


Présentation

BOM - DOM

DOM 3

• **Browser Object Model**

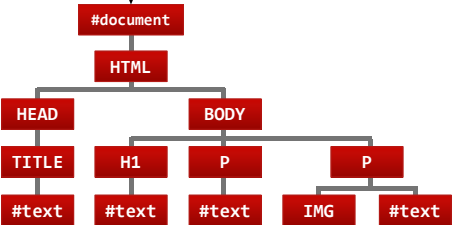
Représentation de l'interface navigateur sous la forme d'une hiérarchie d'objets.



• **Document Object Model**

Représentation de la page affichée sous la forme d'un arbre d'objets.

```
<html>
<head>
  <title>Exemple</title>
</head>
<body>
  <h1>Le DOM</h1>
  <p>Modèle hiérarchique</p>
  <p>Un sac de noeuds</p>
</body>
</html>
```

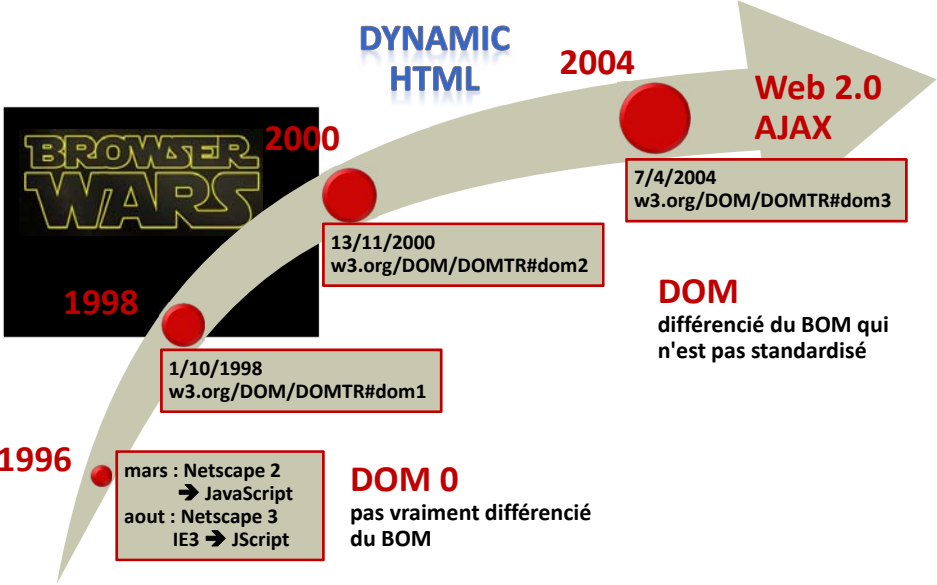


francois.piat@univ-fcomte.fr

Présentation

Evolution

DOM 4



1996 mars : Netscape 2 → JavaScript
aout : Netscape 3
IE3 → JScript

1998 1/10/1998
w3.org/DOM/DOMTR#dom1

2000 13/11/2000
w3.org/DOM/DOMTR#dom2

2004 7/4/2004
w3.org/DOM/DOMTR#dom3

Web 2.0 AJAX

DOM
différencié du BOM qui n'est pas standardisé

DOM 0
pas vraiment différencié du BOM

francois.piat@univ-fcomte.fr

2

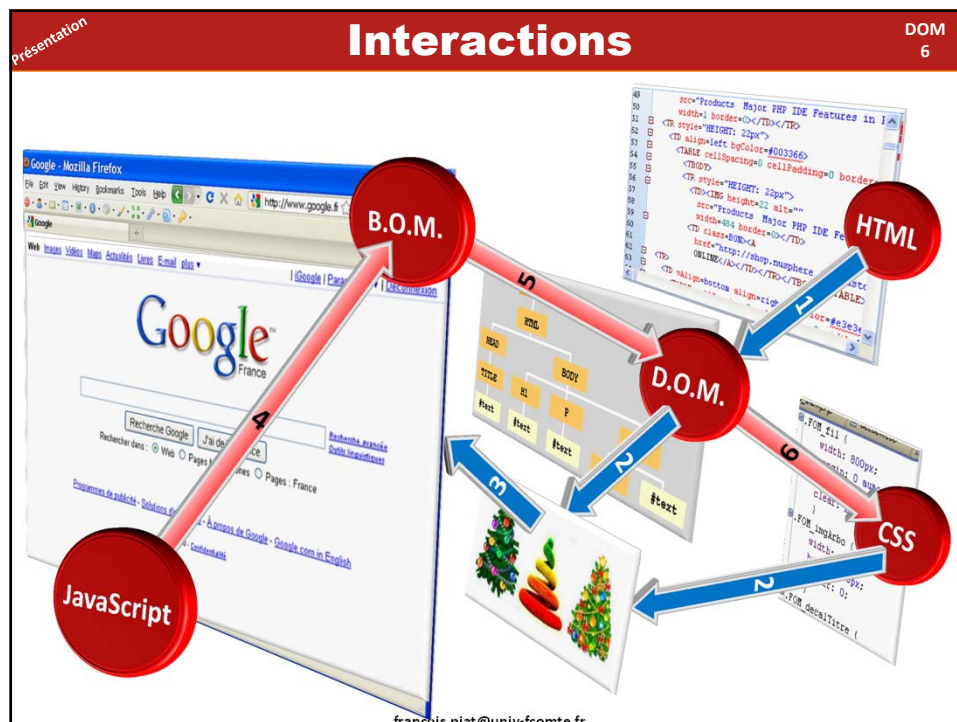
Présentation

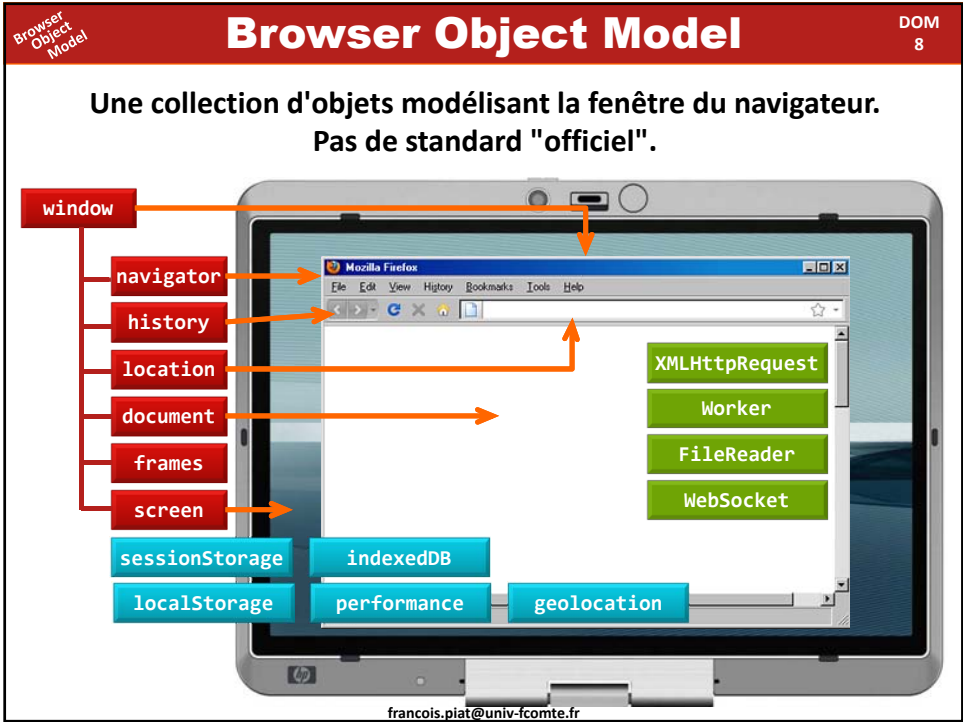
Evolution

DOM
5

- Le règne de Microsoft (2000-2008) a été une bonne chose car la course incessante des navigateurs a été arrêtée :
 - on a pris le temps de corriger des bugs
 - pas de nouvelles incompatibilités
 - les navigateurs deviennent plus stables
- Comme il l'a fait pour HTML et CSS, le W3C tente de définir un standard pour unifier les pratiques et les développements.

francois.piat@univ-fcomte.fr





Browser Object Model

window : objet Global

DOM 9

- Contexte d'exécution de JavaScript.
- **Tout** ce qui est manipulé par JavaScript est une propriété ou une méthode de l'objet **window**.
- Pas de persistance de données : le contenu de l'objet global est totalement réinitialisé au chargement d'une page.
- Pas besoin de préciser l'objet pour invoquer ses méthodes et travailler avec ses propriétés.

