Haute École Bruxelles-Brabant
École Supérieure d'Informatique
Bachelor en Informatique

2020 - 2021
WEBR4-DIN

Développement Internet

JavaScript - Langage et DOM

Nous avons abordé les langages HTML et CSS qui nous permettent de structurer et présenter l'information. Ici nous allons aborder le moyen le plus utilisé pour apporter du dynamisme à la présentation et de l'interactivité avec l'utilisateur.

1 Présentation

JavaScript est un langage de scripts essentiellement [1] utilisé pour enrichir les pages HTML de code exécutable par le navigateur. Attention ! Il n'a guère qu'une partie de son nom en commun avec le langage Java !

Ce langage permet d'apporter du dynamisme au niveau des pages Web. Nous percevons immédiatement une contradiction entre le fait qu'un navigateur doive essentiellement se contenter de réaliser l'affichage de pages et celui de lui faire exécuter du code. On veillera donc à limiter au maximum l'utilisation de code JavaScript pour :

- éviter d'alourdir les documents HTML
- faciliter leur maintenance
- ne pas surcharger de traitements les postes de travail

Veillons aussi à produire des pages qui pourront être exploitées sur un maximum de types de navigateurs en n'oubliant pas que certains d'entre eux n'intègrent pas JavaScript (ex. certains browsers de smartphones) et que certains utilisateurs inhibent JavaScript.

JavaScript est le premier outil nous permettant de faire évoluer un navigateur de la notion de « client léger » à celle de « client riche ».

De manière générale, JavaScript est utilisé pour [2]

1. permettre de réaliser des validations de données saisies dans des formulaires
2. pour gérer les cookies
3. pour éviter d'augmenter inutilement le nombre de requêtes soumises au serveur Web
4. Il peut aussi permettre d'enrichir la présentation de la page (menus dynamiques par exemple)

1. L'utilisation de JavaScript s'étend de plus en plus vers de nouvelles cibles comme la programmation sur le serveur. Nous ne prendrons en compte ici que son utilisation pour les navigateurs.

2. Il peut aussi être utilisé pour faire clignoter une guirlande, faire traverser l'écran par un canard ou demander à un poisson de suivre votre souris (comme quoi il faut savoir ne pas abuser des bonnes choses. . .)

Version du document : 16 février 2021

2 Intégration de JavaScript

Comme à chaque fois, nous ne présentons que les aspects essentiels du langage et vous renvoyons sur le site de la W3School pour un tutoriel plus complet (http://www.w3schools.com/js/default.asp).

2.1 Intégration au HTML

Un script JavaScript sera fourni dans la balise [3] `script` avec l'attribut `type` de valeur `text/javascript` [4].

soit directement dans le code HTML :

```
<script type="text/javascript">
        //code du script...
</script>
```

soit par référence à un fichier externe :

```
<script type="text/javascript" src="scriptname.js"></script>
```

On préfèrera cette deuxième option par souci de lisibilité du code.

2.2 Déclenchement du code

Une balise `<script type="text/javascript">` peut apparaître soit dans le `header` de la page, soit dans le `body`. Le chargement du fichier de script JavaScript interrompt celui du fichier HTML. Tous les scripts invoqués dans le fichier JavaScript sont immédiatement exécutés lors du chargement dudit fichier.

2.3 Quelques éléments de syntaxe de JavaScript

- Les instructions sont délimitées par le point-virgule mais celui-ci n'est obligatoire que si l'on désire reprendre plusieurs instructions sur la même ligne.
- Les commentaires sont repris, sur une ligne derrière les caractères //
- Les noms de variables sont sensibles à la casse.
- Les instructions de contrôle de séquence (`if`, `while`,...) sont sensiblement les mêmes qu'en Java.

