
D.O.M.

Modèle Objet de Document

DOM, cela veut dire :

- **Document** : C'est tout simplement la page **HTML** vue par l'utilisateur. Ne pas confondre avec le fichier **HTML** qui a été chargé.
 - **Objet** : C'est le concept informatique d'objet. Il est nommé ainsi par analogie avec les objets du monde réel.
 - **Modèle** : Représentation reproduisant les caractéristiques et comportements de façon appréhendable par l'ordinateur.
-

Le DOM

- C'est un standard fixé par le W3C,
 - Le Modèle Objet de Document est une interface de programmation indépendante de la plateforme et du langage. Elle permet aux programmes et scripts d'accéder dynamiquement et de mettre à jour le contenu, la structure et les styles d'un document.
 - Il a été développé pour permettre la portabilité de scripts JavaScript et de programmes Java d'un navigateur Web à l'autre.
-

Ce que représente le DOM :

un document HTML

```
<html>
<head>
  <title>Test</title>
</head>
<body>
  <h3>Un titre</h3>
  <p id="para">Un texte
    <b>en gras</b>
  </p>
</body>
</html>
```

Ce qu'affiche
le navigateur

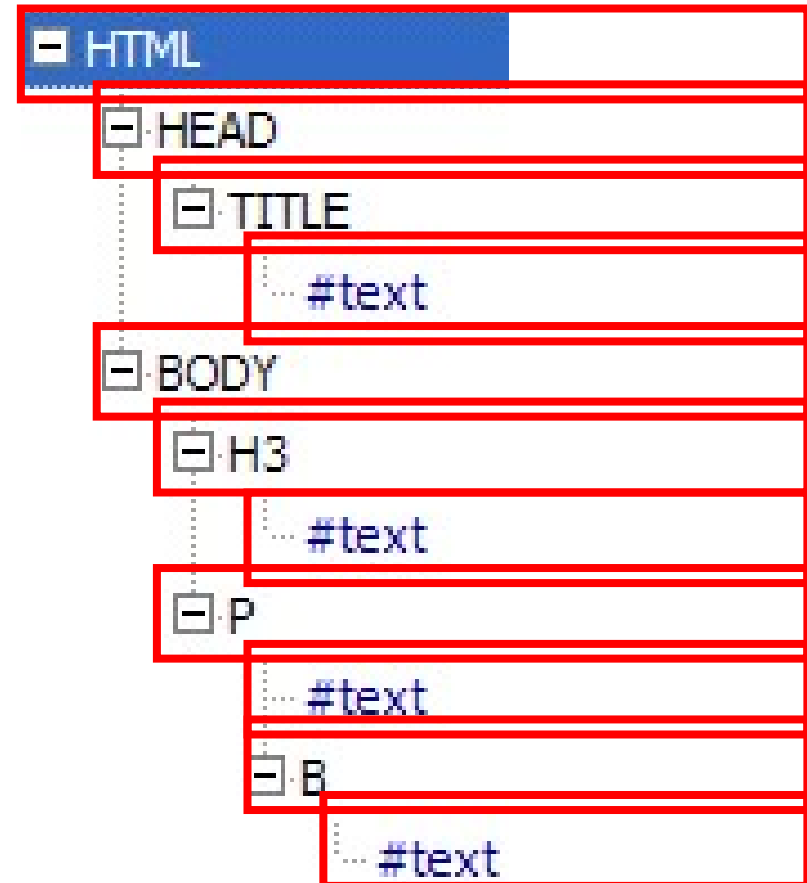
Un titre

Un texte **en gras**

Ce que représente le DOM : une arborescence

```
<html>
<head>
  <title>Test</title>
</head>
<body>
  <h3>Un titre</h3>
  <p id="para">Un texte
    <b>en gras</b>
  </p>
</body>
</html>
```

Ce que comprend le navigateur



Ce que représente le DOM :

explication du modèle

- Le langage **HTML** est constitué de balises.
 - Une balise ouvrante et une balise fermante forment un **élément HTML**.
 - Un **élément HTML** peut contenir d'autre **éléments**, on les appelle éléments enfants.
 - Il existe un **élément** particulier l'**élément #text**
Il ne peut pas avoir d'**éléments** enfants.
Il contient simplement du texte littéral.
 - Le **DOM** modélise tout cela par une arborescence reflétant l'imbrication et les relations parent/enfants
-

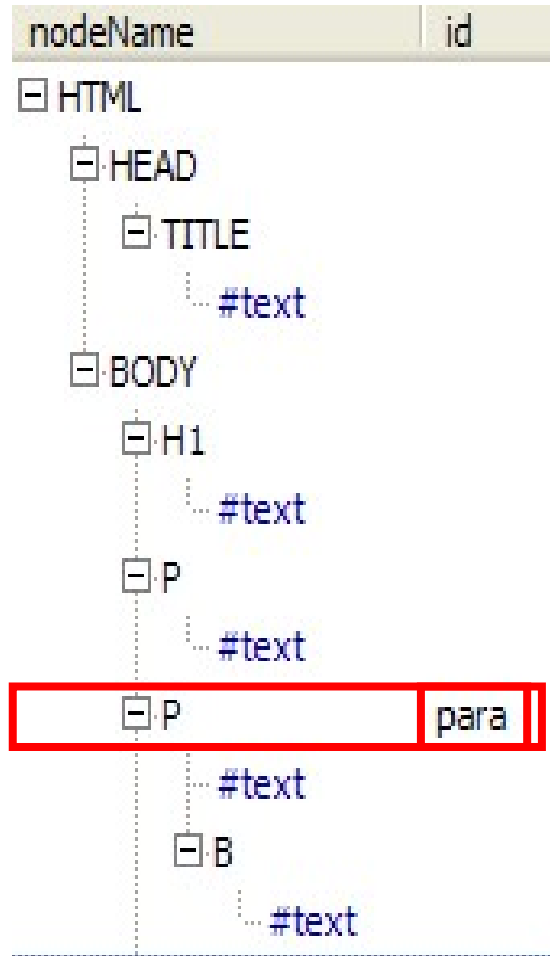
Comment accéder au DOM :

La racine de l'arborescence

- C'est l'**Objet document** qui est le point d'entrée du **DOM**
 - Il possède une **propriété documentElement** qui donne accès à l'**élément** racine d'un document **HTML**, l'**élément HTML**.
 - Pour des raisons de compatibilité il est préférable d'utiliser **document.documentElement** pour accéder à l'**élément HTML**.
 - `...`
`...`
-

Comment accéder au DOM :

Un ou des éléments particuliers



L'**Objet document** possède des méthodes pour aider à retrouver des **éléments HTML**

`<p>texte</p>`

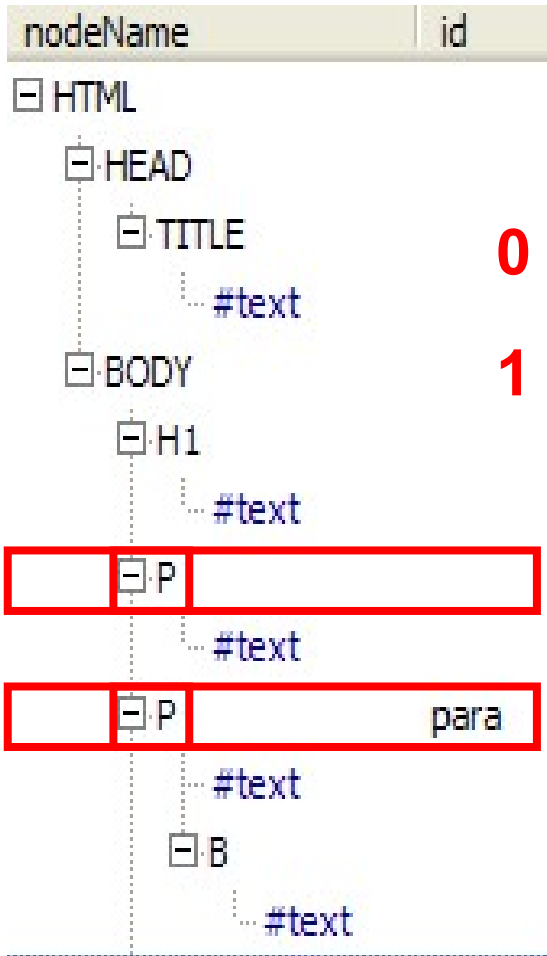
`<p id="para">Un texte
en gras</p>`

Sélectionner l'**élément** ayant un **id** particulier

`document.getElementById('para')`

Comment accéder au DOM :

Un ou des éléments particuliers



L'**Objet document** possède des méthodes pour aider à retrouver des **éléments HTML**

0

`<p>texte</p>`

1

`<p id="para">Un texte
en gras</p>`

0

`document.getElementsByTagName('p')`

Sélectionner tous les **éléments** ayant un nom de balise données, retourne un tableau:

1

`document.getElementsByTagName('p')[0]`

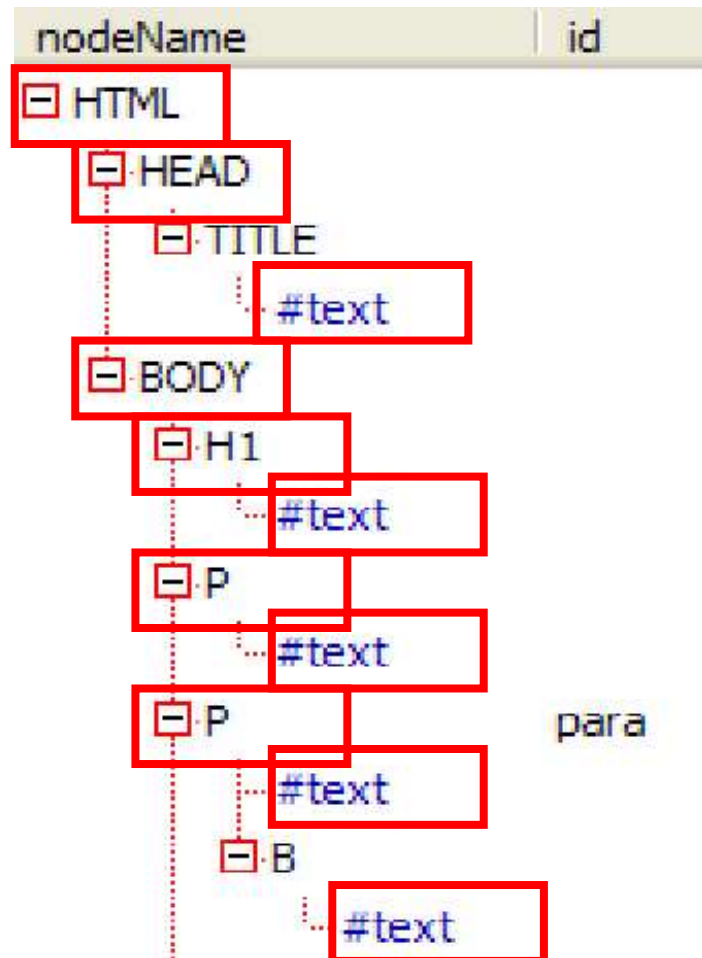
`document.getElementsByTagName('p')`

Ici on appelle le 1^{er} :

`document.getElementsByTagName('p')[0]`

Le modèle Objet du DOM :

Structure et éléments



Le **DOM** modélise la structure des **éléments** du document **HTML** sous la forme d'une arborescence d'**Objets**.

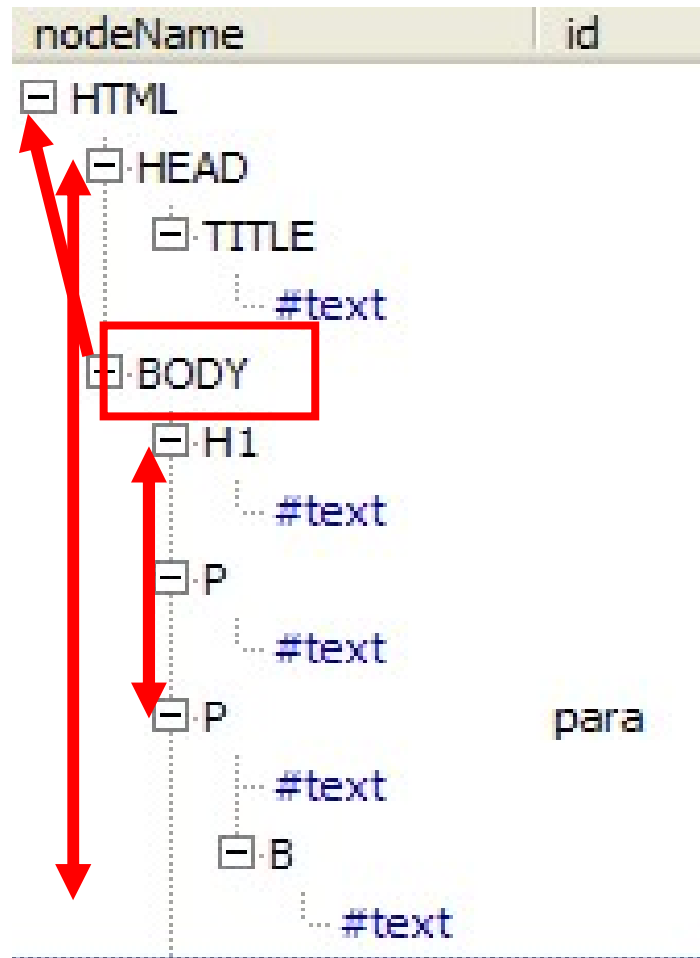
Les **Objets** du modèle ont des **propriétés** et **méthodes** :

- Communes pour la structure
- Distinctes suivant le type d'**éléments**

HTML HEAD BODY H1 P #text

Le modèle Objet du DOM :

Le concept de la structure arborescente



Le **DOM** modélise l'arborescence par le biais de relations **parent/enfants**.

On considère l'élément **BODY**

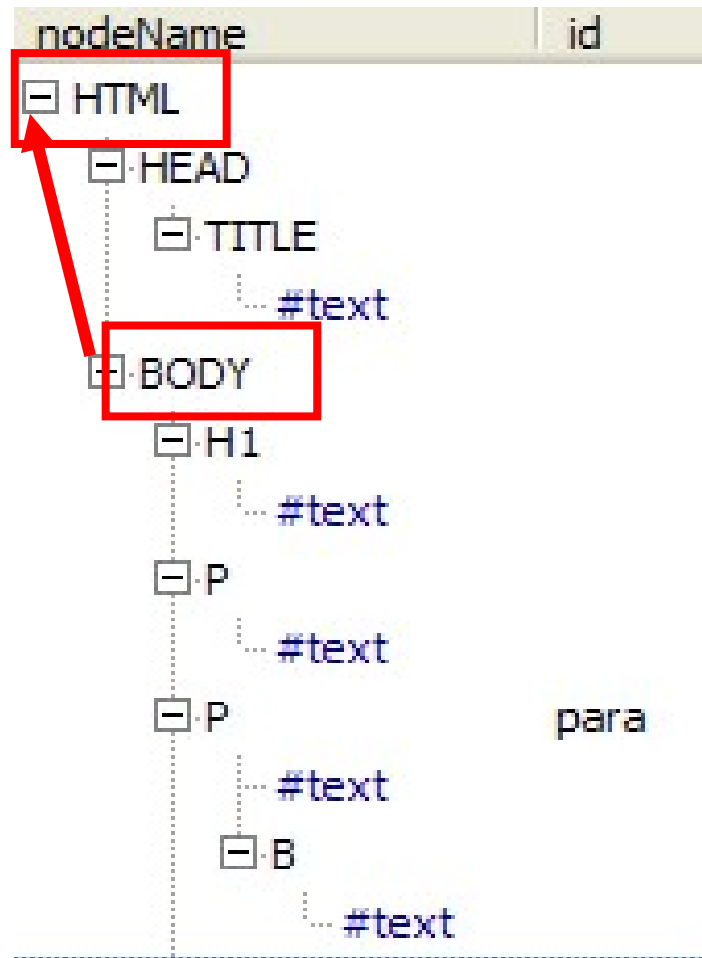
Parent désigne l'élément dans lequel il est inclus

Frère désigne les éléments ayant même **parent**

Enfant désigne les éléments dont il est le **parent**

Le concept de la structure arborescente

Le parent d'un noeud



Le **DOM** modélise l'arborescence par le biais de relations parent/enfants.

On considère l'élément **BODY**

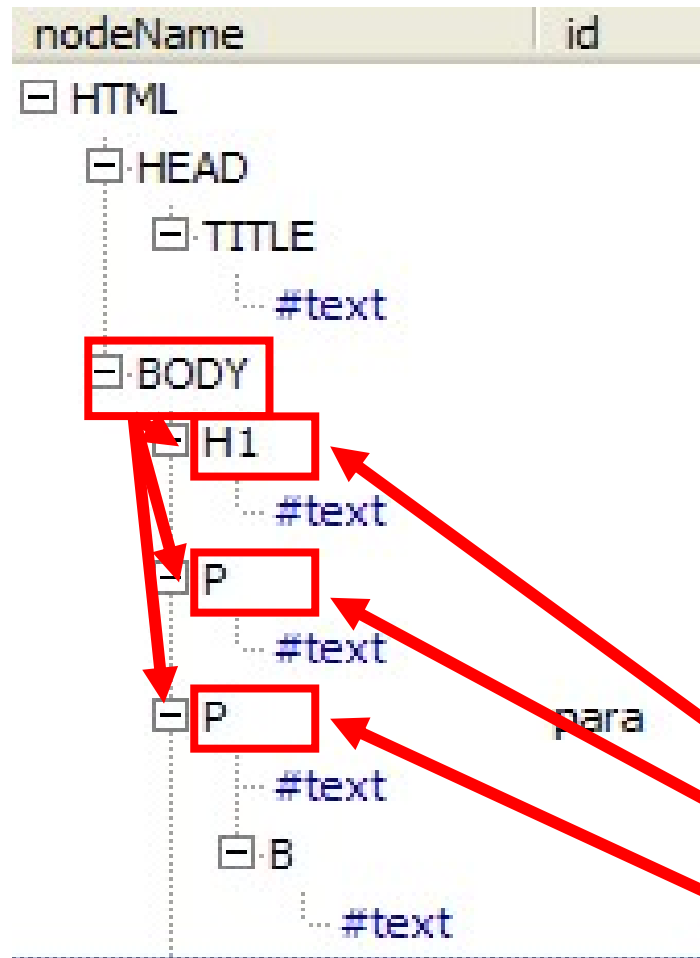
BODY a pour parent **HTML**

En notation **Objet** :

Mon_BODY.parentNode == Mon_HTML

Le concept de la structure arborescente

Les enfants d'un noeud



Le **DOM** modélise l'arborescence par le biais de relations parent/enfants.