`alert('Bonjour');` \Leftrightarrow `window.alert('Bonjour');`

francois.piat@univ-fcomte.fr
<https://developer.mozilla.org/en/DOM/window>

Browser Object Model

window : fenêtre navigateur

DOM 10


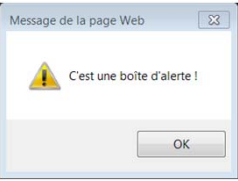
valeur scalaire
objet

Browser Object Model


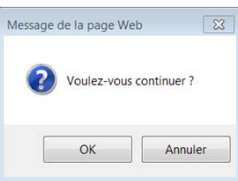
window : méthodes "message"

DOM 11

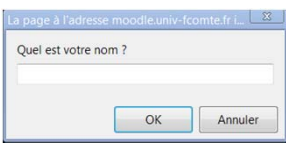
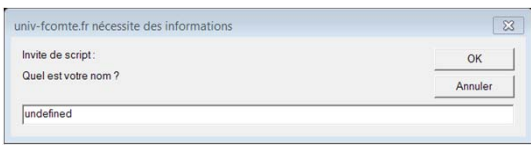
`alert("C'est une boîte d'alert !");`

`confirm('Voulez-vous continuer ?');`

`prompt('Quel est votre nom ?');`

francois.piat@univ-fcomte.fr

Browser Object Method

window : méthodes "minuteur"

DOM 12

`setTimeout(fonction, millisecondes)`
`setInterval(fonction, millisecondes)`

```

timer1 = setTimeout(maFonction, 1000);

timer2 = setTimeout("autreFonction(10, 'abc')", 2000);

timer3 = setTimeout(function(10, 'abc') {
    ...Code de la fonction...
}, 2000);
        
```

`clearTimeout(référence timer)`
`clearInterval(référence timer)`

```

clearTimeout(timer1);
clearTimeout(timer2);
clearTimeout(timer3);
        
```

francois.piat@univ-fcomte.fr

Browser Object Model

window : méthodes "popup"

DOM 13

open(url, "nom", "options de la fenêtre")

Options sous la forme : "option[=valeur], ..., ..."

height=NbPixels	menubar
width=NbPixels	resizable
left=NbPixels	scrollbars
top=NbPixels	status
location	toolbar

La méthode **open()** renvoie un handler (cf pointeur) sur la fenêtre ouverte.

francois.piat@univ-fcomte.fr

Browser Object Model

popup window

DOM 14

```
var pop = open('index.html', 'tutoJS',
'width=520, height=800, left=900, top=150, scrollbars, resizable');
```

Fenêtre principale
Propriété opener
de la fenêtre pop

Fenêtre pop

francois.piat@univ-fcomte.fr

Browser Object Model

window : méthodes "popup"

DOM 15

- Le handler renvoyé par la méthode **open()** permet de :
 - manipuler la fenêtre popup depuis la fenêtre principale,
 - échanger des informations entre les fenêtres.

```
var pop = open('index.html', 'tutoJS', 'width=520, height=800, left=900, top=150, scrollbars, resizable');
```

pop.focus();

pop.blur();

pop.moveTo(200, 50);

pop.resizeTo(600, 500);

if (!pop.closed) {
 pop.close();
}

pop.uneVariable = 10;

pop.uneFonction();

francois.piat@univ-fcomte.fr

Browser Object Model

Objet location

DOM 16

- Propriétés

href

{
http://www.site.fr/repert/page.php?num=123
}

protocol

host

path

search

location.href = 'url/autre/page';

⇒

Le navigateur charge et affiche url/autre/page
- Méthodes

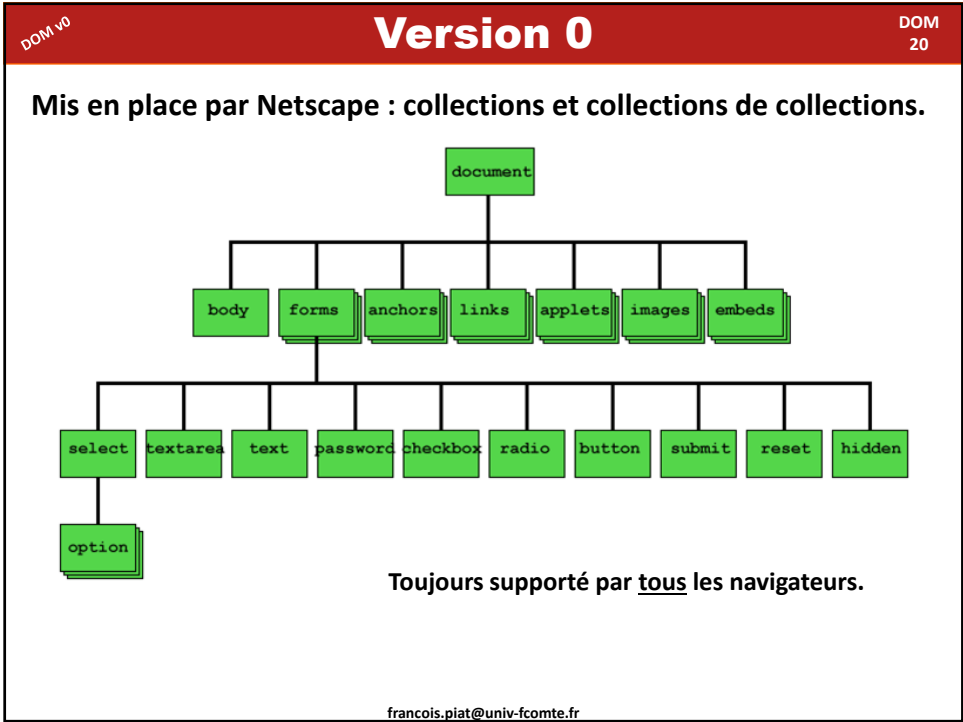
location.replace('url')

location.reload()

francois.piat@univ-fcomte.fr

Objet navigator			
Browser Object Model		DOM 17	
	appCodeName	appName	userAgent
Firefox 18	Mozilla	Netscape	Mozilla/5.0 (Windows NT 6.1; WOW64; rv:18.0) Gecko/20100101 Firefox/18.0
IE 8	Mozilla	Microsoft Internet Explorer	Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; WOW64; Trident/4.0; .NET CLR 2.0.50727; ...)
IE 9	Mozilla	Microsoft Internet Explorer	Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; WOW64; Trident/5.0; .NET CLR 2.0.50727; ...)
IE 10	Mozilla	Microsoft Internet Explorer	Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.1; WOW64; Trident/6.0; SLCC2; .NET C...)
Safari 5	Mozilla	Netscape	Mozilla/5.0 (Windows; U; Windows NT 6.1; tr-TR) AppleWebKit/533.20.25 (KHTML, like Gecko) Version/5.0.4 Safari/533.20.27
Chrome 24	Mozilla	Netscape	5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.17 (KHTML, like Gecko) Chrome/24.0.1312.57 Safari/537.17
Opera 11	Mozilla	Opera	Opera/9.80 (Windows NT 6.1; U; fr) Presto/2.10.229 Version/11.61
http://www.useragentstring.com/			
francois.piat@univ-fcomte.fr			

