Javascript

Dans cette partie du cours, on s'intéresse à un langage qui permet de programmer dans le navigateur ! En effet, le langage javascript est un langage coté client, ie il s'exécute dans le client Web pas sur le serveur.

1. Introduction

Le javascript est un langage interprété, qui ne donne pas lieu au processus habituel de compilation/Edition de lien. L'avantage est qu'il n'est pas nécessaire de compiler/distribuer pour chaque navigateur/système d'exploitation/processeur client. A la place, on a besoin juste d'un interpréteur. Ce rôle est rempli par un composant du client HTTP, appelé moteur Javascript.
Les moteurs Javascript les plus connus sont (Carakan utilisé par Opéra), V8 de Google (utilisé par Opéra 15+ et Google Chrome), Chakra (utilisé dans Internet Explorer), SpiderMonkey (utilisé dans Firefox et toutes les applications Mozilla basée sur Geko) et JavaScriptCore (utilisé dans Safari).

Javascript est un langage qui a été crée en 1995 par Brandan Eich de Netscape Communications et utilisé dans le navigateur « Netscape Navigator » puis dans Firefox (le successeur de Netscape Navigator). Aujourd'hui c'est le nom de l'implémentation d'un standard international appelé ECMA-62. D'autres implémentations de ce standard sont le langage Jscript de Microsoft dans IE et ActionScript, implémentation d'Adobe dans Flash.

Javascript est un langage côté client, interprété (de script), orienté objet et à typage implicite (la première déclaration/affectation détermine le type d'une variable).

Les balises <script> </script> peuvent être utilisées soit dans le <head> soit partout dans le document HTML pour y insérer du code J.S. Lorsque un fichier J.S. est importé, l'attribut src='URL du fichier.js' est utilisé.
Le mot clé **var** permet de déclarer des variables simples ou objet. Le type est implicite à la première affectation d'une valeur à une variable. Le mot clé **function**, lui permet de définir des fonctions J.S.

2. Notion de Contexte Global

C'est l'ensemble des objets existants et pouvant être accédés et utilisés dans du code J.S.
Exemples :
+L'objet prédéfini window qui contient tous les objets et fonctions globaux de J.S.
+L'objet location, qui est un attribut de window permet d'accéder à toutes les informations possibles sur l'URL actuellement visitée. window.location.reload() : est un appel pour recharger la page actuelle.
+L'objet history, membre de window contient l'ensemble de l'historique de navigation. window.history.forward() permet d'avancer dans l'historique, window.history.go(1) permet d'aller à la première page de l'historique de navigation, etc.
+Les fenêtres modales alert('Message') reponse=confirm('Message') et msg=Prompt('Question') sont globales

+Le DOM (Document Object Model) : objet global représentant l'ensemble de la page HTML actuellement affiché par le navigateur sous forme d'un arbre dont la racine est l'objet « document ».

3. Le DOM

Le DOM est une API qui permet d'accéder aux éléments HTML,
-d'en sélectionner des portions, de les modifier
-de modifier les propriétés CSS
-d'accéder au contenu texte des éléments HTML
En pratique c'est un arbre dont la racine est l'objet **document**, et les noeuds sont les balises de la page HTML représentée. Chaque balise de la page HTML est représentée par un noeud dans le DOM. La balise fille 'BFille' contenue dans une la balise parente 'Bparent' dans la page HTML sera représentée dans le DOM par un objet OBFille dont le parent est l'objet OBParent qui représente la balise Bparent. Les attributs HTML d'une balise deviennent de fait des attributs membres de l'objet la représentant dans le DOM. Ainsi, l'objet **head** devient un objet fils de l'objet document ainsi que l'objet **body**.
On peut parcourir les enfants de tout objet dans le DOM par des méthodes JS prédéfénie : firstChild renvoie la référence du premier enfant d'un noeud, lastChild renvoie quant à elle la référence du dernier objet fils. La liste des enfants directs est disponible dans une table/liste childNodes[i] où i dans [0..n] et n = childNodes.length()-1. On peut aussi connaître la référence du parent d'un noeud avec la propriété spéciale parentNode.
Ainsi document.body.firtChild.parentNode=document.body !