2.4 Un exemple simple

Plaçons ceci dans le body d'un de nos précédents documents HTML :

```
<script type="text/javascript">
        document.write("Added by JavaScript code : ")
        document.write("He2b-Esi <3")
</script>
```

3. Il existe également une version plus complexe tenant compte des vieux navigateurs qui ne gèrent pas le JavaScript.

4. En HTML5, on déconseille de spécifier explicitement le type s'il s'agit de `text/javascript` : https://developer.mozilla.org/en-US/docs/Web/HTML/Element/script

Nous pouvons aussi sauvegarder ce script à part dans un fichier `MyScript.js`, dont la référence relative au document HTML ou sa référence absolue sera fournie dans l'attribut source de l'élément `script`. Si le fichier se trouve dans le même répertoire que le document HTML, nous pourrons le référencer comme suit :

```
<script type="text/javascript" src="MyScript.js"></script>
```

3 Quelques utilisations classiques

3.1 Validation de formulaire

Un cas classique est celui de la validation d'un champ de saisie devant être numérique. Nous allons réaliser la validation lors de la perte du focus [5] par le contrôle.

```
<html>
<header>
        <meta charset="utf-8" />
        <title>Validation de formulaire</title>
        <script type="text/javascript">
                function validateNumeric(champ) {
                        if (isFinite(champ.value)) {
                                return true
                        }
                        else {
                                alert("Vous ne pouvez entrer que des nombres!")
                                return false
                        }
                }
        </script>
</header>

<body>
        <h1>Validation externe</h1>
        <form name="InfoPersonnelle">
                <b> Age : </b> <input type="text" name="saisie" onblur="validateNumeric
        </form>
</body>

</html>
```

5. Voir http://www.w3schools.com/tags/ref_eventattributes.asp pour une liste des événements gérés.

Remarques :

— D'autres événements peuvent vous intéresser : `onclick`, `onkeypress`, `onload`... En voici la liste complète : http://www.w3schools.com/jsref/dom_obj_event.asp.

— Par convention, si la méthode associée à l'événement renvoie `false` (comme dans l'exemple ci-dessus), cela signifie que l'événement n'a pas eu lieu : la page ne sera pas soumise, le caractère par entré,...

— Si nous désirons réaliser une validation lors de l'envoi du formulaire nous serons obligés d'écrire un script spécifique au formulaire.

— Nous verrons par la suite que nous veillerons à éviter l'utilisation des fonctions `alert()`, `confirm()`, `prompt()`,... qui ont tendance à alourdir l'interface.

3.2 Gestion des cookies

Un **Cookie** est petit fichier texte, chiffré ou non, que des sites visités peuvent stocker sur votre machine pour être récupérés d'une visite à l'autre (pour vous faciliter la navigation, pour vous tracer,...). Seul le site ayant stocké le cookie y a accès.

Un cookie est une paire `nom=valeur` associée à un serveur et est stocké sur le poste client. Lorsque le navigateur envoie une requête à un serveur, il lui envoie également tous les cookies associés.

Un cookie possède aussi une date d'expiration. Javascript permet, via la propiété cookie du document [6], de lire et de créer des cookies [7].

```
function setCookie( nom, valeur ) {
        document.cookie = nom + "=" + encodeURI(valeur);
}

function getCookie( nom ) {
        // Tous les cookies sont reçus dans une seule chaîne,
        // séparés par des point-virgules, qu'il faut découper.
        start = document.cookie.indexOf( nom + "=" );
        if ( start != -1 ) {
                start = start + nom.length+1;
                end = document.cookie.indexOf(";", start);
                if ( end ==-1 ) end = document.cookie.length;
                return unescape( document.cookie.substring(start,end) );
        } else {
                return "";
        }
}
```

6. https://developer.mozilla.org/en-US/docs/Web/API/Document/cookie
7. La version simplifiée qu'on présente ici peut être prise en défaut dans certains cas particuliers.

4 DOM (Document Object Model)

Le Document Object Model va nous permettre de manipuler (consulter/modifier) notre page web via le code javascript. On va pouvoir, par exemple, ajouter un message d'erreur à coté d'un champ erroné. C'est également un élément essentiel d'Ajax que nous verrons ultérieurement.

DOM (Document Object Model) [8] Le DOM, ou modèle objet de document, est une API pour les documents HTML et XML. Le DOM fournit une représentation structurelle du document, permettant de modifier son contenu et sa présentation visuelle. Fondamentalement, il relie les pages Web aux scripts et langages de programmation.

Concrètement, cela signifie qu'au niveau de javascript on dispose d'une représentation du document HTML sous forme d'un « arbre », chaque « nœud » correspondant à une balise du document. Cet arbre peut être parcouru mais surtout modifié. Les possibilités sont énormes mais nous l'utiliserons essentiellement :

- Pour faire apparaître des messages d'informations ou d'erreurs aux endroits appropriés de la page.
- Avec la technologie AJAX dont il est une des composantes essentielles.