Objet document	
Browser Object Model	
DOM 18	
<ul style="list-style-type: none"> ● Représente la page affichée dans la fenêtre. ● Sert de support au DOM 0. ● Sert de support au DOM W3C. 	
francois.piat@univ-fcomte.fr	



DOM v0
Version 0
DOM 21

- Le DOM niveau 0 fut inventé par Netscape en même temps que JavaScript et fut mis en application la première fois sur Netscape 2 - 1996.
- Il offre l'accès à quelques éléments HTML, d'une manière plus importante aux formes et (plus tard) aux images.
- Tous les éléments ne sont pas accessibles, par exemple les éléments de structure (<p>, <h>, <table>), les éléments de mise en forme (,).
- Microsoft fut d'abord forcé de copier le DOM de Netscape dans Internet Explorer 3 pour assurer la compatibilité.
- Ces collections sont toujours supportées par les navigateurs actuels.
- Par conséquent le DOM de niveau 0 est vraiment unifié. Avec les DOM suivants, cette situation a changé.

On a ici qu'une partie de la hiérarchie car au départ on trouvait aussi la fenêtre du navigateur, le navigateur et d'autres éléments externes au document lui-même. Maintenant on appelle cela le BOM : Browser Object Model.

```

graph TD
    document[document] --> body[body]
    document --> forms[forms]
    document --> anchors[anchors]
    document --> links[links]
    document --> applets[applets]
    document --> images[images]
    document --> embeds[embeds]
    forms --> select[select]
    forms --> textarea[textarea]
    forms --> text[text]
    forms --> password[password]
    forms --> checkbox[checkbox]
    forms --> radio[radio]
    forms --> button[button]
    forms --> submit[submit]
    forms --> reset[reset]
    forms --> hidden[hidden]
    select --> option[option]
        
```

francois.piat@univ-fcomte.fr

DOM v0
Objets DOM
DOM 22

- Tous les éléments HTML ne sont pas accessibles.
- Les éléments HTML sont représentés par des objets avec des méthodes et des propriétés.
- Les méthodes et les propriétés sont différentes suivant le type de l'élément.
- Toutes les propriétés sont lisibles, mais toutes ne sont pas modifiables.
- Certains objets gèrent aussi des événements :
onclick, onmouseover, onkeypress, onload, onsubmit, ...

francois.piat@univ-fcomte.fr

DOM v0

Objets DOM

DOM 23

- JavaScript a accès à ces objets et peut agir sur leur propriété pour changer ce qui est affiché dans le navigateur.
- Les éléments des collections sont accessibles par un indice numérique.
- L'indice numérique est défini par la place de l'éléments dans le code HTML.
- Les éléments des collections sont accessibles également par leur nom ou leur id si ils possèdent l'attribut correspondant (**id** ou **name**).

francois.piat@univ-fcomte.fr

DOM v0

Accès avec indice numérique

DOM 24

```
document.collection[0].propriete = "Valeur";
document.collection[0].methode();
```

document.images[0]

↓

↑

document.images[1]

```
document.images[0].width = 200;

var gif = document.images[1];
gif.width = 32;
gif.height = 32;
gif.src = "smilley.gif";
```

francois.piat@univ-fcomte.fr

DOM v0

Accès avec le nom

DOM 25

- L'élément doit avoir un attribut **name** ou **id**.


```
document.images['moi']
```

↓

```


```

↑

```
document.images['btn']
```

```
document.images['moi'].width = 200;
document.images['btn'].src = "smilley.gif";
```
- La valeur des attributs **name** ou **id** doit être unique dans la page.


```


```

↑

Cet élément ne pourra jamais être atteint.

francois.piat@univ-fcomte.fr

DOM v0

Accès avec la syntaxe à point

DOM 26

- L'élément doit avoir un attribut **name** ou **id**.


```
document.moi
```

↓

```


```

↑

```
document.btn
```

```
document.moi.width = 200;
document.btn.src = "smilley.gif";
```
- La valeur des attributs **name** ou **id** doit être unique dans la page.


```


```

↑

Cet élément ne pourra jamais être atteint.

francois.piat@univ-fcomte.fr

DOM v0

Hiérarchie

DOM 27

```
<form name="frm">
  Nom : <input type="text" name="nom" value="Piat">
  <input type="submit" name="btn" value="Valider">
</form>
```

```
document.forms[0].elements[0].value
document.forms['frm'].elements['nom'].value
document.forms['frm'].nom.value
document.frm.nom.value

document.forms[0].elements['btn'].value
document.forms['frm'].elements[1].value
document.forms.frm.elements['btn'].value
document.frm.btn.value
```

}

Piat

}

Valider

francois.piat@univ-fcomte.fr

DOM v0

Événements

DOM 28

- Certains objets gèrent des événements.
- Gestionnaires d'événement dans le code HTML.

```
<body onload="fonctionAExecuter(123)">

```

- Gestionnaires d'événement dans le code JavaScript

```
window.onload = fonctionAExecuter;

window.onload = function(123) {
  ...code de la fonction...
};

document.images[0].onclick = function() {
  alert('Bonjour');
};
```

francois.piat@univ-fcomte.fr

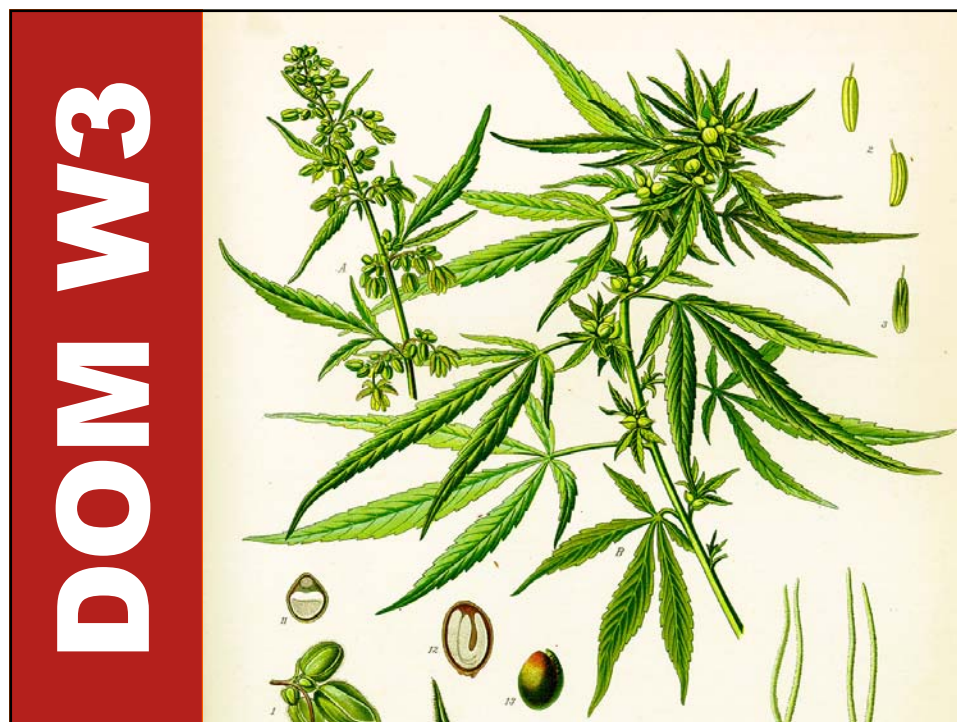
DOM v0

Dynamic HTML

DOM 29

- Avec IE 4, Microsoft rend accessible tous les éléments HTML.
- Les propriétés de style CSS des éléments HTML sont accessibles à travers la propriété **style** des objets.
- Si les propriétés de style ne sont pas définies, on a des propriétés propres aux objets.
- La modification des propriétés de style est immédiatement prise en compte par le navigateur.
- Tous les éléments gèrent des événements.

