1-Rappel de quelques propriétés et méthodes du DOM

1-1-Objet document :

createAttribute()
créer un nœud d'attributs

createElement()
créer un nœud d'éléments (non inséré dans l'arborescence)

createTextNode()
créer un nœud de texte (non inséré dans l'arborescence)

getElementById()
Accès à l'élément HTML par l'attribut Id

getElementsByName()
Accès à l'élément HTML par l'attribut name

getElementsByTagName()
Accès à l'élément HTML par liste d'éléments

write()
écrire dans la fenêtre du document

1-2-Objet node(noeud) :

attributes
attributs

childNodes
nœud enfant

data
données en caractères

firstChild
premier nœud enfant

lastChild
dernier nœud enfant

nextSibling
prochain nœud d'un type

nodeName
nom du nœud

nodeType
type du nœud

nodeValue
valeur/contenu du nœud

parentNode
nœud parent

previousSibling
nœud précédent d'un type

innerHTML
valeur/contenu HTML d'un noeud

appendChild()
ajouter un nœud enfant

appendData()
ajouter des données en caractères

cloneNode()
copier un nœud

deleteData()
effacer des données en caractères

getAttribute()
rechercher la valeur d'un nœud attribut

hasChildNodes()
vérifier l'existence de nœuds enfants

insertBefore()
insérer un nœud

insertData()
insérer des données en caractères

removeAttribute()
effacer la valeur d'un nœud attribut

removeChild()
effacer un nœud

`replaceChild()`
remplacer un nœud enfant

`replaceData()`
remplacer des données en caractères

`setAttribute()`
fixer la valeur d'un nœud attribut

`setAttributeNode()`
créer un nœud attribut

2-Introduction à AJAX (Asynchronous JavaScript XML)

Les deux principaux avantages d'AJAX résident dans sa capacité à :
-envoyer des requêtes asynchrones au serveur donc sans rechargement de la page
-analyser et travailler avec des documents XML.

AJAX emploie l'objet non standard XMLHttpRequest() pour communiquer avec des scripts situés sur le serveur. Un requête se passe en 3 temps :

1-Instanciation de l'objet XMLHttpRequest
2-Ecoute et gestion de la réponse du serveur
3-Envoie de la requête au serveur

2-1- Instancier l'objet de requêtes XMLHttpRequest

Pour faire une requête HTTP vers le serveur à l'aide de JavaScript, il faut disposer d'une instance d'un objet fournissant cette fonctionnalité. Un tel objet a d'abord été introduit dans Internet Explorer sous la forme d'un ActiveX appelé XMLHTTP. Par la suite, Mozilla, Safari et d'autres navigateurs ont suivi en implémentant un objet XMLHttpRequest qui fournit les mêmes méthodes et propriétés que l'objet ActiveX original de Microsoft.

Par conséquent, pour créer une instance (un objet) de la classe désirée fonctionnant sur plusieurs navigateurs, vous pouvez utiliser une fonction regroupant ces deux objets (**`ajaxlib.js`**) :

```
function getXhr() {
    var xhr = null;
    if(window.XMLHttpRequest) xhr = new XMLHttpRequest();
    else if(window.ActiveXObject) {
        try {
            xhr = new ActiveXObject("Msxml2.XMLHTTP");
        } catch (e) {
            xhr = new ActiveXObject("Microsoft.XMLHTTP");
        }
    }
    else xhr = false;

    return xhr;
}
```

2-2-Ecoute et gestion de la réponse du serveur

Pour écouter et gérer la résponse du serveur, on appel la fonction **getXhr()** et on utilise l'évènement **`onreadystatechange`** de l'objet **`xhr`** pour indiquer quelle fonction traitera la réponse.

```
<script type="text/javascript" src=ajaxlib.js">
//inclusion de l'objet getXhr()
</script>
<script type="text/javascript">
function makeRequest(){
    xhr=getXhr();
    xhr.onreadystatechange=function(){
        //instructions de traitement de la réponse
    }
}
</script>
```

Vous remarquerez que l'on utilise une fonction anonyme pour faire référence à cette fonction sans l'appeler.

