javascript : manipuler DOM

au programme...

au programme...

le type `Node`

javascript propose des fonctionnalités pour manipuler la structure de l'arbre DOM :

- accès aux informations sur les nœuds de l'arbre
- création de nouveaux nœuds
- insertion, suppression, déplacement de nœuds dans l'arbre

toute manipulation de la structure l'arbre a une répercussion immédiate sur le document affiché

type `Node`

Les nœuds de l'arbre DOM sont des objets de type `Node`.

l'objet `Node`

Un objet `Node` propose les propriétés :

- `nodeName` : le nom du nœud
- `nodeType` : le type du nœud défini par des constantes nommées prédéfinies, `Node.ELEMENT_NODE` (= 1), `Node.TEXT_NODE` (= 3)
- `nodeValue` : `null` si ce nœud est un nœud élément, le contenu pour un nœud texte
- `parentNode` : son nœud parent
- `childNodes` : la liste de ses nœuds enfants
- `firstChild`, `lastChild` : premier, dernier de ses nœuds enfants
- `previousSibling`, `nextSibling` : le nœud frère précédent, suivant
- etc.

`Node` sur MDN – `nodeType` sur MDN – exemple_dom.html

création de nœuds

`document.createElement(balise)`
crée un nouveau nœud avec la *balise* donnée

`document.createTextNode(text)`
crée un nouveau nœud texte avec pour contenu *text* (non interprété)

le résultat est le nœud créé

création de nœuds

`document.createElement(balise)`
crée un nouveau nœud avec la *balise* donnée

`document.createTextNode(text)`
crée un nouveau nœud texte avec pour contenu *text* (non interprété)

le résultat est le nœud créé

`node.cloneNode(prof)`
crée un nouveau nœud copie de *node*

création de nœuds

`document.createElement(balise)`
crée un nouveau nœud avec la *balise* donnée

`document.createTextNode(text)`
crée un nouveau nœud texte avec pour contenu *text* (non interprété)

le résultat est le nœud créé

`node.cloneNode(prof)`
crée un nouveau nœud copie de `node`

mêmes attributs mais pas listeners, doit être ajouté au document

- `prof == true` clone également tous les descendants
- `prof == false` seul le nœud est cloné, pas ses descendants

insertion dans le document

noeudParent.**insertBefore**(*noeud*,*noeudRéférence*)
insère *noeud* **avant** *noeudRéférence* comme fils de *noeudParent*

insertion dans le document

noeudParent.**insertBefore**(*noeud*,*noeudRéférence*)
insère *noeud* **avant** *noeudRéférence* comme fils de *noeudParent*

nb : *noeudRéférence*.parentNode == *noeudParent*

insertion dans le document

noeudParent.**insertBefore**(*noeud*,*noeudRéférence*)
insère *noeud* **avant** *noeudRéférence* comme fils de *noeudParent*

nb : *noeudRéférence*.parentNode == *noeudParent*

parent.**appendChild**(*noeud*)
noeud est ajouté à la fin des fils de *parent*

insertion dans le document

noeudParent.**insertBefore**(*noeud*,*noeudRéférence*)
insère *noeud* **avant** *noeudRéférence* comme fils de *noeudParent*

nb : *noeudRéférence*.parentNode == *noeudParent*

parent.**appendChild**(*noeud*)
noeud est ajouté à la fin des fils de *parent*

nb : si le nœud *inséré* ou *ajouté* existe déjà dans le document, il est alors **déplacé**, donc supprimé de la position existante et inséré/ajouté à la position demandée.

suppression et remplacement

parent.**removeChild**(*noeud*)
noeud est supprimé des fils de *parent*

suppression et remplacement

parent.**removeChild**(*noeud*)
noeud est supprimé des fils de *parent*

parent.**replaceChild**(*remplaçant*,*remplacé*)
remplaçant prend la place de *remplacé* comme fils de *parent*

suppression et remplacement

parent.**removeChild**(*noeud*)
noeud est supprimé des fils de *parent*

parent.**replaceChild**(*remplaçant*,*remplacé*)
remplaçant prend la place de *remplacé* comme fils de *parent*

nb : ces deux fonctions ont pour résultat le nœud supprimé/remplacé

exemple_dom2.html

à suivre...

javascript : clôtures