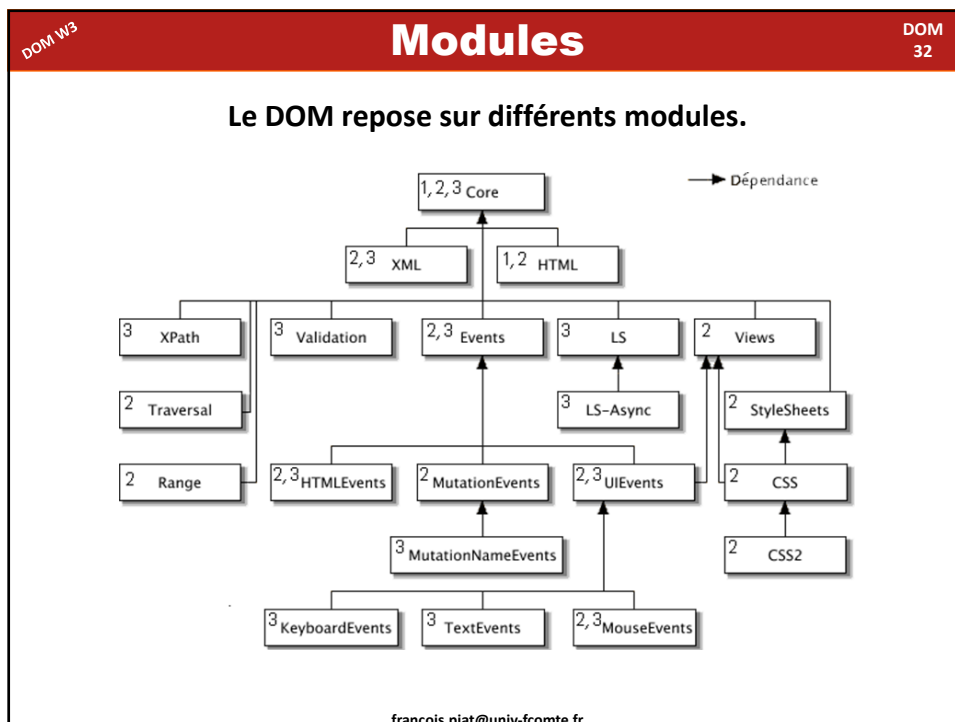
francois.piat@univ-fcomte.fr



DOM W3
Document Object Model
DOM 31

- "Avec le DOM, les programmeurs peuvent
 - **construire** des documents,
 - **naviguer** dans leur structure,
 - **ajouter, modifier, ou supprimer** soit des éléments soit du contenu."
- 3 versions :
 - DOM niveau 1 - Octobre 1998 - www.w3.org/DOM/DOMTR#dom1
 - DOM niveau 2 - Novembre 2000 - www.w3.org/DOM/DOMTR#dom2
 - DOM niveau 3 - Avril 2004 - www.w3.org/DOM/DOMTR#dom3
 - DOM niveau 4 – toujours en brouillon - <http://dvcs.w3.org/hg/domcore/raw-file/tip/Overview.html>
 - Aucun navigateur n'implémente complètement aucune version
- Accès à tous les éléments du document, qu'ils soient ou non référencés par un attribut **id** ou **name**

francois.piat@univ-fcomte.fr



DOM W3

Modules

DOM 33

- Les modules sont spécialisés dans le traitement d'un domaine particulier. Il sont transparent pour le développeur final.
- Les navigateurs n'implémentent pas tous les modules.
- Le Core DOM (cœur du DOM, noyau) est l'interface de plus bas niveau. Elle repose sur la représentation du document sous la forme d'un arbre, et permet de traverser la hiérarchie des éléments de la structure.
- Le DOM HTML fournit les moyens de manipuler aisément les documents HTML.
- Le DOM Events (événements) permet de gérer les événements qui se produisent lors de la modification de la structure de l'arbre (Mutation Events), du rendu HTML (HTML Events) ou des événements utilisateurs (UIEvents : User Interface Events).
- Le DOM CSS permet d'accéder aux propriétés des feuilles de styles et d'en modifier les valeurs.

```
graph TD
    Core["1, 2, 3 Core"]
    XML["2, 3 XML"]
    HTML["1, 2 HTML"]
    XPath["3 XPath"]
    Validation["3 Validation"]
    Events["2, 3 Events"]
    LS["3 LS"]
    Views["2 Views"]
    Traversal["2 Traversal"]
    Range["2 Range"]
    HTMLEvents["2, 3 HTMLEvents"]
    MutationEvents["2 MutationEvents"]
    LSAsync["3 LS-Async"]
    StyleSheets["2 StyleSheets"]
    CSS["2 CSS"]
    CSS2["2 CSS2"]
    KeyboardEvents["3 KeyboardEvents"]
    TextEvents["3 TextEvents"]
    MouseEvents["2, 3 MouseEvents"]

    Core --> XML
    Core --> HTML
    XML --> XPath
    XML --> Validation
    HTML --> Events
    HTML --> LS
    HTML --> Views
    Events --> HTMLEvents
    Events --> MutationEvents
    Events --> LSAsync
    Views --> StyleSheets
    Views --> CSS
    Views --> CSS2
    HTMLEvents --> KeyboardEvents
    HTMLEvents --> TextEvents
    MutationEvents --> MouseEvents
```

francois.piat@univ-fcomte.fr

DOM W3

Interfaces

DOM 34

- Les modules définissent des interfaces et pas des classes.

```
classDiagram
    class Document {
        +doctype [READONLY] : DOMString
        +documentElement [READONLY] : Element
        +createElement(tagName : DOMString) : Element
        +createDocumentFragment() : DocumentFragment
        +createTextNode(data : DOMString) : Text
        +createComment(data : DOMString) : Comment
        +createCDATASection(data : DOMString) : CDATASection
        +createPI(target : DOMString, data : DOMString) : PI
        +createAttribute(name : DOMString) : Attr
        +createEntityReference(name : DOMString) : EntityReference
        +getElementsByName(tagName : DOMString) : NodeList
    }
    class Node {
        +nodeName [READONLY] : DOMString
        +nodeValue : DOMString
        +nodeType [READONLY] : unsigned short
        +parentNode [READONLY] : Node
        +childNodes [READONLY] : NodeList
        +firstChild [READONLY] : Node
        +lastChild [READONLY] : Node
        +previousSibling [READONLY] : Node
        +nextSibling [READONLY] : Node
        +attributes [READONLY] : NamedNodeMap
        +ownerDocument [READONLY] : Document
        +insertBefore(newChild : Node, refChild : Node) : Node
        +replaceChild(newChild : Node, oldChild : Node) : Node
        +removeChild(oldChild : Node) : Node
        +appendChild(newChild : Node) : Node
        +hasChildNodes() : boolean
        +cloneNode(deep : boolean) : Node
    }
    class Element {
        +tagName [READONLY] : DOMString
        +getAttribute() : DOMString
        +setAttribute(name : DOMString, value : DOMString) : void
        +removeAttribute(name : DOMString) : void
        +getAttributeNode(name : DOMString) : Attr
        +setAttribute(newAttr : Attr) : Attr
        +removeAttributeNode(oldAttr : Attr) : Attr
        +getElementsByName(name : DOMString) : NodeList
        +normalize() : void
    }
    class CharacterData {
        +data : DOMString
        +substringData(offset : u_long, count : u_long) : DOMString
        +appendData(arg : DOMString) : void
        +insertData(offset : u_long, arg : DOMString) : void
        +deleteData(offset : u_long, count : u_long) : void
        +replaceData(offset : u_long, count : u_long, arg : DOMString) : void
    }
    class DocumentFragment {
    }
    class Attr {
        +name [READONLY] : DOMString
        +specified [READONLY] : boolean
        +value : DOMString
    }
    Document --> Node
    Node --> Element
    Node --> CharacterData
    Node --> DocumentFragment
    Node --> Attr
```