On considère l'élément **BODY**

BODY a pour enfants :

H1 **P** **P**

En notation **Objet** :

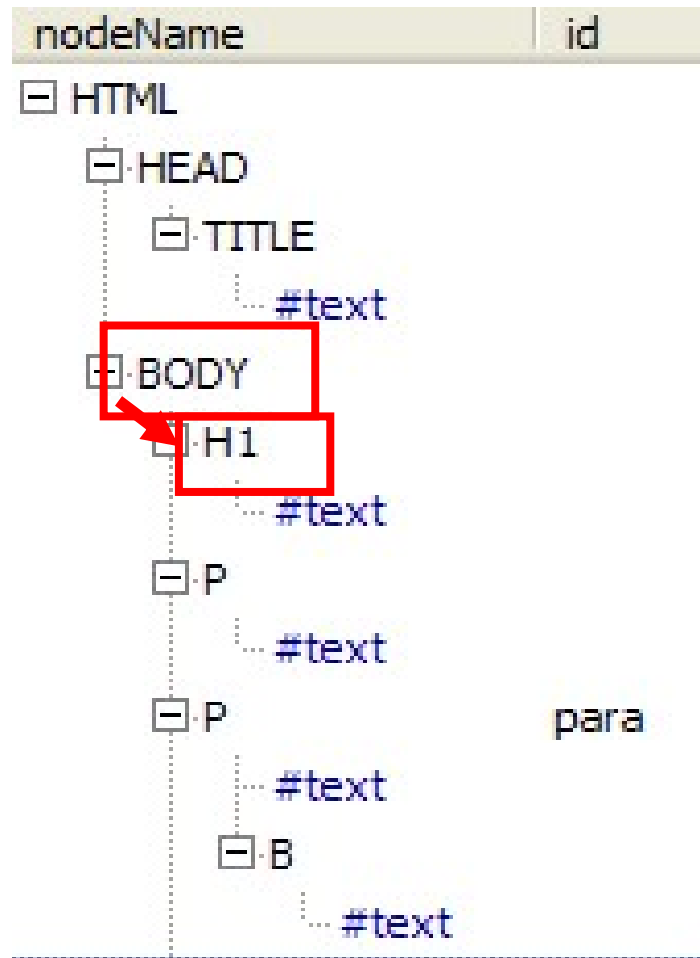
Mon_BODY.childNodes[0]

Mon_BODY.childNodes[1]

Mon_BODY.childNodes[2]

Le concept de la structure arborescente

Le premier enfant d'un noeud



Le **DOM** modélise l'arborescence par le biais de relations parent/enfants.

On considère l'élément **BODY**

BODY a pour premier enfant

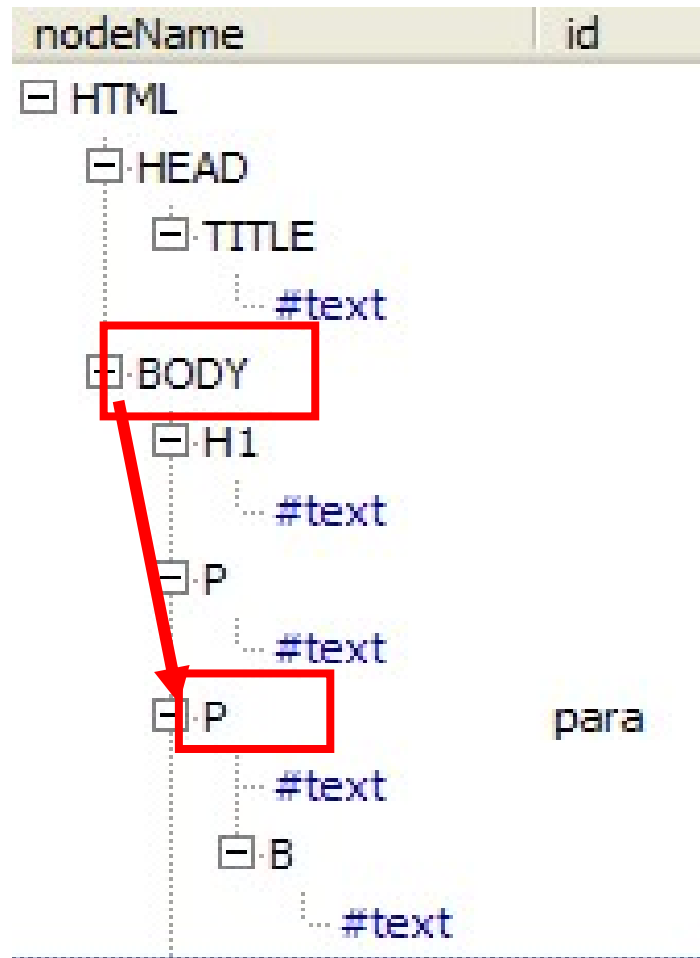
H1

En notation **Objet** :

Mon_BODY.firstChild == Mon_H1

Le concept de la structure arborescente

Le dernier enfant d'un noeud



Le **DOM** modélise l'arborescence par le biais de relations parent/enfants.

On considère l'élément **BODY**

BODY a pour dernier enfant

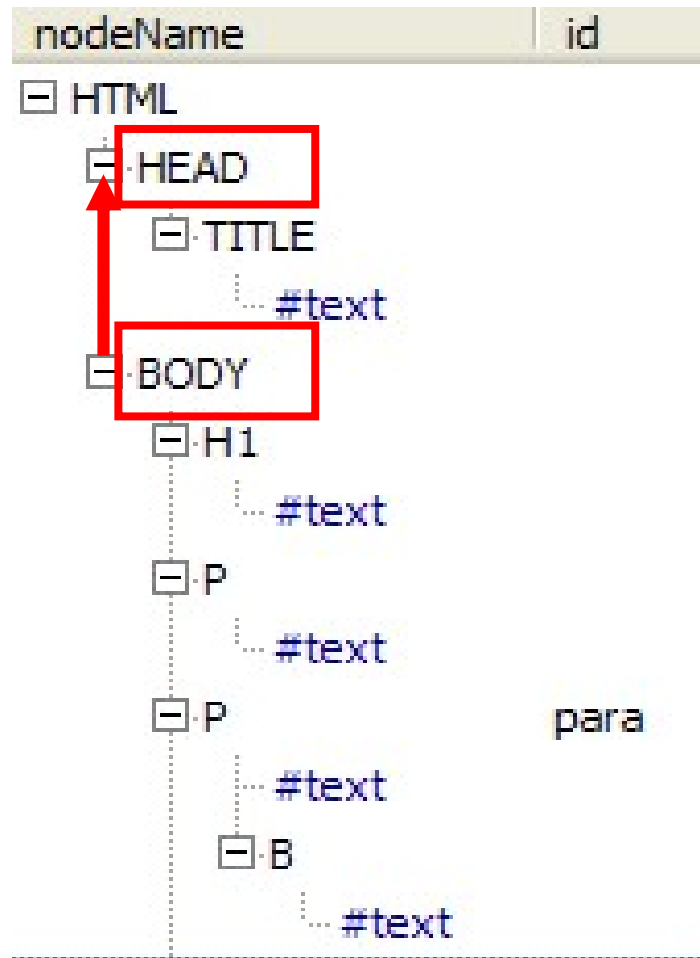
P

En notation **Objet** :

Mon_BODY.lastChild == Second_P

Le concept de la structure arborescente

Le frère « aîné » d'un noeud



Le **DOM** modélise l'arborescence par le biais de relations parent/enfants.