Mais cette façon de parcourir le DOM n'est pas très aisée pour chercher et référencer un objet. D'où l'intérêt des fonctions getElementById, getElementByTagNames, getElementsByClassName et autres qui permettent d'obtenir la référence d'un objet en connaissant seulement une de ses caractéristiques (son id, son nom d'élément , sa classe, etc...).

3.1. Accéder à un élément HTML ou chercher sa référence

Pour pouvoir manipuler les objets du DOM, il faut d'abord savoir comment chercher les références d'un ou de plusieurs objets. J.S. offre un ensemble de méthodes membre de l'objet document qui permettent de résoudre ce problème.

A- En connaissant son attribut 'id'
var reference=document.getElementById('nomdidentificateur') permet d'obtenir une référence sur une balise HTML en utilisant sont attribut id.

Exemple
HTML : <input type='password' id='pwd1'>
J.S. : var adresse_pwd1 = document.getElementById('pwd1') ;
```
if( adresse_pwd1.value == '')
    alert('Veillez saisir votre mot de passe') ;
```
N.B : Remarquez dans cet exemple la valeur de l'attribut id de la balise input. Remarquez aussi comment on a utilisé l'attribut membre 'value' de l'adresse, car il existe un attribut de la balise <input> qui s'appelle value.

B- Chercher toutes les balises d'un type donné
Se fait grace à la méthode getElementByTagNames qui prend comme paramètre le nom d'un élément HTML comme div, p, span ,...etc.
Exemple :
var divs = document.getElementByTagNames('div') ;
divs sera une liste de references sur tous les objets du DOM qui représentent une balise dans le document HTML. La longueur de toutes listes en J.S. peut être calculée grace à la fonction membre pré-définie length().
Exemple : var nombreDeDivs= divs.length() ;
On peut ensuite accéder à topus les élements de la liste comme suit
for (i=0 ; i< nombreDeDivs ; i++){
divs[i].nodeName, divs[i].innerHTML= ….
}

C-Chercher toutes les balises d'une classe donnée
grace à la méthode getElementsByClassName('NomDeClasse') qui retourne toutes les références des objets qui ont un attribut class (qui est attribut de toutes les balises HTML) qui contient la class 'NomDeClasse'. Rappelz-vous que l'attribut class prend une liste de classes comme class = 'classTexteRouge classTextePenche classTexteSouligne', ie un ensemble de nom de classe CSS.

document.getElementsByClassName('NomDeClasse') retrourne une liste qu'on peut parcourrir comme dans B-.

D-Chercher toutes les balises qui satisfont un selecteur CSS
var listeBalises=document.querySelectorAll('SelecteurCSS') ; retourne la liste des objets qui satosfont le selecteur CSS en paramètre. Liste parcourable comme dans B-.

E-Chercher la première balise qui satisfait un sélecteur CSS
var adresseBalise=document.querySelector('SelecteurCSS') retourne la première reference qui possède le sélecteur indiqué.

3.2. Modifier/ Editer un noeud

Une fois la référence d'un objet trouvée, on peut accéder à ses attributs/méthodes et les modifier à volonté.

A-Modifier le contenu texte : attribut **textContent**

Exemple

HTML :

Accès et Modification avec J.S. :

var scoreElmt=document.getElementByClassName('LeScore') ;

var referenceScore= scoreElmt.firstChild ; //Fonction memebre d'une liste qui retourne le premier

//élément d'une liste, lastChild renvoie le dernier
//élément de la liste.

ReferenceScore.**textContent**= 12 ; //On modifie le conteu du span !

B-Modifier les éléments HTML

On peut modifier leur contenu (attribut **innerHTML**), styles, propriétés et attributs.