- Comme on travaille avec des interfaces, n'importe quel langage peut implémenter une gestion DOM (JavaScript, Java, C++, etc).

francois.piat@univ-fcomte.fr

DOM W3

Interfaces

DOM 35

- Une interface définit pour un objet les méthodes et les propriétés qui permettent aux "utilisateurs" de cet objet de travailler avec lui. Quand une interface est implémentée dans un objet, on est sûr de trouver comme méthodes de l'objets les méthodes de l'interface qu'il implémente.
- Certaines interfaces en implémente d'autres. Par exemple l'interface Document implémente aussi l'interface Node, l'interface Element implémente aussi l'interface Node.
- L'interface la plus importante est l'interface Node
- Des méthodes supplémentaires, non définies dans l'interface, peuvent être implémentées dans l'objet (des "méthodes propriétaires"), tout comme des objets non définis dans les interfaces peuvent être implémentés si nécessaire.
- Les noms d'objets utilisés dans une implémentation du DOM peuvent ne pas directement correspondre aux noms des interfaces (par exemple l'objet Document implémente l'interface HTMLDocument).
- Les objets peuvent implémenter plusieurs interfaces. Par exemple, si le navigateur prend en charge les CSS, l'objet Document implémentera en plus de l'interface HTMLDocument, les interfaces DocumentStyle et DocumentCSS.
- La norme ne définit pas de constructeur (par exemple nous ne pouvons pas créer un objet Text avec `var T = new Text('abcd')`). La création d'objet se fait avec des méthodes de fabrication (par exemple `var T = document.createTextNode('abcd')`).

```

graph TD
    Document --> Node
    DocumentType --> Node
    Element --> Node
    DocumentFragment --> Node
    Attr --> Node
    CharaceterData --> Node
    
```

francois.piat@univ-fcomte.fr

DOM W3

DOCUMENT Object Model

DOM 36

- **Document = hiérarchie de nœuds (node).**
- **Un nœud peut être**
 - un tag,
 - un attribut du tag,
 - le contenu textuel entouré par un tag,
 - un saut de ligne.
- **Parent – enfants : un nœud peut contenir d'autres nœuds qui peuvent contenir d'autres nœuds qui peuvent contenir d'autres nœuds qui peuvent contenir d'autres nœuds qui peuvent ...**

francois.piat@univ-fcomte.fr

DOM W3
DOCUMENT Object Model
DOM 37

```

<html>
  <head>
    <title>Exemple</title>
  </head>
  <body>
    <h1>Le DOM</h1>
    <p>Modèle hiérarchique</p>
    <p>Un sac de noeuds</p>
  </body>
</html>

```

Racine de l'arbre
 nœud de type Document et non le nœud <html> comme on pourrait s'y attendre (la raison en est la compatibilité avec XML).

francois.piat@univ-fcomte.fr

DOM W3
Node
DOM 38

- Les nœuds sont représentés par des objets de type **Node** implémentant l'interface **Node** qui définit des propriétés et des méthodes permettant de naviguer dans l'arborescence (**firstChild**, **lastChild**, **childNodes**, **parentNode**, etc) et de la modifier dynamiquement (**appendChild()**, **removeChild()**, etc).
- Comme les nœuds ne représentent pas tous la même chose (un tag, un attribut, du texte, etc) la propriété **nodeType** permet de savoir quelle interface particulière implémente le nœud. Par exemple un nœud représentant un tag HTML implémentera l'interface **Element**, et un nœud représentant un attribut HTML implémentera l'interface **Attr**.

Node.DOCUMENT

Node.ELEMENT

Node.TEXT

francois.piat@univ-fcomte.fr

DOM W3

Node

DOM 39

- Les nœuds implémentent généralement plusieurs interfaces.
- Les tags HTML ci-contre sont les seuls n'ayant que l'interface `HTMLElement`.
- Tous les autres tags HTML ont en plus de `HTMLElement` une interface spécifique (par exemple le tag de lien `a` implémente aussi `HTMLLinkElement`, les zones `input` implémentent aussi `HTMLInputElement`, etc).

abbr acronym address b
bdo big center cite code
dd dfn dt em i kbd
noframe noscript samp
small span strike strong
sub sup tt u var

#documentHTMLDocument → Document → Node

HTML

HEAD

BODYHTMLBodyElement → HTMLElement → Element → Node

TITLE

H1

P

P

#text

#text

#text

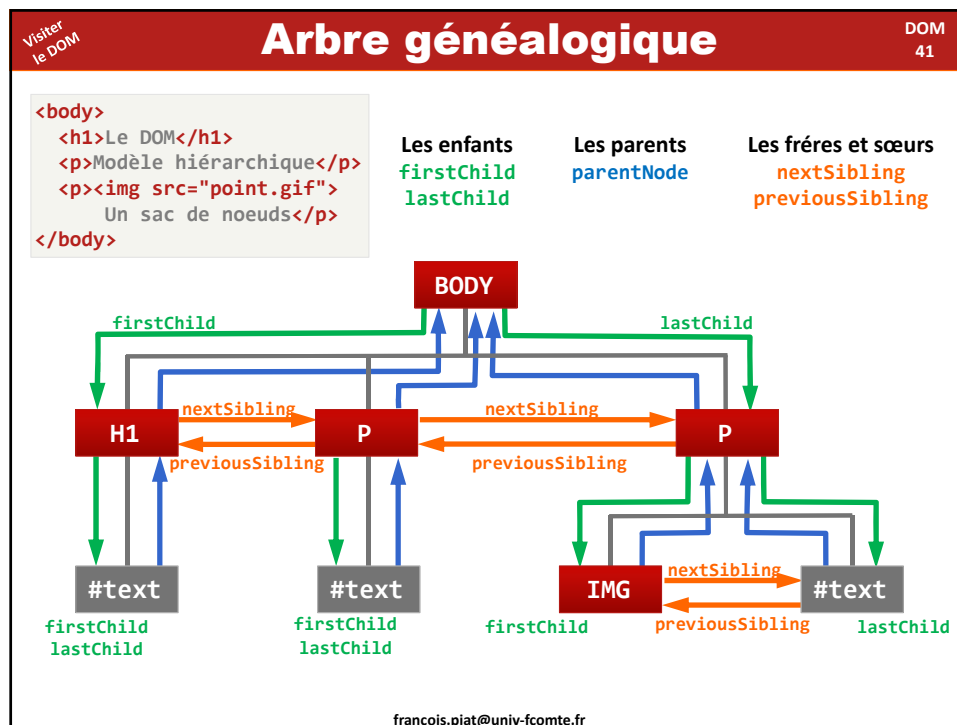
IMGHTMLImageElement → HTMLElement → Element → Node