On considère l'élément **BODY**

BODY a pour frère « aîné »

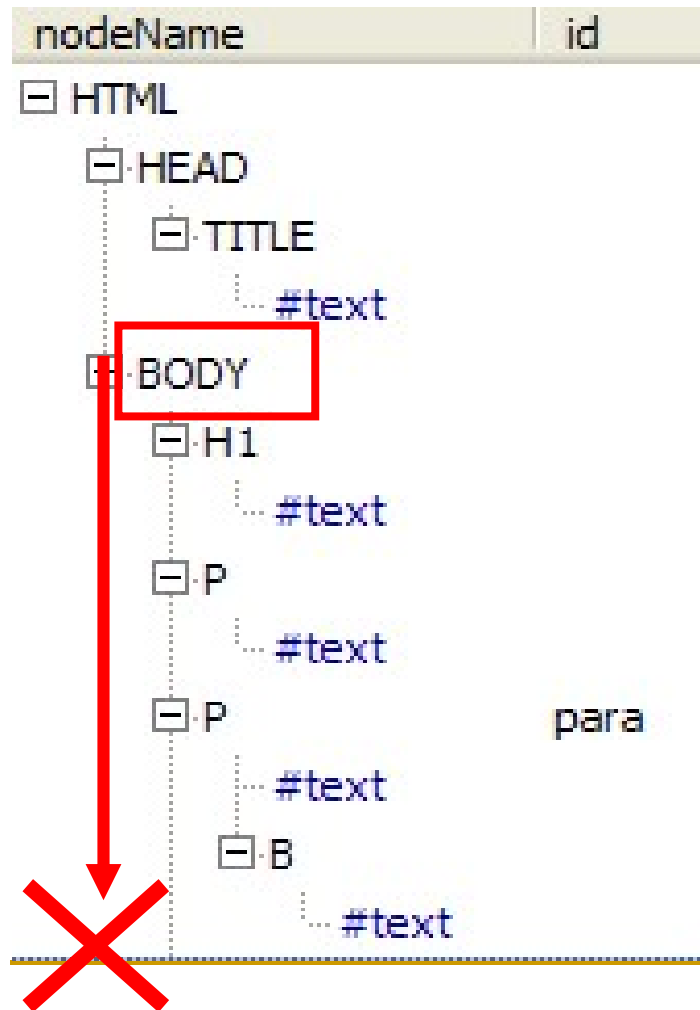
HEAD

En notation **Objet** :

Mon_BODY.previousSibling
== Mon_HEAD

Le concept de la structure arborescente

Le frère « cadet » d'un noeud



Le **DOM** modélise l'arborescence par le biais de relations parent/enfants.

On considère l'élément **BODY**

BODY n'a pas de frère «cadet»

En notation **Objet** :

Mon_BODY.nextSibling == undefined

Le modèle Objet du DOM :

Parcourir la structure arborescente ?

- Les modélisations des relations parent/enfants dans les **Objets** du **DOM** permettent de « parcourir » le document.
 - On peut ainsi « atteindre » n'importe quel objet figurant un **élément HTML**.
 - Malheureusement tous les navigateurs ne donnent pas la même interprétation d'un document **HTML**. Ils interprètent différemment les éléments « vides ».
 - On préférera l'usage de :
`document.getElementById('un_id');`
 - Mais comprendre le modèle du DOM est indispensable, si on souhaite pouvoir le manipuler.
-

Le modèle Objet du DOM :

Les éléments « vides »

Deux documents HTML

```
<html>
<head>
<title>Test</title>
</head>
<body>
<h3>Un titre</h3>
<p id="para">Un texte
    <b>en gras</b>
</p>
</body>
</html>
```

```
<html>
<head>
<title>Test</title>
</head>
<body><h3>Un titre
</h3><p id="para">Un texte
<b>en gras</b></p></body></html>
```

Un titre

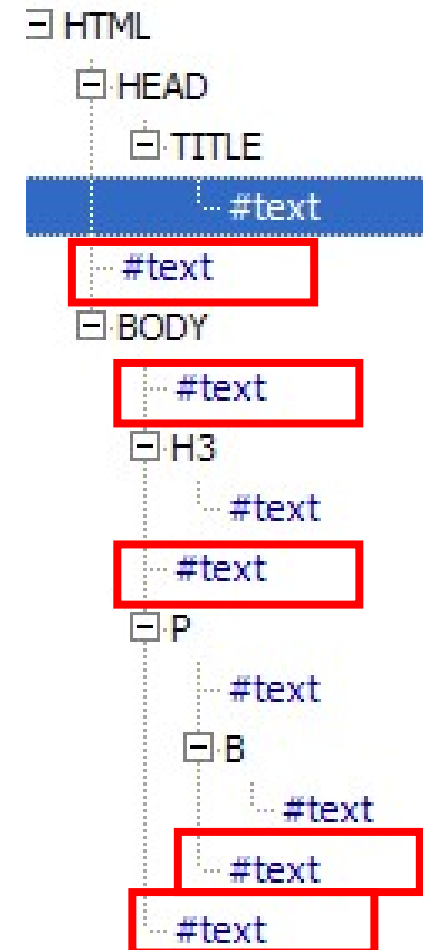
Un texte en gras

Un même affichage

Le modèle Objet du DOM :

Les éléments « vides »

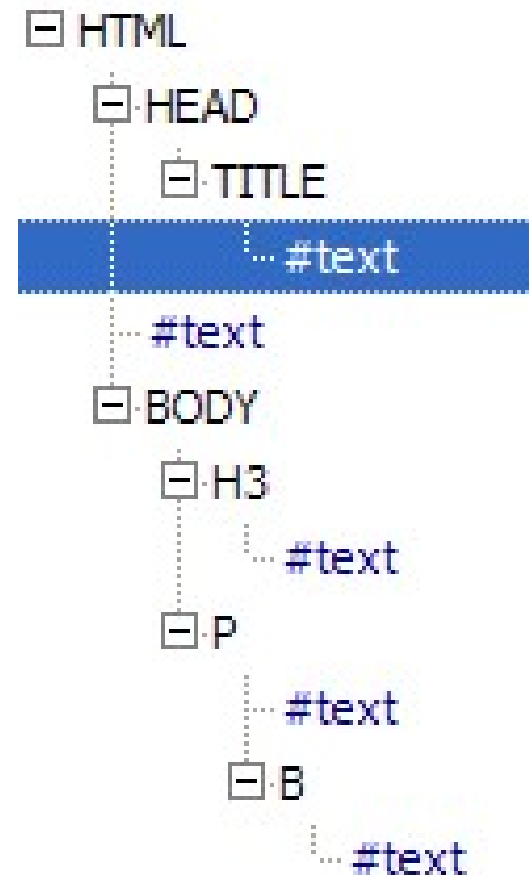
- `<html>`
`<head>`
`<title>Test</title>`
`</head>`
`<body>`
`<h3>Un titre</h3>`
`<p id="para">Un texte`
 `en gras`
`</p>`
`</body>`
`</html>`
- Beaucoup de nœuds texte sont le résultat d'espaces entre les balises.



Le modèle Objet du DOM :

Les éléments « vides »

- `<html>`
`<head>`
`<title>Test</title>`
`</head>`
`<body><h3>Un titre`
`</h3><p id="para">Un texte`
`en`
`gras</p></body></html>`
- Il est possible d'éviter les éléments vides, mais cela ne facilite pas la lecture du code **HTML**.
- **XML** a été conçu pour parer à de telles ambiguïtés.



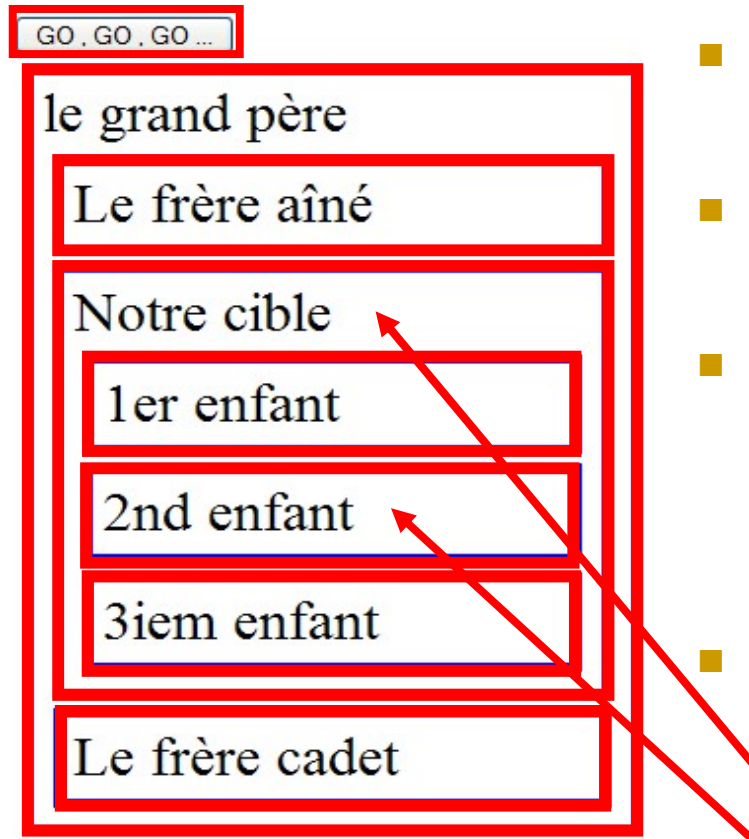
Le modèle Objet du DOM :

Modifier la structure arborescente

- Le concept parent/enfants est également employé pour permettre de modifier le document.
 - C'est ce qui rend le **DOM** si intéressant, c'est la possibilité de changer intégralement la structure du document.
 - On réalise cela par le biais d'une interface de programmation simple et adaptée, sans réaliser un programme qui « écrive » du **HTML**
 - Il est inutile de « recharger » la page pour visualiser les changements, ils sont immédiatement répercutés sur l'affichage.
-