Tout d'abord, cette fonction doit vérifier l'état de la requête avec la propriété **readyState**.
Cette propriété renvoie 5 valeurs selon l'état de la requête :

* 0 (non initialisée)
* 1 (en cours de chargement)
* 2 (chargée)
* 3 (en cours d'interaction)
* 4 (terminée)

On peut donc tester l'état de la requête ainsi :

```
<script type="text/javascript" src=ajaxlib.js">
//inclusion de l'objet getXhr()
</script>
<script type="text/javascript">
function makeRequest(){
    xhr=getXhr();
    xhr.onreadystatechange=function(){
        if(xhr.readyState==4){
            //la réponse à été reçue
        }
        else{
            //réponse en cours
        }
    }
}
</script>
```

La seconde vérification concerne le code d'état de la réponse HTTP du serveur et se fait à l'aide de la propriété **status**. Nous sommes seulement intéressés par la réponse 200 OK.

```
<script type="text/javascript" src=ajaxlib.js">
//inclusion de l'objet getXhr()
</script>
<script type="text/javascript">
function makeRequest(){
    xhr=getXhr();
    xhr.onreadystatechange=function(){
        if(xhr.readyState==4){
            if(xhr.status==200){
                // traitement de la réponse.
            }
            else{
                //problème de réponse HTTP (404, 403, 500)
            }
        }
        else{
            //réponse en cours
        }
    }
}
</script>
```

Après avoir vérifié l'état de la requête et le code d'état HTTP de la réponse, vous pouvez traiter à votre guise les données envoyées par le serveur. Il existe deux propriétés permettant d'accéder à ces données :

*__responseText__ – renvoie la réponse du serveur sous la forme d'une chaîne de texte
* __responseXML__ – renvoie la réponse sous la forme d'un objet XMLDocument que vous pouvez parcourir à l'aide des fonctions DOM de JavaScript

```
<script type="text/javascript" src=ajaxlib.js">
//inclusion de l'objet getXhr()
</script>
<script type="text/javascript">
function makeRequest(){
    xhr=getXhr();
    xhr.onreadystatechange=function(){
        if(xhr.readyState==4){
            if(xhr.status==200){
                var result=xhr.responseText;
                alert(result)// par ex.
            }
            else{
                //problème de réponse HTTP (404, 403, 500)
            }
        }
        else{
            //réponse en cours
```

```
        }
    }
}
</script>
```

2-3-Envoi de la requête au serveur

Ensuite, après avoir déclaré ce qui se produit lorsque la réponse est reçue, il s'agit de lancer effectivement la requête. Il faut pour cela appeler les méthodes **`open()`** et **`send()`** de la classe de requête HTTP, comme ceci :

```
<script type="text/javascript" src=ajaxlib.js">
//inclusion de l'objet getXhr()
</script>
<script type="text/javascript">
function makeRequest(){
    xhr=getXhr();
    xhr.onreadystatechange=function(){
        if(xhr.readyState==4){
            if(xhr.status==200){
                var result=xhr.responseText;
                alert(result)// par ex.
            }
            else{
                //problème de réponse HTTP (404, 403, 500)
            }
        }
        else{
            //réponse en cours
        }
    }
    xhr.open("GET","fichier.xml",true);
    xhr.send(null);
}
</script>
```

Le premier paramètre de l'appel à **`open()`** est la méthode de requête HTTP – **`GET`**, **`POST`** ou toute autre méthode que vous voulez utiliser et est gérée par le serveur. Laissez le nom de la méthode en majuscules comme spécifié par la norme HTTP ; autrement certains navigateurs (comme Firefox) peuvent ne pas traiter la requête.

Le second paramètre est l'URL de la page que vous demandez. Pour des raisons de sécurité, il n'est pas possible d'appeler des pages se situant sur un autre domaine. Veillez à utiliser le nom de domaine exact sur toutes vos pages ou vous obtiendrez une erreur 'permission denied' à l'appel d'**`open()`**.

Le troisième paramètre précise si la requête est asynchrone. Si mis à **true**, l'exécution de la fonction JavaScript se poursuivra en attendant l'arrivée de la réponse du serveur. C'est le A d'AJAX.