#textNode

- Les nœuds correspondant aux attributs des éléments ne sont jamais représentés car les nœuds de type attribut ne font pas directement partie de l'arborescence. Les spécifications DOM permettent d'accéder aux nœuds de type attribut par le tableau `attributes` de l'interface `Node`.

francois.piat@univ-fcomte.fr





Visiter le DOM

visiterDOM() en action

DOM 43

```
function visiterDOM(Noeud) {
  for (var E = Noeud.firstChild; E !== null; E = E.nextSibling) {
    visiterDOM(E);
  }
}
```

```
<html>
<head>
  <title>Exemple</title>
</head>
<body>
  <h1>Le DOM</h1>
  <p>Modèle hiérarchique</p>
  <p>
    Un sac de noeuds</p>
</body>
</html>
```

francois.piat@univ-fcomte.fr

Visiter le DOM

visiterDOM() utilisation

DOM 44

```
function visiterDOM(Noeud, uneFonction) {
  uneFonction(Noeud);

  for (var E = Noeud.firstChild; E !== null; E = E.nextSibling) {
    visiterDOM(E, uneFonction);
  }
}
```

```
var Msg = '';
visiterDOM(document, function(Nd) {
  Msg += Nd.nodeName;
});
alert(Msg);
```

#document
 HTML
 HEAD
 TITLE
 #text
 BODY
 H1
 #text
 P
 #text
 P
 IMG
 #text

francois.piat@univ-fcomte.fr

Visiter le DOM

getElementById()

DOM 45

Trouver un élément HTML identifié par un attribut ID.

```

<body>
  <h1>Le DOM</h1>
  <p id="p1">Modèle hiérarchique</p>
  <p id="p2">
    Un sac de noeuds</p>
</body>
```

```
var Picto = document.getElementById('i1');
```

```
Picto.src = 'rond.gif';
```

```
document.getElementById('i1').src = 'rond.gif';
```

```
var Para = document.getElementById('p1');
```

```
Para.className = 'rouge';
```

```
document.getElementById('p1').className = 'rouge';
```

francois.piat@univ-fcomte.fr

Visiter le DOM

getElementsByTagName()

DOM 46

Pour trouver tous les éléments d'un certain type.

```

<body>
  <h1>Le DOM</h1>
  <p id="p1">Modèle hiérarchique</p>
  <p id="p2">
    Un sac de noeuds</p>
</body>
```

```
var Elems = document.getElementsByTagName('P');
```

↓

Liste de nœuds (NodeList),
à traiter comme un tableau.

Elems[0]

Elems[1]

Ne pas utiliser les listes
et méthodes prévues

```
Elems.item(0)
```

```
Elems.item[0]
```

```
Elems.namedItem('p1')
```

```
Elems.namedItem['p1']
```

```
Elems.item(1)
```

```
Elems.item[1]
```

```
Elems.namedItem('p2')
```

```
Elems.namedItem['p2']
```

francois.piat@univ-fcomte.fr

Visiter le DOM

getElementsByClassName()

DOM 47

Pour trouver tous les éléments avec une certaine classe CSS.

```
<body>
  <h1>Le DOM</h1>
  <p class="marge10">Modèle hiérarchique</p>
  <p id="p2">
  Un sac de noeuds</p>
</body>
```

var Elems = document.getElementsByClassName('marge10');

↓
Liste de nœuds (NodeList),
à traiter comme un tableau.

Ne pas utiliser les listes
et méthodes prévues

Elems.item(0)
Elems.item[0]
Elems.namedItem('p1')
Elems.namedItem['p1']

Elems.item(1)
Elems.item[1]
Elems.namedItem('p2')
Elems.namedItem['p2']

francois.piat@univ-fcomte.fr

Visiter le DOM

getElementsByClassName()

DOM 48

```
if (! document.getElementsByClassName) {

  document.getElemenByClassName = function(Noeud, Cls) {
    var ExpReg = new RegExp('^|\\s' + Cls + '\\s|$');
    var Elems = [];
    visiterDOM( Noeud,
      function(Nd) {
        if (ExpReg.test(Nd.className)) {
          Elems[Elems.length] = Nd;
        }
      }
    );
    return Elems;
  };
}
```

francois.piat@univ-fcomte.fr

Visiter le DOM

querySelector()

DOM 49

Pour trouver le 1^{er} élément avec un certain sélecteur CSS.

```

<body>
  <h1>Le DOM</h1>
  <p class="marge10">Modèle hiérarchique</p>
  <p id="p2">
    Un sac de noeuds</p>
</body>

```

```
var Elem = document.querySelector('p.marge10');
```

```
var Elem = document.querySelector('#p2 img');
```

```
var Elem = document.querySelector('p + p img');
```

francois.piat@univ-fcomte.fr

Visiter le DOM

querySelectorAll()

DOM 50

Pour trouver tous les éléments avec un certain sélecteur CSS.

```

<body>
  <h1>Le DOM</h1>
  <p class="marge10">Modèle hiérarchique</p>
  <p class="marge10">
    Un sac de noeuds</p>
</body>

```

```
var Elems = document.querySelectorAll('p.marge10');
```

↓

Liste de nœuds (NodeList),
à traiter comme un tableau.

Elems[0]

Elems[1]

francois.piat@univ-fcomte.fr

Visiter le DOM

Méthodes document et element

DOM 51

- `getElementsByTagName`, `getElementsByClassName`, `querySelector` et `querySelectorAll` sont des méthodes de l'objet **document** et des méthodes de l'objet **element**

```
<body>
  <h1>Le DOM</h1>
  <p class="marge10">Modèle hiérarchique</p>
  <p id="p2">
    Un sac de noeuds</p>
</body>
```

```
var P = document.querySelector('#p2');

var E = P.querySelector('.marge10');

var Elems = P.getElementsByTagName('img');
```

francois.piat@univ-fcomte.fr

Visiter le DOM

Attributs d'un élément

DOM 52

- Les attributs sont théoriquement des nœuds mais ils n'apparaissent pas comme enfant d'un élément.
- L'interface **Element** fournit des méthodes pour atteindre les attributs

<code>hasAttribute('NomAttrib')</code>	<code>true / false</code>
<code>getAttribute('NomAttrib')</code>	renvoie la valeur
<code>getAttributeNode('NomAttrib')</code>	renvoie un objet Node

- On peut atteindre directement les attributs

```
<p id="p2">Un sac de noeuds</p>
```

```
var Url = document.getElementById('i1').src;
var CSS = document.getElementById('p2').className;
```

francois.piat@univ-fcomte.fr



Modifier le DOM

Changer la hiérarchie

DOM 54

- 4 méthodes de l'interface **Node**.

`appendChild()` `insertBefore()` `replaceChild()` `removeChild()`
- A invoquer avec le nœud parent du nœud à traiter.

`noeudParent.appendChild(nouveauNoeud);`
`noeudParent.insertBefore(nouveauNoeud, noeudEnfant);`
`noeudParent.replaceChild(nouveauNoeud, noeudEnfant);`
`noeudParent.removeChild(noeudEnfant);`
- Toutes les méthodes renvoient une référence sur le nœud à traiter.

francois.piat@univ-fcomte.fr

Modifier le DOM

appendChild()

DOM 55

- La méthode **appendChild()** ajoute un enfant à la fin des enfants d'un nœud.
- Quand le nœud à ajouter est pris dans la hiérarchie, il est déplacé.
- Si le nœud ajouté a des descendants, ils sont ajoutés également.
- La méthode renvoie une référence sur le nœud ajouté.