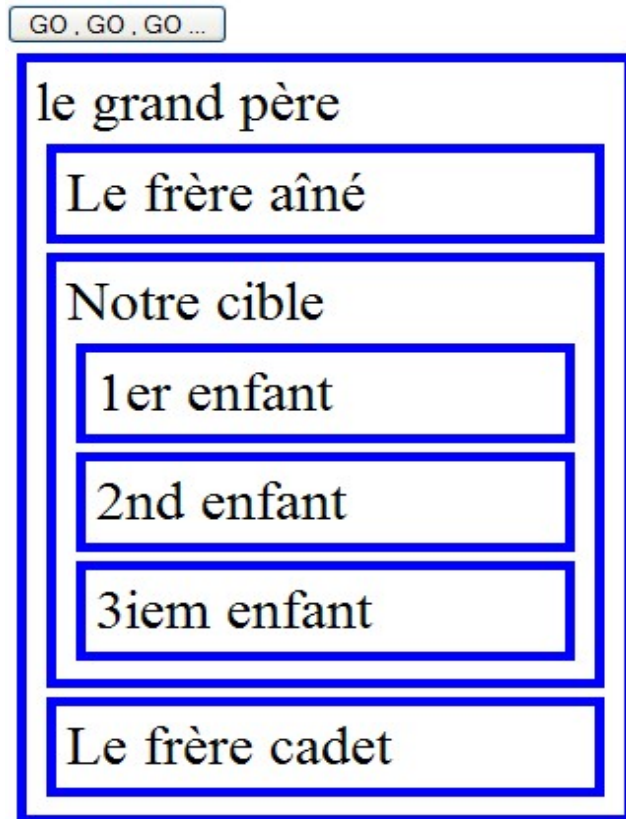
Modifier la structure arborescente

Le support d'exemples



- On utilisera cette page **HTML** comme support des exemples
- Elle comporte un bouton pour lancer le code
- Elle est constituée de divisions imbriquées figurant la structure hiérarchique des **éléments**, on leur a appliqué un **style**.
- Des **id** ont été placés pour :
 - ❑ Notre cible : **id**="notre_cible"
 - ❑ 2nd enfant : **id**="un_enfant"

Le support d'exemples : La feuille de style



- Le but est simplement de visualiser aisément la structure du document
- **div {**
border: 5px solid blue;
padding: 5px;
margin : 5px;
font-size: 30px;}**}**

Le support d'exemples :

Le code HTML

le grand père

Le frère aîné

Notre cible

1er enfant

2nd enfant

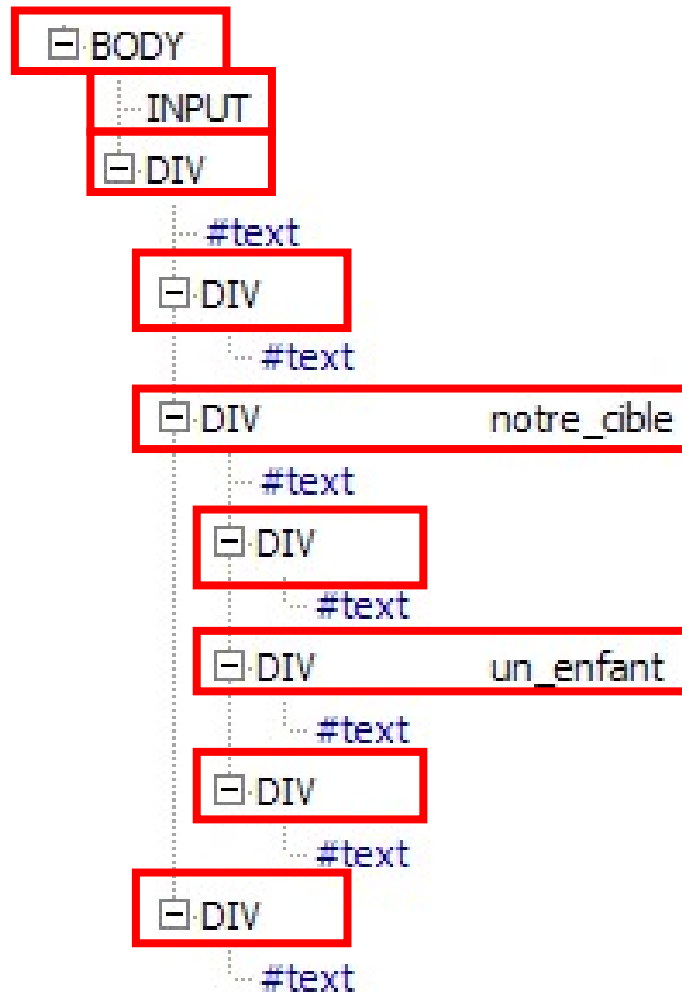
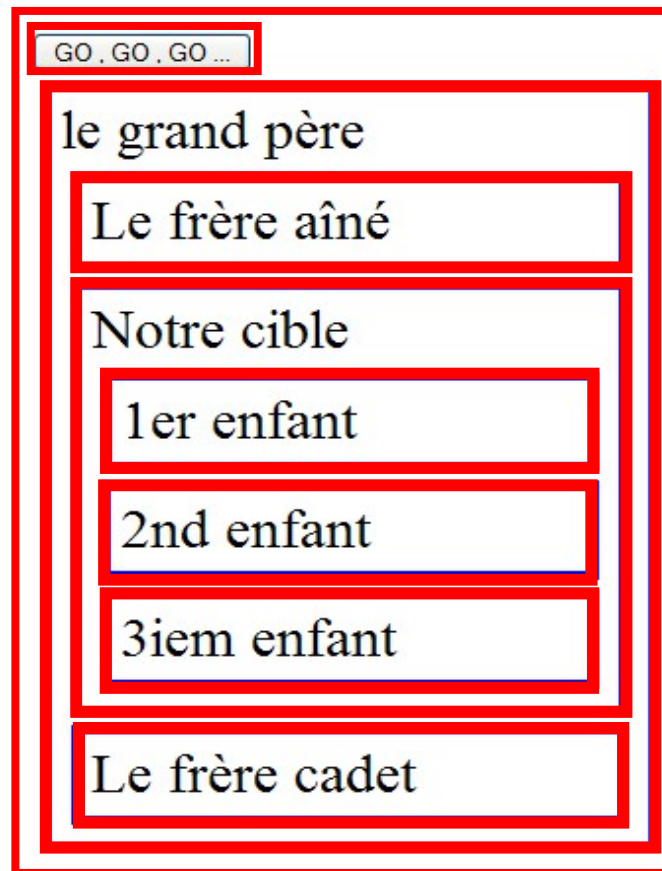
3iem enfant

Le frère cadet

```
<body>  
<input type="button"  
      value="GO , GO , GO ..."  
      onclick="ma_fct()">  
<div>  
  le grand père  
  <div>Le frère aîné</div>  
  <div id="notre_cible">  
    Notre cible  
    <div>1er enfant</div>  
    <div id="un_enfant">  
      2nd enfant</div>  
    <div>3iem enfant</div>  
  </div>  
  <div>Le frère cadet</div>  
</div>  
</body>
```

Le support d'exemples :

La représentation du DOM



Le support d'exemples :

Le code JavaScript

- Le code JavaScript est appelé par le bouton, il comporte les étapes suivantes :
 - ❑ Placer dans des variables des références à des **Objets** figurant des éléments de la page en se basant sur leurs **id** : "notre_cible" "un_enfant".
 - ❑ Construire de nouveaux éléments et garder dans une variable une référence à leurs **Objets**.
 - ❑ Assembler ces **Objets** entre eux.
 - ❑ Insérer dans le **DOM** les **Objets** ainsi construits.
- Il ne restera qu'à visualiser le résultat.

Le code JavaScript

L'appel du code

- On déclare une fonction dans les entêtes :

```
<script language="JavaScript">
```

```
<!--
```

```
function ma_fct() {
```

```
    //On place le code ici ...
```

```
}
```

```
// -->
```

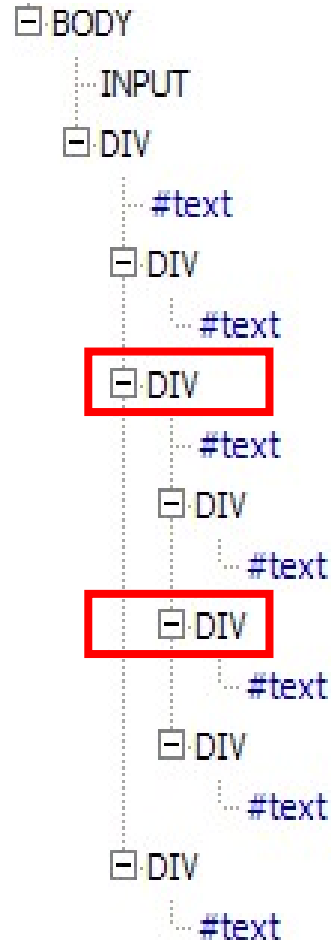
```
</script>
```

- On appelle cette fonction lorsque l'on clique sur le bouton :

```
<input type="button" value="GO , GO , GO ..."  
      onclick="ma_fct()">
```

Le code JavaScript

Obtenir des références sur des éléments



- On retrouve les **éléments** par leurs **id** et l'on place une référence à l'**Objet** dans une variable.
- ```
var Notre_Cible =
 document.getElementById(
 'notre_cible');
```
- ```
var Un_Enfant =  
    document.getElementById(  
        'un_enfant');
```