Le paramètre de la méthode **send()** est **null** si on utilise la méthode GET ou une chaine représentant les données à envoyer :
nom=valeur&autrenom=autrevaleur

En cas de requête de type POST, vous devez ajouter une entête de type :
setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');

ex d'envoi en POST :

```
<script type="text/javascript" src=ajaxlib.js">
//inclusion de l'objet getXhr()
</script>
<script type="text/javascript">
function makeRequest(){
    xhr=getXhr();
    xhr.onreadystatechange=function(){
        if(xhr.readyState==4){
            if(xhr.status==200){
                var result=xhr.responseText;
                alert(result)// par ex.
            }
            else{
                //problème de réponse HTTP (404, 403, 500)
            }
        }
        else{
            //réponse en cours
        }
    }
    parameter="name="+document.getElementById('name').value;
    xhr.open("POST","fichier.txt",true);
    xhr.setRequestHeader('Content-Type', 'application/x-www-
    form-urlencoded');
    xhr.send(parameter);
    return false;
}
</script>
```

Vous noterez la présence de **return false** en fin de fonction **makeRequest**. Ce retour permet de ne pas envoyer les données du formulaire vers le fichier notifié dans l'attribut **action** du formulaire et de restreindre l'action du formulaire au code AJAX. En cas de désactivation de javascript, le formulaire continuerait cependant de fonctionner en envoyant ses données vers le fichier notifié dans **action**.

Ex de script d'envoi de données POST :

```
<script type="text/javascript" src="ajaxlib.js"></script>
<script type="text/javascript">
window.onload=init;

function init(){
    document.getElementById('formul').onsubmit=makeRequest;
}

function makeRequest() {

    var xhr=getXhr();
    xhr.onreadystatechange = function(){writeResult(xhr);}
    xhr.open('POST','http://localhost/search.php', true);
    xhr.setRequestHeader("Content-type","application/x-www-form-
urlencoded");
    parameters="keyword="+document.getElementById
('keyword').value;
    xhr.send(parameters);
    return false;
}

function writeResult(xhr){
    document.getElementById('result').innerHTML="Recherche des
r&eacute;sultats";
    if (xhr.readyState == 4) {
        if (xhr.status == 200) {
        result=xhr.responseText;
        document.getElementById('result').innerHTML=result;
        }
        else{
            alert('There was a problem with the request.');
        }
    }
}
</script>



<form id="formul" method="post" action="search.php">
<p>
<input type="text" name="keyword" id="keyword" value=""/>
<input type="submit" value="chercher" />
</p>
</form>
<ul id="result">
</ul>
```

2-4-Lecture et analyse de documents XML

un document XML reçu à l'aide de **`responseXML`**, est lu différemment selon les navigateurs. En effet, Firefox interprète les espace et les tabulations comme des noeuds alors qu'IE ne les interprète pas.

On pourra donc utiliser une fonction permettant de *nettoyer* le document pour l'analyse de façon unifiée. (cette fonction peut être placée dans la librairie ajaxlib.js)

```
function nodeCleaner(n) {
    if(!n.data.replace(/\s/g,'')) n.parentNode.removeChild(n);
}

function cleanXML(docElement) {
    var node = docElement.getElementsByTagName('*');
    for(i = 0; i < node.length; i++) {
        a = node[i].previousSibling;
        if(a && a.nodeType == 3) nodeCleaner(a);
        b = node[i].nextSibling;
        if(b && b.nodeType == 3) nodeCleaner(b);
    }
    return docElement;
}
```

lors de la récupération du document XML, on pourra appeler cette fonction ainsi :

```
var result=cleanXML(xhr.responseXML.documentElement);
```

ressources :
http://developer.mozilla.org/fr/docs/AJAX:Premiers_pas
http://javascript.developpez.com/cours/
http://www.xul.fr/xml-ajax.html
http://fr.selfhtml.org/javascript/index.htm

fpumir@maraboutee.net
professionnel indépendant
SIRET # 440 049 252 00024
formateur enregistré auprès du préfet de la région Bretagne sous le # 53350735335
chef de projet développeur formateur