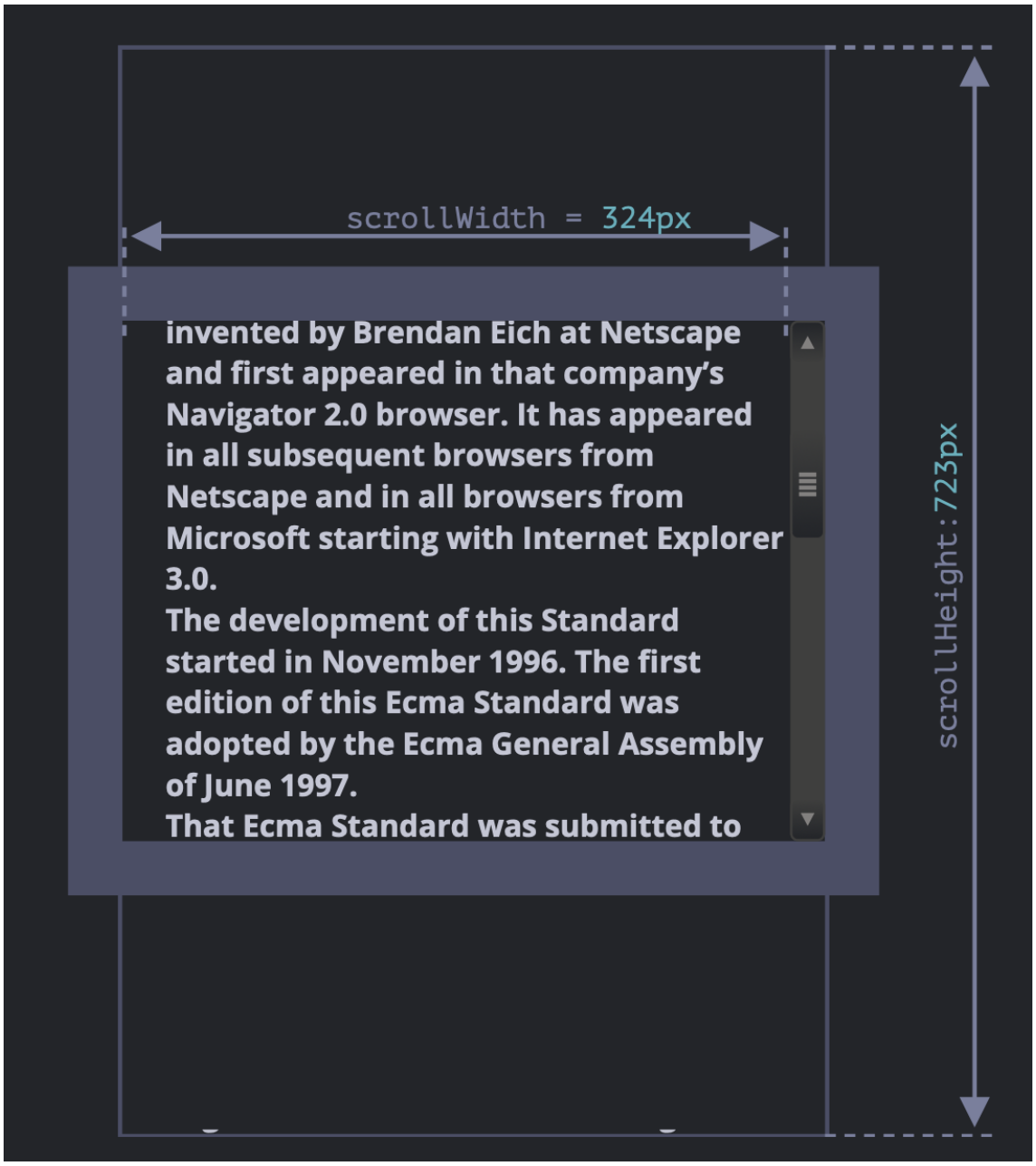
Si no hi ha farcits, llavors clientWidth/Height és exactament l'àrea de contingut, dins de les vores i la barra de desplaçament (si n'hi ha).



Llavors, quan no hi ha farcit, podrem usar clientWidth/clientHeight per a obtindre la grandària de l'àrea de contingut.

7.7.scrollWidth/Height

Estes propietats són com clientWidth/clientHeight, però també inclouen les parts desplaçades (ocultes):



En la imatge de dalt:

- scrollHeight = 723 – és l'altura interior completa de l'àrea de contingut, incloent-hi les parts desplaçades.
- scrollWidth = 324 – és l'ample interior complet, ací no tenim desplaçament horitzontal, per la qual cosa és igual a clientWidth.

Podem usar estes propietats per a expandir l'element al seu ample/alt complet.