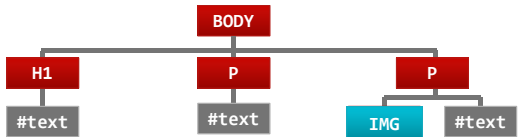
francois.piat@univ-fcomte.fr

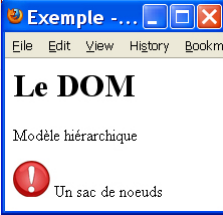
Modifier le DOM

appendChild()

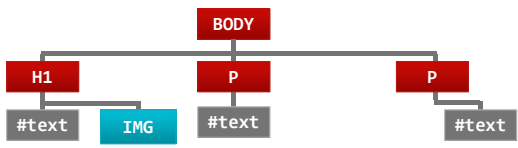
DOM 56


```
<body>
<h1>Le DOM</h1>
<p id="p1">Modèle hiérarchique</p>
<p id="p2">Un sac de noeuds</p>
</body>
```





```
var Noeud = document.getElementById('i1');
var Parent = document.getElementsByTagName('H1')[0];
Parent.appendChild(Noeud);
```





francois.piat@univ-fcomte.fr

Modifier le DOM

insertBefore()

DOM 57

- La méthode **insertBefore()** ajoute un nœud devant un autre nœud.
- Quand le nœud à insérer est pris dans la hiérarchie, il est déplacé.
- Si le nœud inséré a des descendants, ils sont ajoutés également.
- La méthode renvoie une référence sur le nœud inséré.

francois.piat@univ-fcomte.fr

Modifier le DOM

insertBefore()

DOM 58

```
<body>
  <h1>Le DOM</h1>
  <p id="p1">Modèle hiérarchique</p>
  <p id="p2">Un sac de noeuds</p>
</body>
```

Exemple

Le DOM

Modèle hiérarchique

Un sac de noeuds

```
var Noeud = document.getElementById('p2');
var Avant = document.getElementById('p1');
var Parent = document.body;
Parent.insertBefore(Noeud, Avant);
```

Exemple

Le DOM

Un sac de noeuds

Modèle hiérarchique

francois.piat@univ-fcomte.fr

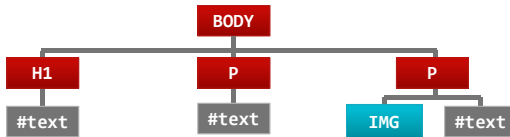
Modifier le DOM
removeChild()
DOM 59

- La méthode **removeChild()** supprime un nœud.
- Si le nœud a des descendants, ils sont supprimés également.
- La méthode renvoie une référence sur le nœud supprimé.

francois.piat@univ-fcomte.fr

Modifier le DOM
removeChild()
DOM 60

```
<body>
  <h1>Le DOM</h1>
  <p id="p1">Modèle hiérarchique</p>
  <p id="p2">Un sac de noeuds</p>
</body>
```



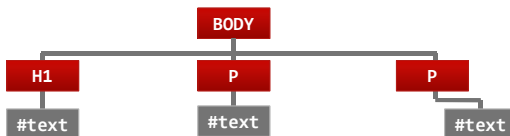
Exemple ...

Le DOM

Modèle hiérarchique

Un sac de noeuds

```
var Enfant = document.getElementById('i1');
var Parent = document.getElementById('p2');
Parent.removeChild(Enfant);
```



Exemple ...

Le DOM

Modèle hiérarchique

Un sac de noeuds

francois.piat@univ-fcomte.fr

Modifier
le DOM

removeChild()

DOM
61

Supprimer tous les enfants d'un nœud sans supprimer le nœud :

```
function cleanNoeud(Noeud){
    while (Noeud.firstChild) {
        Noeud.removeChild(Noeud.firstChild);
    }
}
```

francois.piat@univ-fcomte.fr

Modifier
le DOM

replaceChild()

DOM
62

- La méthode **replaceChild()** remplace un noeud par un autre.
- Quand le nœud à ajouter est pris dans la hiérarchie, il est déplacé.
- Si le nœud a des descendants, ils sont déplacés également.
- La méthode renvoie une référence sur le nœud remplacé.

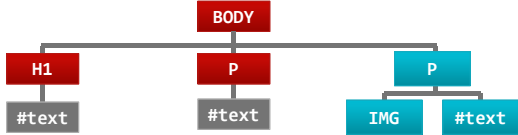
francois.piat@univ-fcomte.fr

Modifier le DOM

replaceChild()

DOM 63

```
<body>
  <h1>Le DOM</h1>
  <p id="p1">Modèle hiérarchique</p>
  <p id="p2">Un sac de noeuds</p>
</body>
```



Exemple -...

File Edit View History Bookm


Le DOM

Modèle hiérarchique

!

Un sac de noeuds

```
var Noeud = document.getElementById('p2');
var Ancien = document.getElementsByTagName('H1')[0];
var Parent = document.body;
Parent.replaceChild(Noeud, Ancien);
```



Exemple -...

File Edit View History Bookm

!

Un sac de noeuds

Modèle hiérarchique

francois.piat@univ-fcomte.fr

Modifier le DOM

Créer des éléments

DOM 64

Deux phases de traitement :

1

Créer le nouvel élément avec :

createElement()

createTextNode()

cloneNode()

2

Attacher le nouvel élément à un nœud avec :

appendChild()

insertBefore()

replaceChild()

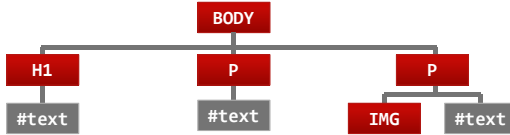
**Plus simple,
plus rapide :
innerHTML**

francois.piat@univ-fcomte.fr

32

Modifier le DOM
createElement()
DOM 65

La méthode **createElement()** permet de créer des éléments HTML à la volée. L'élément est créé sans attribut.
 Paramètre : le nom de l'élément HTML à créer.



Exemple - ...

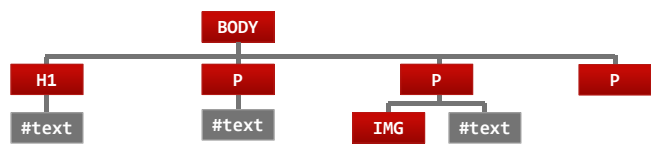
File Edit View History Bookm

Le DOM

Modèle hiérarchique

Un sac de noeuds

```
var Para = document.createElement('P');
Para.className = 'enRouge';
Para.id = 'p3';
document.body.appendChild(Para);
```



Exemple - ...

File Edit View History Bookm

Le DOM

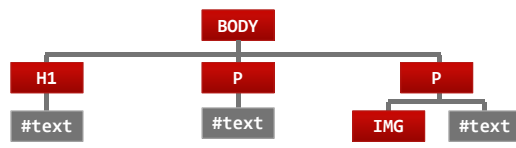
Modèle hiérarchique

Un sac de noeuds

francois.piat@univ-fcomte.fr

Modifier le DOM
createTextNode()
DOM 66

La méthode **createTextNode()** permet de créer des nœud de type texte.
 Paramètre : le texte à ajouter à l'élément.



Exemple - ...

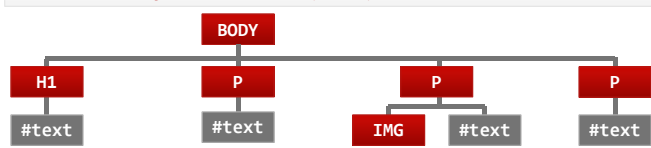
File Edit View History Bookm

Le DOM

Modèle hiérarchique

Un sac de noeuds

```
var Para = document.createElement('P');
Para.className = 'enRouge';
Para.id = 'p3';
var Txt = document.createTextNode('Du texte');
Para.appendChild(Txt);
document.body.appendChild(Para);
```



Exemple - ...