Le code JavaScript

Construire de nouveaux éléments

- Il existe deux **méthodes** de l'**Objet** document suivant le type **d'éléments** que l'on veut créer
- Pour un **élément** correspondant à une **Balise**
var Mon_Visiteur =
document.createElement('div');

La représentation
correspondant au
DOM



Le code **HTML**
qui correspondrait

`<div> </div>`

L'affichage qu'il
donnerait



Le code JavaScript

Construire de nouveaux éléments

- Il existe deux **méthodes** de l'**Objet** document suivant le type **d'éléments** que l'on veut créer
- Pour un **élément** recevant du **texte**
var Texte_Mon_Visiteur =
document.createTextNode('mon visiteur');

La représentation
correspondant au
DOM

#text

Le code **HTML**
qui correspondrait

Mon visiteur

L'affichage qu'il
donnerait

mon visiteur

Le code JavaScript

Assembler les Objets entre eux

- Assembler les **Objets** c'est définir leurs relations parent/enfants
- Pour définir que **Texte_Mon_Visiteur** est un enfant de **Mon_Visiteur** :

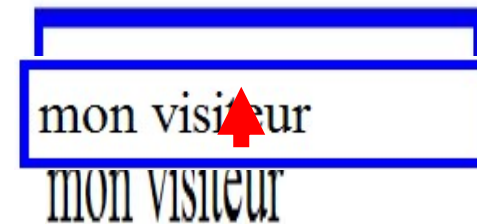
La représentation
correspondant au
DOM



Le code **HTML**
qui correspondrait

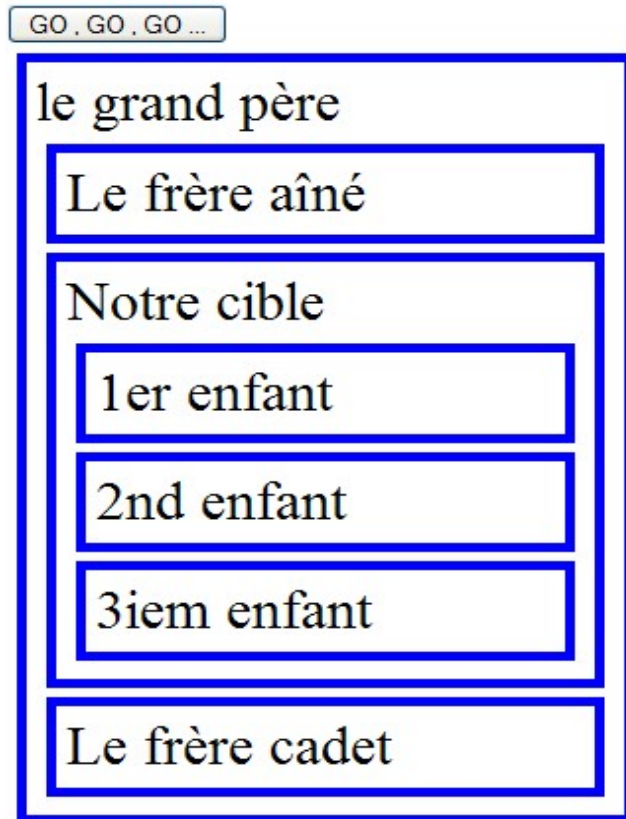
```
<div>
  Mon visiteur
</div>
```

L'affichage qu'il
donnerait



Le code JavaScript

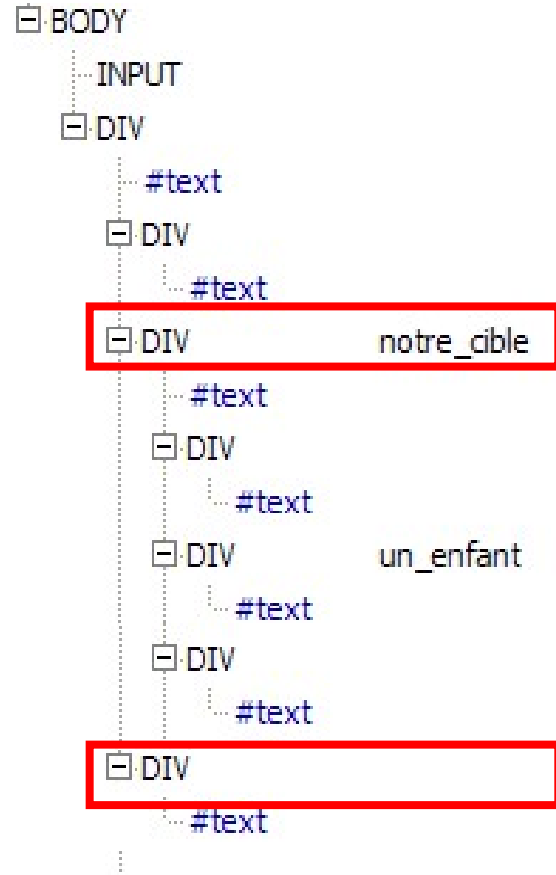
Insérer dans le DOM de la page

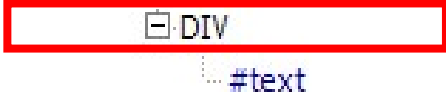


- Les éléments que l'on vient de construire n'apparaissent pas dans la page.
- Il faut les insérer de la même façon qu'on les a assemblés
- On va placer **Mon_Visiteur** comme un enfant de **Notre_Cible** .
- On utilisera exactement la même méthode.

Le code JavaScript

Insérer un élément dans le DOM de la page



- La variable `Notre_Cible` référence l'élément qui sera le parent.
- On va lui ajouter comme enfant. l'élément référencé par `Mon_Visiteur`

- `Notre_Cible.appendChild(
Mon_Visiteur);`

Le code JavaScript : Le résultat

GO, GO, GO ...

le grand père

Le frère aîné

Notre cible

1er enfant

2nd enfant

3iem enfant

mon visiteur

Le frère cadet

- Il suffit d'un appui sur le bouton pour lancer le code et visualiser le résultat
- **L'affichage est immédiat**
- **Chercher l'intrus**
- Je ne vous montre pas le code **HTML** correspondant a cette affichage car il n'a jamais été écrit, ni par vous ni par le navigateur. On a directement manipulé la structure de la page.

Le code JavaScript : tout le code

```
<html>
<head><title>Test</title></head>
<style type="text/css">
div { border: 5px solid blue;
      padding: 5px;
      margin : 5px;
      font-size: 30px;}
</style>
<script language="JavaScript">

function ma_fct() {
var Notre_Cible=
  document.getElementById('notre_cible');
var Un_Enfant=
  document.getElementById('un_enfant');
var Mon_Visiteur=
  document.createElement('div');
var Texte_Mon_Visiteur=
  document.createTextNode('mon visiteur');
Mon_Visiteur.appendChild(
  Texte_Mon_Visiteur);
Notre_Cible.appendChild(Mon_Visiteur);
}
// </script>
```

```
<body>
<input type="button"
       value="GO , GO , GO ..."
       onclick="ma_fct()">
<div>
  le grand père
  <div>Le frère aîné</div>
  <div id="notre_cible">
    Notre cible
    <div> 1er
enfant</div>
      <div
id="un_enfant">
        2nd
enfant</div>
      <div>3iem
enfant</div>
    </div>
    <div>Le frère
cadet</div>
  </div>
</body>
</html>
```

Le support d'exemples :

Variation sur un thème

GO, GO, GO ...

Notre_Cible.appendChild(Mon_Visiteur);

RAZ

le grand père

Le frère aîné

Notre cible

1er enfant

2nd enfant

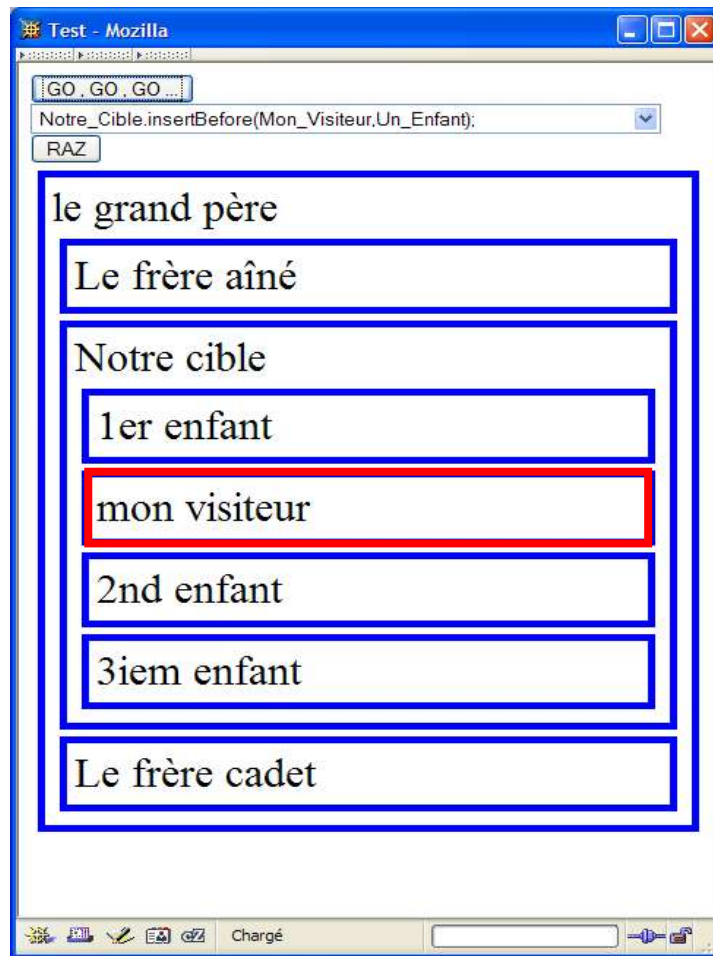
3iem enfant

Le frère cadet

- Le modèle **DOM** offre d'autres possibilités pour modifier le document

Le support d'exemples :