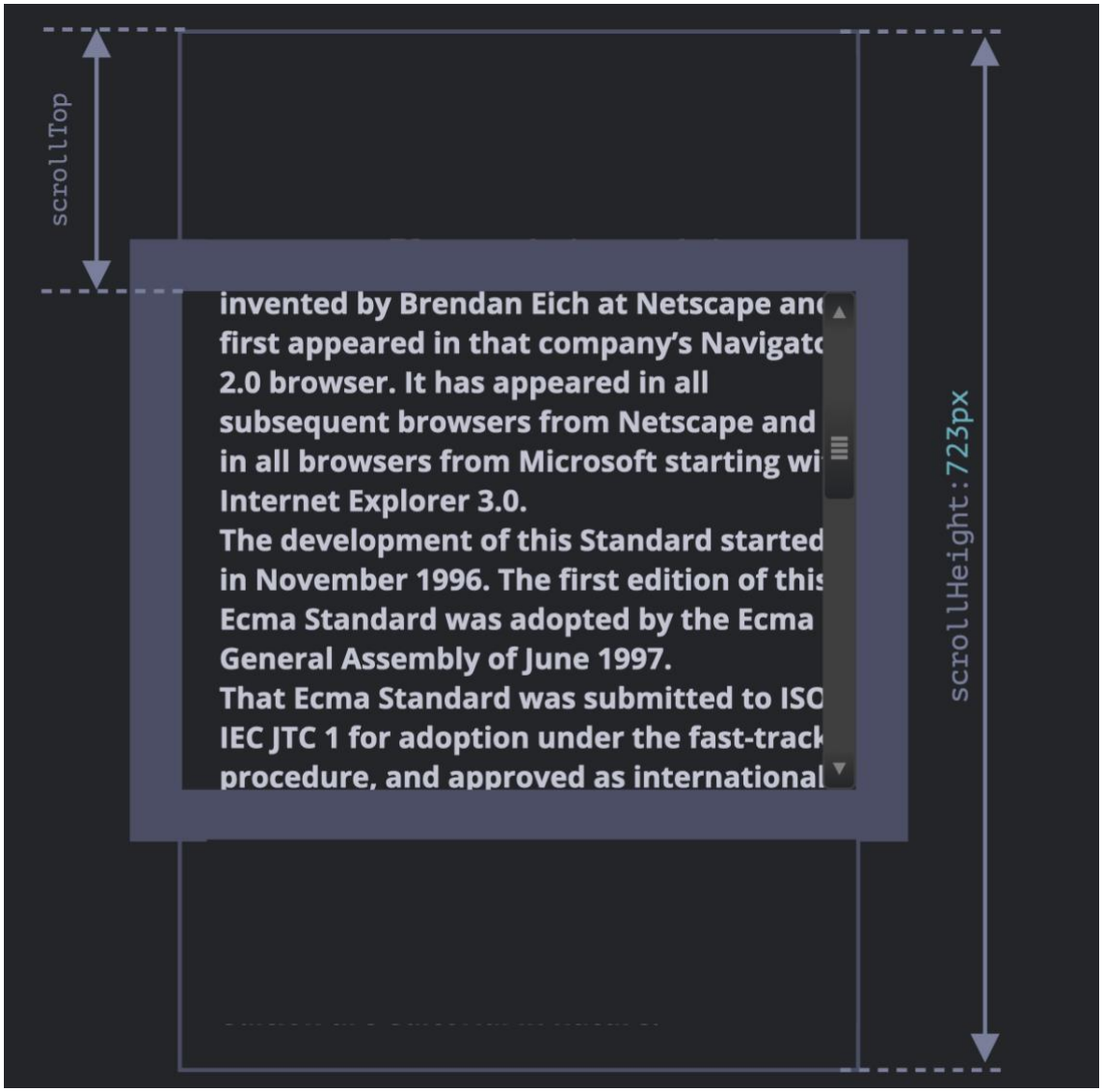
Com això:

```
// expandisca l'element a l'altura completa del contingut
element.style.height = `${element.scrollHeight}px`;
```

7.8. scrollLeft/scrollTop

Les propietats scrollLeft/scrollTop són l'ample/alt de la part oculta i desplaçada de l'element.

En la imatge a baix podem veure scrollHeight i scrollTop per a un bloc amb un desplaçament vertical.



En altres paraules, scrollTop és "quant es desplaça cap amunt".

