DÉCOUVERTE DU LANGAGE JAVASCRIPT P2 - SÉQUENCE « DÉVELOPPER DES PAGES WEB »

P/2 Introduction

P/3 1. L'interactivité avec l'html

- 1.1 le DOM
- 1.2 JavaScript et les propriétés des éléments
- 1.3 JavaScript et les éléments e formulaires html

P/7 2. L'interactivité avec les événements

- 2.1 Généralités
- 2.2 Les types d'événements
- 2.3 Mettre en place des types d'événements
- 2.4 L'objet Event
- 2.5 Supprimer un événement

P/10 Glossaire

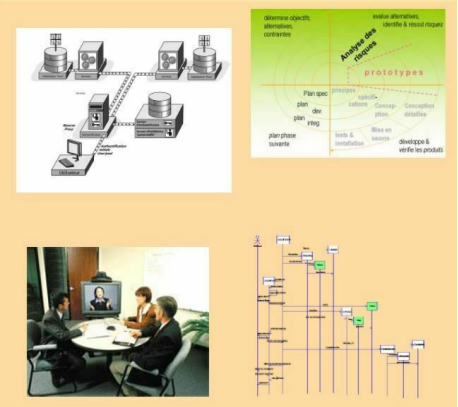
P/11 Crédits

Appui technique

Introduction

Bonjour et bienvenue dans ce module consacré au langage Javascript.

Découverte du langage JavaScript P2 - Apprentissage - Séquence « Développer des pages Web »



1. L'interactivité avec l'html

1.1 le DOM

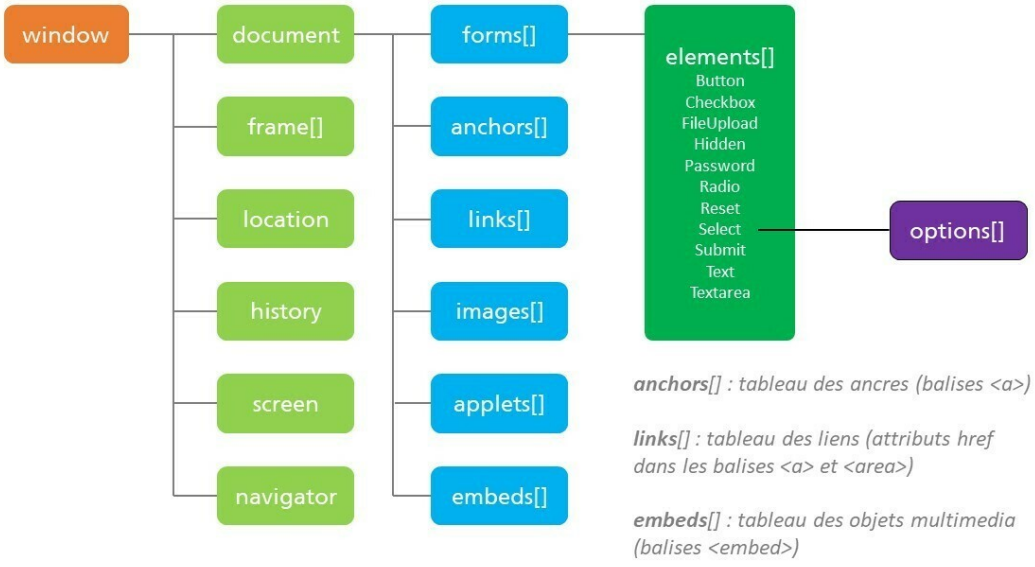
Le **DOM ou Document Object Model** est une interface de programmation (ou API [def. 1]) pour les documents XML et HTML ; c'est donc un ensemble d'outils qui permettent de faire communiquer entre eux, dans le cas présent, les langages HTML et JavaScript.

Le W3C [def. 2] a défini un DOM standard qui permet d'accéder à tous les éléments du document. Ce standard a évolué dans le temps, la version actuelle étant DOM-2. Il réside quelques différences entre les navigateurs en termes d'implémentation du DOM.

Remarque

Le DOM convertit la description Html en une structure arborescente d'objets JavaScript (plus rapide et plus facile à traiter).