3.3. Fonctions membres J.S. prédéfinies

variable.**hasAttribute**('nomAttribut') retourne true si la varable variable possède un attribut membre qui s'appelle 'nomAttribut', false sinon.

variable.**getAttribute**('nomAttribut') retourne la valeur de l'attribut 'nomAttribut'.

variable.**setAttribute**('nomAttribut','NouveauContenuAttribut') modifie l'attribut 'nomAttribut' de la balise représentée par variable pour y mettre la valeur 'NouveauContenuAttribut'.

variable.**removeAttribute**('nomAttribut') supprime un attribut

Exemple récapitulatif :

if (divs[i].hasAttribute('class')){

divs[i].setAttribute('class', 'TexteRouge') ;//Change la classe

divs[i].setAttribute('style', 'font-size:15px ;') ;//Change la police dans l'attribut style !

divs[i].**innerHTML**=' Ceci est un code HTML inséré dans ce div ' ;

}

3.4. Pour créer un noeud dans le DOM

La méthode prédéfinie createElement('nomDeBalise') permet de créer un élément HTML de n'importe quel type (, ,<article>,<header>....<audio>....etc. Après sa création, il faut le placer dans le DOM en renseignant sa propriété parentNode. Ensuite, on pourra lui affecter, un contenu, un style, etc.

Exemple : On crée un élément audio, on vérifie qu'il accepte un format, on charge un fichier audio adapté et on lance l'audio.

```
var audio=document.createElement('audio') ;
audio.parentNode=document.body ;
if (audio.canPlayType('audio/mp3')){
     audio.setAttribute('src','/sounds/alert1.mp3') ;
}else{
     if (audio.canPlayType('audio/ogg')){
```

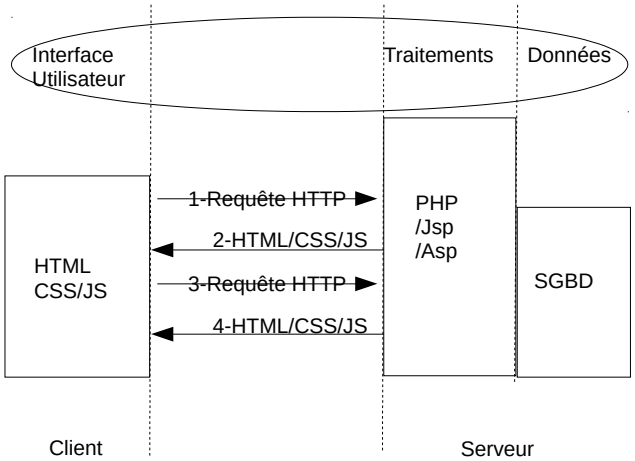
```
        audio.setAttribute('src','/sounds/alert1.ogg') ;
        }
    }
    audio.play() ;
```

4. Applications AJAX

Ajax : Asynchronous Javascript And XML (eXtended Markup Language)

Si au début de son existence, le langage JS a surtout permis d'implanter les contrôles d'erreur des champs de formulaires saisis par les utilisateurs (comme dans Exercice 1 de TP N°4) et de faire des animations (HTML dynamique), son existence a permis pour la première fois d'envisager d'effectuer une partie des traitements faits sur le serveur dans le navigateur. Ceci a pour grand avantage de nécessiter moins de ressources côté serveur, en termes de machines/énergie electrique/ressources humaines ;

Pour l'utilisateur, les avantages sont que sa bande passante, généralement déjà restreinte, n'est pas utilisée pour télécharger du HTML/CSS pour chaque page, mais uniquement les données. Regardez l'exemple suivant pour avoir une idée sur l'utilisation de l'objet XMLHttpRequest afin de créer une page HTML rien qu'avec du J.S. (http://help.dottoro.com/ljspgwvf.php).



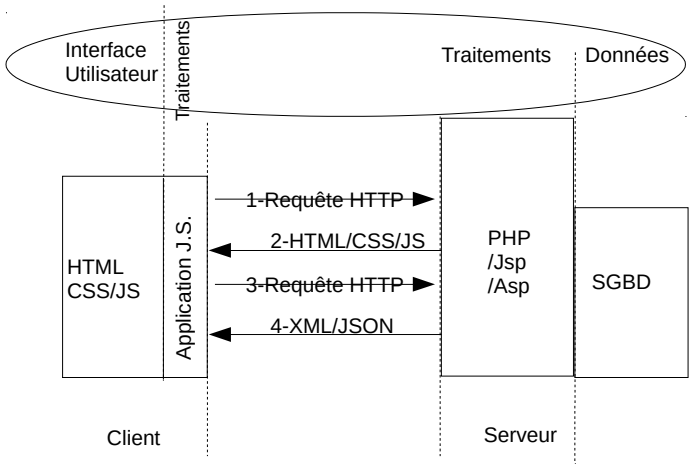
Application web 2-tiers (C/S). Le dialogue HTTP concerne des ressources HTML/CSS/J.S.