File Edit View History Bookm

Le DOM

Modèle hiérarchique

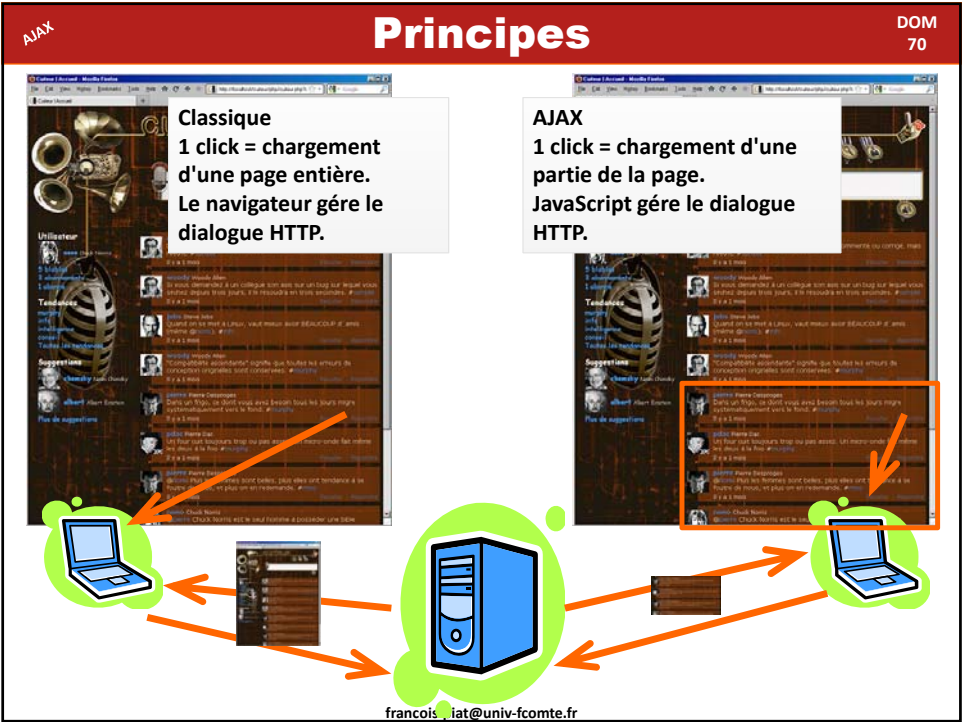
Un sac de noeuds

Du texte

francois.piat@univ-fcomte.fr

Modifier le DOM	innerHTML	DOM 67
<ul style="list-style-type: none"> • Merci Microsoft. • Propriété innerHTML : code HTML complet d'un nœud élément. • Propriété en lecture et en écriture. • A utiliser plutôt que les méthodes de création DOM : <ul style="list-style-type: none"> • beaucoup plus simple, • beaucoup plus rapide. 		
francois.piat@univ-fcomte.fr		

Modifier le DOM	innerHTML	DOM 68
Le code à ajouter	<pre><p> François Piat enseigne les langages du web. </p></pre>	
Méthodes DOM	<pre>var Para = document.createElement('P'); var Img = document.createElement('IMG'); Img.src = 'piat.gif'; Para.appendChild(Img); var Strong = document.createElement('STRONG'); var Texte1 = document.createTextNode('François Piat'); Strong.appendChild(Texte1); Para.appendChild(Strong); var Texte2 = document.createTextNode(' enseigne ... Web. '); Para.appendChild(Texte2); document.body.appendChild(Para);</pre>	
innerHTML	<pre>var Para = document.createElement('P'); Para.innerHTML = 'François Piat enseigne les langages du web.'; document.body.appendChild(Para);</pre>	
francois.piat@univ-fcomte.fr		



AJAX	A.J.A.X.	DOM 71
<ul style="list-style-type: none">● Asynchronous JavaScript And XML (18/02/2005) http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications● Asynchrone : l'envoi de la requête HTTP ne bloque pas le navigateur● JavaScript :<ul style="list-style-type: none">● instanciation de l'objet XMLHttpRequest pour les requêtes,● réponses gérées par une fonction callback,● modification du DOM de la page.● XML :<ul style="list-style-type: none">● totalement inutilisé,● remplacé par JSON (JavaScript Object Notation).		
francois.piat@univ-fcomte.fr		

AJAX	L'objet XMLHttpRequest	DOM 72
<ul style="list-style-type: none">● Merci Microsoft :<ul style="list-style-type: none">● 09/1998 - IE5 - objet activeX : XMLHttpRequest● XMLHttpRequest : Firefox 2002 / Safari 2004 / Opera 2005● XMLHttpRequest (XHR) permet de gérer le dialogue client/serveur directement avec JavaScript<ul style="list-style-type: none">● http://www.w3.org/TR/XMLHttpRequest/● Traitement :<ol style="list-style-type: none">1. Créer un objet XMLHttpRequest var XHR = new XMLHttpRequest();2. Initialiser diverses propriétés de l'objet (fonction callback)3. Envoyer une requête (GET ou POST)4. Traiter la réponse fournie par le serveur		
francois.piat@univ-fcomte.fr		

AJAX
Méthode open()
DOM 73

- Ouvre la connexion avec le serveur.

open(méthodeHttp, url, type)

'GET'
'POST'

↑
adresse de la page sur le serveur

↑
true (asynchrone)
false (synchrone)

```
var XHR = new XMLHttpRequest();
XHR.open('GET', 'page.php?id=123&nom=piat', true);
```

```
var XHR = new XMLHttpRequest();
XHR.open('POST', 'autre_page.php', true);
```

francois.piat@univ-fcomte.fr

AJAX
Méthode send()
DOM 74

- Envoie la requête au serveur.

send(contenu)

si méthode HTTP GET : null

↑
si méthode HTTP POST
de la forme : var1=val1&var2=val2

```
var XHR = new XMLHttpRequest();
XHR.open('GET', 'page.php?id=1234&nom=piat', true);
XHR.send(null);
```

```
var XHR = new XMLHttpRequest();
XHR.open('POST', 'autre_page.php', true);
XHR.send('id=1234&nom=piat');
```

francois.piat@univ-fcomte.fr

AJAX	Recevoir les données	DOM 75
<ul style="list-style-type: none"> ● La propriété readyState permet de savoir où en est la requête envoyée au serveur : <ul style="list-style-type: none"> ● 0 : la méthode open() n'a pas été appelée ● 1 : open() a été appelée, mais pas send() ● 2 : send() a été appelée ● 3 : des données sont en transfert ● 4 : les données sont totalement transférées. ● Le gestionnaire d'événement onreadystatechange permet de gérer les traitements en fonction de l'état de la requête. ● Les propriétés status, statusText, responseText et responseXML permettent de gérer les données reçues. 		
francois.piat@univ-fcomte.fr		

AJAX	Recevoir les données	DOM 76
<pre> var XHR = new XMLHttpRequest(); XHR.open('POST', 'autre_page.php', true); XHR.send('id=1234&nom=piat'); XHR.onreadystatechange = function() { if (XHR.readyState != 4) { // Requête en cours return; } if (XHR.status != 200) { // Erreur alert('Erreur ' + XHR.status + ' : ' + XHR.statusText); return; } // On fait le traitement des données reçues // qui se trouvent dans XHR.responseText. // Exemple : affichage des données dans le bloc Retour document.getElementById('Retour').innerHTML = XHR.responseText; } </pre>		
francois.piat@univ-fcomte.fr		