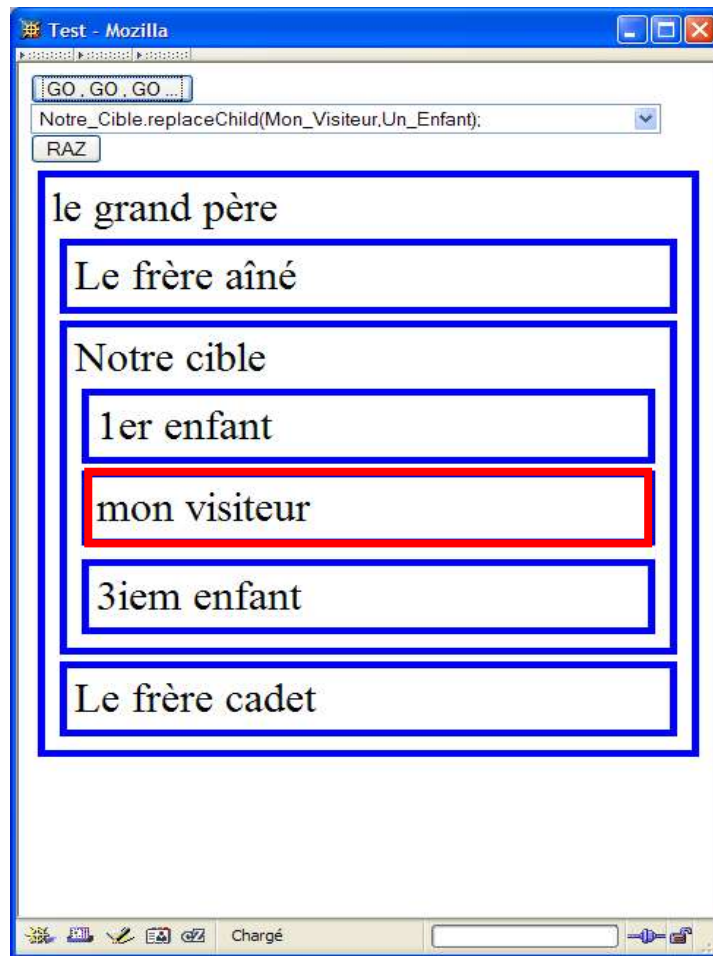
Variation sur un thème



- Placer `Mon_Visiteur` avant `Un_Enfant`
- `Notre_Cible.insertBefore(Mon_Visiteur, Un_Enfant);`

Le support d'exemples :

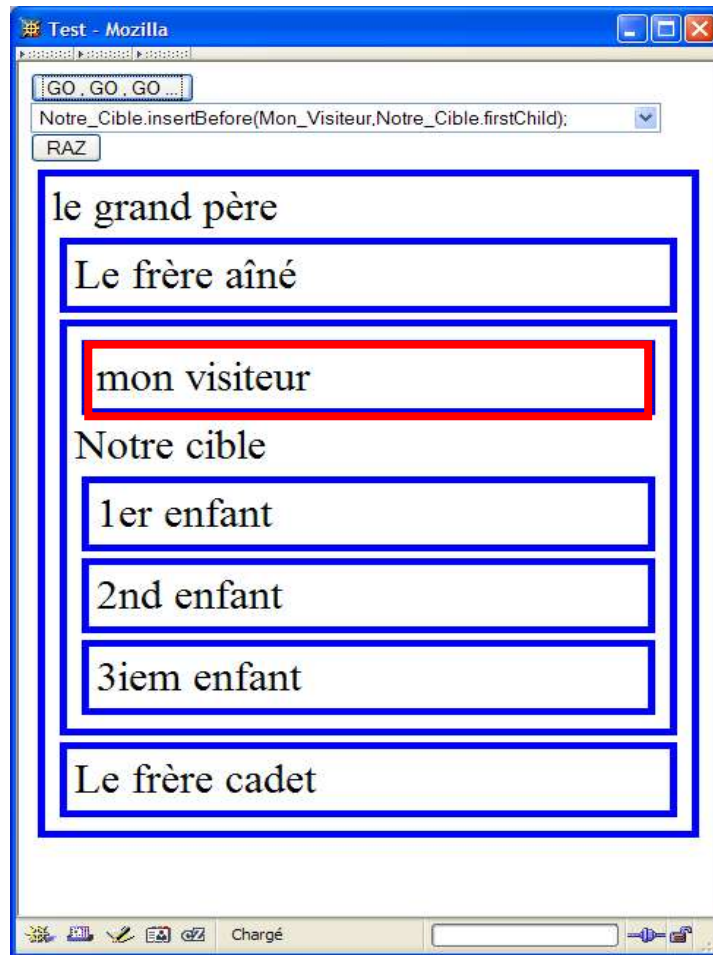
Variation sur un thème



- Remplace `Un_Enfant` par `Mon_Visiteur`
- `Notre_Cible.replaceChild(Mon_Visiteur,Un_Enfant);`

Le support d'exemples :

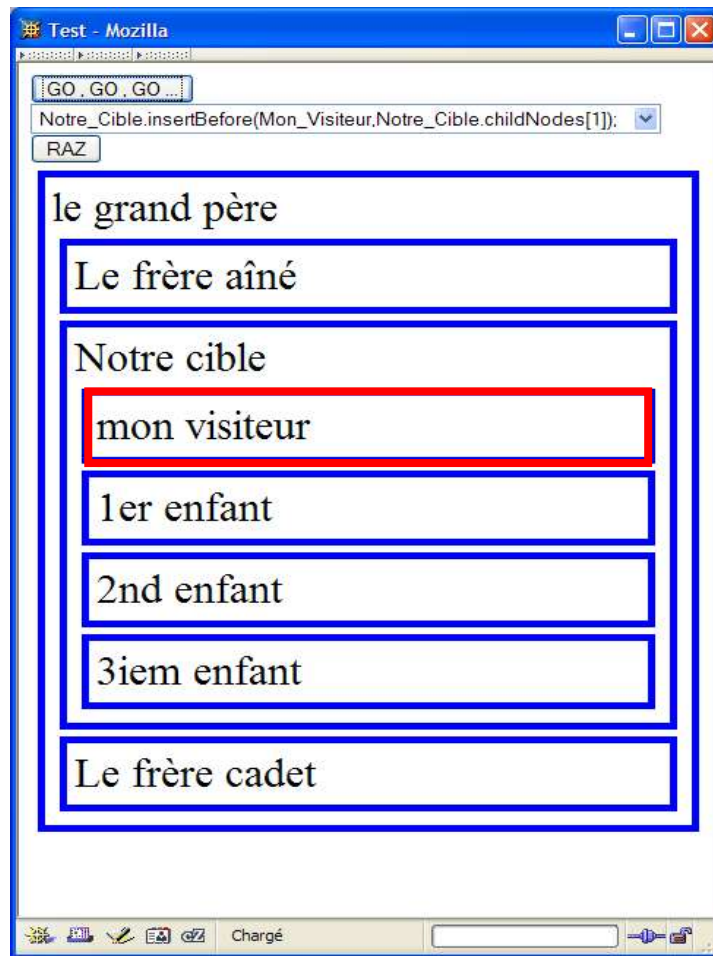
Variation sur un thème



- Place **Notre_Cible** avant le 1er fils de **Notre_Cible**
- `Notre_Cible.insertBefore(Mon_Visiteur, Notre_Cible.firstChild);`
- Pour ceux qui n'ont pas suivi le 1er fils est en fait l'élément `#text`

Le support d'exemples :