Les données échangées entre le client et le serveur HTTP dans une application web AJAX sont formatées soit en XML (eXtended Markup Language) soit en JSON (JavaScript Object Notation). Le format JSON est devenu de plus en plus populaire à cause de ses avantages. En effet des données au format JSON peuvent facilement être utilisées pour créer des objets J.S. là où en format XML, il faut parcourir le DOM des données, créer des variables J.S. et leur assigner les valeurs lues dans le DOM.

Exemple utilisation JSON

```
<!DOCTYPE html>
<html>
<body>

<h2>JSON Object Creation in JavaScript</h2>

<p id="demo"></p>

<script>
var texte = '{"name":"John Johnson","street":"Oslo West 16","phone":"555
1234567"}';

var obj = JSON.parse(texte);

document.getElementById("demo").innerHTML =
obj.name + "<br>" +
obj.street + "<br>" +
obj.phone;
</script>

</body>
</html>
```

Dans l'exemple ci-dessus, la variables text est une chaine de caractère qui a était transformée en un objet grâce à la ligne JSON.parse(texte) (JSON.parse est offerte par tous les moteurs JS). En pratique, on utilise un objet XMLHttpRequest pour demander des données formatées comme la variable '**texte**' puis avec du code J.S. on crée un objet et on utilise ses attributs comme données. En utilisant XML, réaliser le même travail requiert beaucoup plus de code, d'où l'avantage du format JSON.



Application web **AJAX** 2-tiers. Le dialogue HTTP concerne des ressources HTML/CSS/J.S. seulement lors de la phase de l'accès à l'application. Par la suite, uniquement des données (au format XML ou JSON) sont échangées. Une partie des traitements se fait dans le client HTTP

5. Evénements Javascript

Dans le navigateur, on peut associer des fonctions JS à certains événements qui concernent tout objet du DOM.

En effet, à chaque action de l'utilisateur/réseau, un événement est généré. On peut associer une fonction à exécuter lorsque l'événement se réalise.

Exemple : Exercice de TP N°4 revisité.

```
<script>
var password1= document.getElementById('pwd1');
var password2= document.getElementById('pwd2');
var checkPasswordValidity=function(){
if(password1.value != password2.value){
 password2.setCustomValidity('Les mots de passe ne concordent pas');
```



```
}
else{
password2.setCustomValidity('');
}
};//fin de la fonction. Notez le ; à la fin car la fonction est une variable
password2.addEventListener('change', checkPasswordValidity, false);
</script>
```

Notez la foncion **addEventListener**. Elle reçoit en premier paramètre le nom d'un événement parmi les événements pré-définis possibles ; en second argument le nom de la fonction qu'il faut exécuter lorsque l'événement arrive, le troisième paramètre peut être true ou false comme il peut être omis (on passe sur son utilité ici).

Les événement dépendent de l'élément HTML considérés et on peut associer plusieurs fonctions à un élément pour plusieurs événements. par exemple :

```
document.getElementById("myBtn").addEventListener("mouseover", myFunction);
document.getElementById("myBtn").addEventListener("click", someOtherFunction);
document.getElementById("myBtn").addEventListener("mouseout", someOtherFunction);
```

Le nom de l'événement utilisé correspond au nom de l'attribut HTML dédié à l'événement considéré sans le préfixe 'on'. Par exemple, un élément HTML possède les attributs événementiels : onmouseover, onclick, onmouseout, onfocus, etc. En enregistrant des fonctions pour ses événements, on utilisera 'mouseover', 'click', 'mouseout', 'focus' comme paramètre à addEventListner !

6. Conclusion

Dans ce chapitre, nous avons abordé le langage de script javascript utilisé dans le navigateur (côté client). Avec les bibliothèques J.S. comme JQuerry, Angular etc. aujourd'hui on peut créer de nouvelles applications web appelée Ajax qui s'appuient sur ces framework pour faciliter le développement et le déploiement des applications web, en particulier dans leur partie exécutée dans le navigateur.