4.1 Quelques éléments de syntaxe

Nous ne présentons que la toute petite partie qui sert à notre propos. Nous en dirons un peu plus lorsque nous introduirons les technologies XSL et, comme d'habitude, vous trouverez plus d'informations sur le site de la W3 school (http://www.w3schools.com/jsref/dom_obj_document.asp).

- Il existe un certain nombre d'objets prédéfinis. Vous apprendrez à l'usage les méthodes et attributs offerts : document (la racine de l'arbre), window (la fenêtre),...
- On peut accéder à un ou des nœuds de l'arbre :
  - `getElementById("id")` : le nœud correspondant à cet identifiant unique.
  - `getElementsByTagName("tag")` : un tableau des nœuds porteurs de ce tag.
  - `querySelector(selecteur css)` : le premier élément dans la descendance de l'élément appelant la méthode querySelector [9] compatible avec le sélecteur CSS en argument.
- On peut interroger un nœud via des attributs :
  - L'attribut `innerHTML` correspond au code HTML inclus dans un nœud.
  - L'attribut `style` donne le style associé à un nœud.
  - L'attribut `attributes` donne les attributs associés à un nœud.
  - Il y a aussi : `nodeType`, `nodeName`, `nodeValue`,...

8. Tiré de https ://developer.mozilla.org/fr/DOM
9. https://developer.mozilla.org/en-US/docs/Web/API/Element/querySelector

— On peut, à partir d'un nœud, se promener dans l'arbre : `parentNode`, `childNodes`, `firstChild`, `lastChild`, `nextSibling`, `previousSibling`.

— On peut aussi modifier l'arbre : `createElement`, `createTextNode`, `appendChild`, `insertBefore`, `replaceChild`, `cloneNode`, `removeChild`.

4.2 Exemple récapitulatif

Si le document contient `<span id="titre"></span>`, alors le bout de code suivant permet d'insérer un titre :

```
document.getElementById("titre").innerHTML = "Compléments Applications"
```

Le code suivant affiche un message avec le nombre de paragraphes (à lancer via l'événement onload de la balise `<body>` par exemple).

```
function countParagrs() {
        var myParagraphs = document.getElementsByTagName("p");
        alert( myParagraphs.length )
}
```

5 Découplage HTML / JavaScript

Tout comme il est de bonne pratique de découpler le contenu HTML de la mise en forme CSS, en liant le(s) fichier(s) CSS à l'aide de la balise `<link>`, il est conseillé de séparer HTML et JavaScript à l'aide de l'attribur src de la balise `<script>`.

Par ailleurs, il est également conseillé de découpler fortement les élément HTML du JavaScript en associant les gestionnaires d'événements dynamiquement en JavaScript plutôt que statiquement dans le document HTML.

Dans la suite, voyons en détail comment associer une fonction de validation à un champ d'un formulaire, comme cela a été déjà rapidement montré dans la première illustration donnée de l'utilité de JavaScript dans ce TD, en page 3.

5.1 JavaScript interne

Reprenons le document HTML avec un script interne :

```
<html>
<header>
        <meta charset="utf-8" />
        <title>Validation de formulaire</title>
        <script type="text/javascript">
                function validateNumeric(champ) {
                        if (isFinite(champ.value)) {
                                return true
                        }
                        else {
                                alert("Vous ne pouvez entrer que des nombres!")
                                return false
                        }
                }
        </script>
</header>

<body>
        <h1>Validation externe</h1>
        <form name="InfoPersonnelle">
                <b> Age : </b> <input type="text" name="saisie" onblur="validateNumeric
        </form>
</body>

</html>
```

5.2 JavaScript externe

5.2.1 Association statique

Le script est déplacé sur un fichier `formvalidate.js` :

```
function validateNumeric(champ) {
    if (isFinite(champ.value)) {
        return true
    }
    else {
        alert("Vous ne pouvez entrer que des nombres!")
        return false
    }
}
```

Et le fichier HTML est inchangé à l'exception du lien vers ce script :

```
<html>
<header>
        <meta charset="utf-8" />
        <title>Validation de formulaire</title>
        <script type="text/javascript" src="formvalidate.js"></script>
</header>

<body>
        <h1>Validation externe</h1>
        <form name="InfoPersonnelle">
                <b> Age : </b> <input type="text" name="saisie" onblur="validateNumeric
        </form>
</body>

</html>
```