8. GRANDÀRIA DE FINESTRA I DESPLAÇAMENT

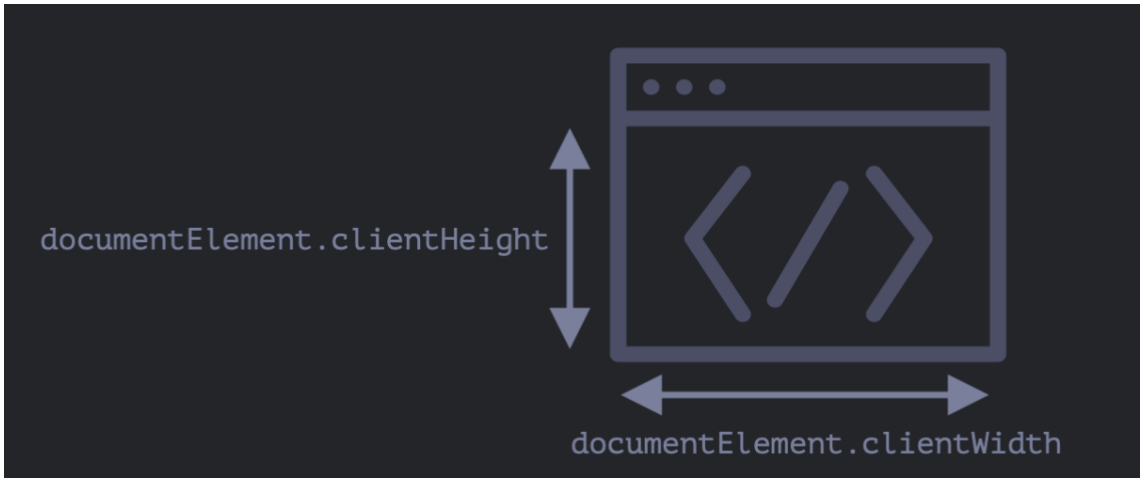
Com trobem l'ample i l'alt de la finestra del navegador? Com obtenim tot l'ample i l'altura del document, inclosa la part desplaçada? Com desplacem la pàgina usant JavaScript?

Per a la majoria d'estes qüestions, podem usar l'element de document arrel document.documentElement, que correspon a l'etiqueta <HTML>. Però hi ha mètodes i peculiaritats addicionals prou importants per a considerar.

8.1. Ample/alt de la finestra

Per a obtindre l'ample i alt de la finestra, podem usar clientWidth / clientHeight de document.documentElement:

Per exemple, l'altura de la seua finestra:




```
alert(document.documentElement.clientHeight)
```

8.2. Ample/Alt del document

Teòricament, com l'element del document arrel és document.documentElement, i inclou tot el contingut, podríem mesurar la grandària completa del document amb document.documentElement.scrollWidth / scrollHeight.

Però en eixe element, per a tota la pàgina, estes propietats no funcionen segons el que es preveu. En Chrome/Safari/Opera si no hi ha desplaçament, llavors documentElement.scrollHeight pot ser fins i tot menor que documentElement.clientHeight! Sona com una ximpleria, rar, veritat?

Per a obtindre de manera de confiança l'altura completa del document, hem de prendre el màxim d'estes propietats:

```
let scrollHeight = Math.max(
document.body.scrollHeight, document.documentElement.scrollHeight,
document.body.offsetHeight, document.documentElement.offsetHeight,
document.body.clientHeight, document.documentElement.clientHeight
);

alert('Altura completa del document, amb part desplaçada: ' + scrollHeight);
```

Per què? Millor no preguntes. Estes inconsistències provenen de temps antics, no una lògica “intel·ligent”.

8.3. Obtindre el desplaçament actual

Els elements DOM tenen el seu estat de desplaçament actual en les seues propietats elem.scrollLeft/scrollTop.

El desplaçament de documents, document.documentElement.scrollLeft / Top funciona en la majoria dels navegadors, excepte els més antics basats en WebKit, com a Safari (bug 5991), on hauríem d'usar document.body en lloc de document.documentElement.

Afortunadament, no hem de recordar estes peculiaritats en absolut, perquè el desplaçament està disponible en les propietats especials window.pageXOffset/pageYOffset:

```
alert('Desplaçament actual des de la part superior: ' + window.pageYOffset);
alert('Desplaçament actual des de la part esquerra: ' + window.pageXOffset);
```

Estes propietats són de només lectura.

També disponible com a propietats window: scrollX i scrollY

Per raons històriques existixen totes dues propietats, però ambdues són el mateix:

```
window.pageXOffset és un àlies de window.scrollX.
window.pageYOffset és un àlies de window.scrollY.
```

8.4. Desplaçament: scrollTo, scrollBy, scrollIntoView

Per exemple, si intentem desplaçar la pàgina des del script en <head>, no funcionarà.

Els elements regulars es poden desplaçar canviant scrollTop/scrollLeft.

Nosaltres podem fer el mateix per a la pàgina usant document.documentElement.scrollTop/Left (excepte Safari, on document.body.scrollTop/Left hauria d'usar-se en el seu lloc).

Alternativament, hi ha una solució més simple i universal: mètodes especials window.scrollBy(x,i) i window.scrollTo(pageX,pageY).

- El mètode scrollBy(x, i) desplaça la pàgina en relació amb la seua posició actual. Per exemple, scrollBy(0,10) desplaça la pàgina 10px cap avall.

- El mètode scrollTo(pageX, pageY) desplaça la pàgina a coordenades absolutes, de manera que la cantonada superior esquerra de la part visible té coordenades (pageX, pageY) en relació amb la cantonada superior esquerra del document. És com configurar scrollLeft / scrollTop. Per a desplaçar-nos fins al principi, podem usar scrollTo(0,0).

Estos mètodes funcionen per a tots els navegadors de la mateixa manera.