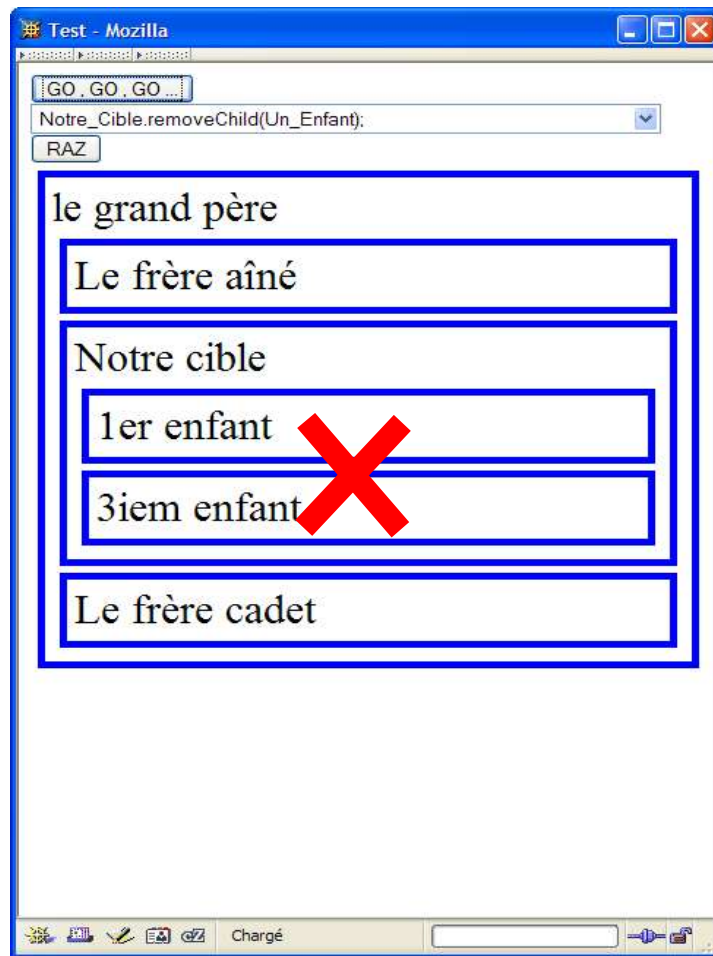
Variation sur un thème



- Place **Mon_Visiteur** avant le 2nd fils de **Notre_Cible**
- `Notre_Cible.insertBefore(Mon_Visiteur, Notre_Cible.childNodes[1]);`
- Rappel : **childNodes** est un tableau, les indices commencent à 0, donc 1 désigne le 2nd

Le support d'exemples :

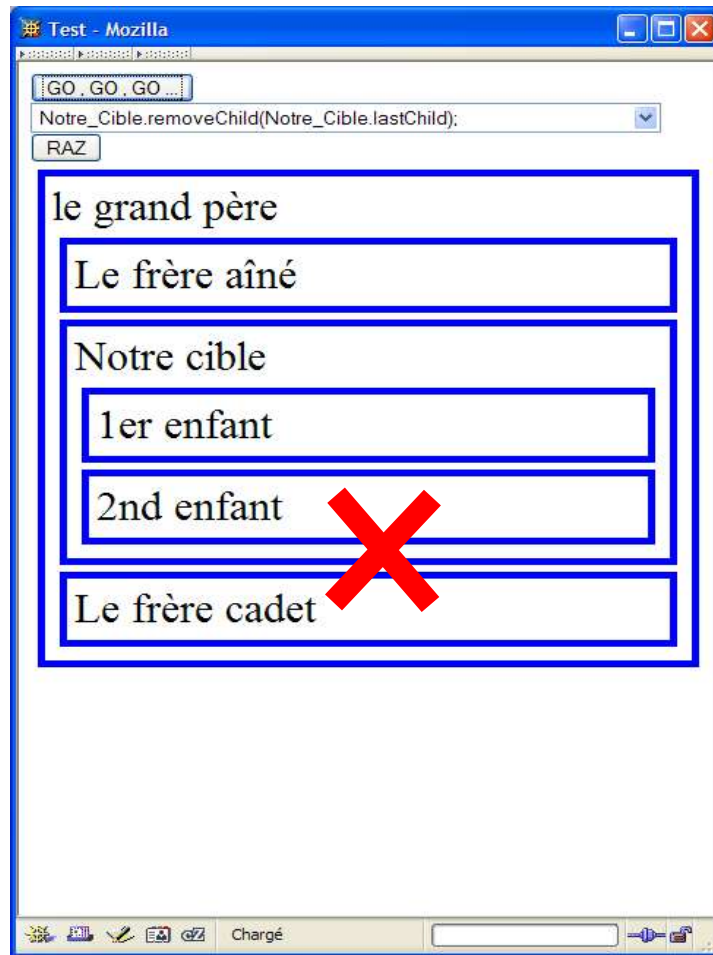
Variation sur un thème



- Efface `Un_Enfant`
- `Notre_Cible.removeChild(Un_Enfant);`

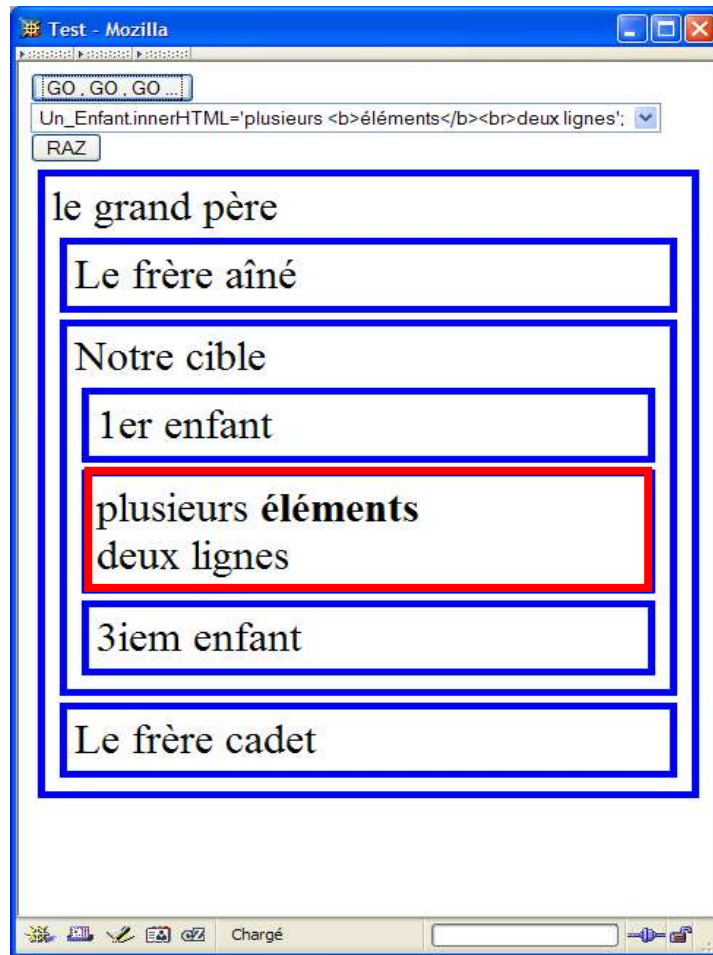
Le support d'exemples :

Variation sur un thème



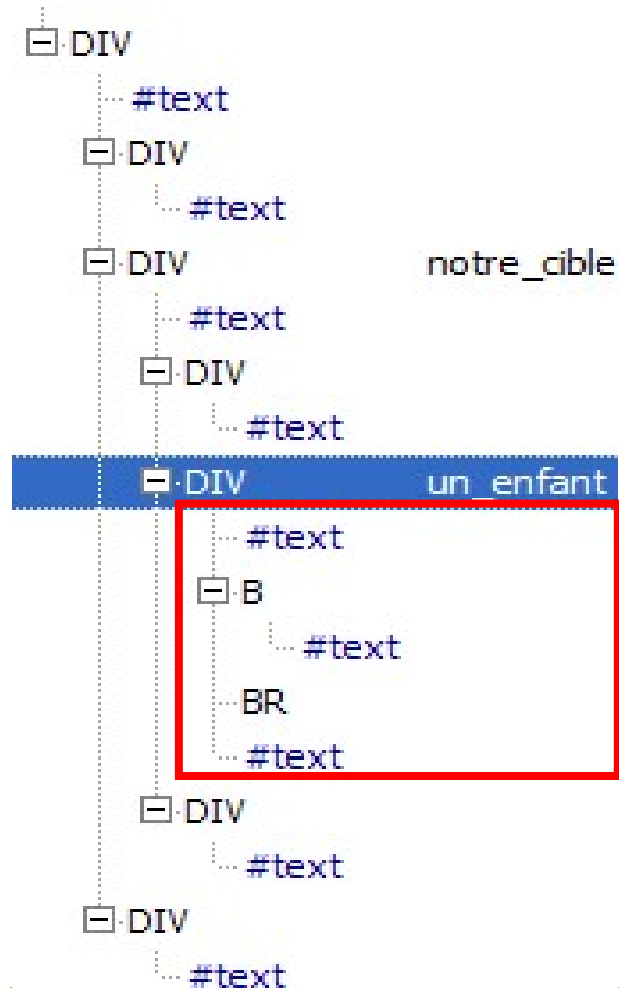
- Efface le dernier enfant de `Notre_Cible`
- `Notre_Cible.removeChild(Notre_Cible.lastChild);`
- Il est possible qu'il faille appuyer 2 fois, car le dernier élément peut être un élément vide !

La propriété innerHTML



- La **propriété innerHTML** désigne le code HTML présent dans l'élément.
- Ce n'est pas une fonction standard, mais elle est très pratique
- Ici on modifie le contenu de **Un_Enfant**
- **Un_Enfant.innerHTML =**
'plusieurs éléments
deux lignes';

La propriété innerHTML



- Tout le contenu de **Un_Enfant** a été remplacé par le code **HTML** 'plusieurs `éléments
` deux lignes';
- On peut remarquer que cela a donné un ensemble d'**éléments** qui est le résultat de l'analyse de la chaîne de caractères.