5.2.2 Association dynamique

On cherche à associer dynamiquement un gestionnaire d'événement à un élément HTML. Pour ce faire, on utilise le DOM via Javascript dans un script `association_event.js`

```
document.getElementById("age").addEventListener("blur",function(event) {
    validateNumeric(document.getElementById("age").value);
});
```

Et on ajoute un lien vers ce script dans le code HTML :

```
<html>
<header>
        <meta charset="utf-8" />
        <title>Validation de formulaire</title>
        <script type="text/javascript" src="formvalidate.js"></script>
</header>

<body>
        <h1>Validation externe</h1>
        <form name="InfoPersonnelle">
                <b> Age : </b> <input type="text" name="saisie" />
        </form>

        <script type="text/javascript" src="association.js"></script>
</body>

</html>
```

Notez qu'on préfère charger les scripts d'association en fin de la balise `<body>`, puisque le code Javascript est exécuté dès qu'il est chargé ; les éléments HTML doivent donc être présents dans le DOM au moment de l'exécution du code.

5.2.3 Association dynamique avec `defer`

Si toutefois on veut charger le script à un autre endroit du code HTML, l'attribut `defer` [10] permet de reporter le chargement du script après la fin du chargement de la page HTML dans son entièreté. Cet attribut n'est disponible que lorsqu'un script est chargé via l'attribut `src`.

Par exemple, on pourrait avoir un fichier de script unique `formvalidate_association.js`

```
function validateNumeric(champ) {
    if (isFinite(champ.value)) {
        return true
    }
    else {
        alert("Vous ne pouvez entrer que des nombres!")
        return false
    }
}

const ageElement = document.getElementById("age");

ageElement.addEventListener("blur",function(event) {
    validateNumeric(document.getElementById("age").value);
});
```

Qui est chargé dans le `head` du HTML comme suit :

```
<html>
<header>
        <meta charset="utf-8" />
        <title>Validation de formulaire</title>
        <script defer type="text/javascript"
                src="formvalidate_association.js"></script>
</header>

<body>
        <h1>Validation externe</h1>
        <form name="InfoPersonnelle">
                <b> Age : </b> <input type="text" name="saisie" />
        </form>
</body>

</html>
```

10. https://flaviocopes.com/javascript-async-defer/

5.2.4 Association dynamique avec l'événement `DOMContentLoaded`

L'événement `DOMContentLoaded`[11] est émis lorsque le document HTML est chargé et traité dans son entièreté. On peut donc aussi s'en servir afin d'exécuter du code JavaScript une fois que le document complet est à notre disposition.

Notre script devient alors :

```
function validateNumeric(champ) {
    if (isFinite(champ.value)) {
        return true
    }
    else {
        alert("Vous ne pouvez entrer que des nombres!")
        return false
    }
}

document.addEventListener("DOMContentLoaded", function() {
    const ageElement = document.getElementById("age");

    ageElement.addEventListener("blur",function(event) {
        validateNumeric(document.getElementById("age").value);
    });
});
```

Qui est chargé dans le `head` du HTML comme suit :

```
<html>
<header>
        <meta charset="utf-8" />
        <title>Validation de formulaire</title>
        <script type="text/javascript"
                src="formvalidate_association_DOMContentLoaded.js"></script>
</header>

<body>
        <h1>Validation externe</h1>
        <form name="InfoPersonnelle">
                <b> Age : </b> <input type="text" name="saisie" />
        </form>
</body>

</html>
```

11. https://developer.mozilla.org/en-US/docs/Web/API/Document/DOMContentLoaded_event

5.2.5 Conclusion

La méthode recommandée est celle qui utilise l'attribut `defer` de la balise `<script>`. Elle est en effet :

— Propre : le HTML et le JavaScript sont séparés car `defer` oblige à utiliser l'attribut `src`

— Lisible : les balises `<script>` sont rassemblées dans le `<head>` du HTML

— Efficace : la récupération du JavaScript se fait de manière synchrone avec l'analyse du HTML

— Sûre : l'exécution des scripts JavaScript a lieu une fois le document HTML complètement construit.