A la racine, l'objet **window** représente l'instance du navigateur ; il référence principalement l'objet location qui symbolise la barre d'adresse, l'objet **history** qui représente l'historique des pages visitées par l'utilisateur, et l'objet document qui représente la page Web en cours, le contenu du **<body>** Html, lui-même référençant tous ses éléments dont le tableau images qui référence tous les éléments **<img>** de la page.



Historiquement, on pouvait accéder par JavaScript aux éléments de la page à travers un système de noms à tiroirs représentant l'arborescence de la page HTML.

Par exemple, l'instruction suivante retourne la valeur du premier élément HTML permettant la saisie dans le premier formulaire de la page.

```
window.document.forms[0].elements[0].value
```

Depuis l'arrivée du standard DOM-2, la méthode **.getElementByID()** de l'objet **document** permet d'adresser directement tout élément HTML doté d'un **attribut id**. Les méthodes **.getElementsByClassName()** et **.getElementsByTagName()** permettent d'adresser un tableau d'éléments selon leur **attribut** class ou leur **type d'élément**.

Enfin, les deux méthodes **.querySelector()** et **.querySelectorAll()** permettent de grandement simplifier la sélection des éléments dans l'arbre DOM ; ces deux méthodes prennent pour paramètre un ou plusieurs **sélecteurs CSS** séparés par des virgules.

Exemple :

```
var allLIAndOl = document.querySelectorAll("li,ol") ;
```

Conseil

Pour bien comprendre la correspondance entre les éléments HTML et les objets JavaScript, déroulez l'animation « **leDOMJavaScript.ppsx** » .

1.2 JavaScript et les propriétés des éléments

Pour récupérer, modifier ou ajouter des attributs aux éléments HTML, JavaScript possède les 2 fonctions **getAttribute()** et **setAttribute()**.

Exemple :

```
var elem = document.getElementById("div1");
var nomDiv1 = elem.getAttribute('name'); // récupère l'attribut 'name'
elem.setAttribute('class', "maClasse"); // ajoute l'attribut "class"
```

Chaque objet JavaScript correspondant à un élément Html de la page est automatiquement doté de propriétés correspondant aux attributs Html (en minuscule), et de méthodes permettant leur manipulation par programmation JavaScript.

Exemples :

```
window.location.href  // retourne l'URL courante de la page
window.location.replace(xxx) ; // modifie l'url courante du navigateur

var monImage = document.querySelector("#img1");
monImage.src = 'xxx.jpg' ;  // modifie sur la source de l'image

// Récupère le contenu de l'élément dont l'id = "msg"
var msg = document.querySelector("#msg").value;
var message = document.getElementById('msg').value;

// change la couleur de fond de page
document.bgColor = 'blue' ; // js reprend le système de couleurs html
```

Si un nom d'attribut Html est composé de plusieurs mots, la propriété correspondante en JavaScript attache les mots et force en majuscule la 1ère lettre de chaque mot (hormis le 1er mot).

Exemple :

```
document.querySelector("#nom").readOnly = true;
```

Certains attributs Html sont des mots-clés réservés en JavaScript. Exemples :

- L'attribut **for** devient la propriété **htmlFor**,
- L'attribut **class** devient la propriété **className ou classList**

JavaScript donne ainsi accès pour consultation ou modification aux classes de styles CSS des éléments Html ; il permet en outre d'accéder à tous les attributs CSS grâce à la propriété **style** qui elle-même dispose de nombreuses propriétés correspondant aux différents attributs CSS.

Exemple :

```
document.getElementById("div1").style.border = "3px solid blue";
```

Il existe 3 propriétés qui permettent de récupérer le code présent dans un élément du DOM :

- La propriété **innerHTML** : récupère le code Html inclus dans un élément,
- Les propriétés **textContent** et **innerText** : récupèrent le code brut sans les balises HTML.

o **innerText** est reconnue par IE et Chrome mais pas Firefox.

o **textContent** est reconnue par Firefox et Chrome mais pas IE.

1.3 JavaScript et les éléments e formulaires html

En ce qui concerne les formulaires Html, il est très fréquent de contrôler en JavaScript les saisies effectuées par l'utilisateur.

Le contenu d'un élément de formulaire est bien entendu disponible par la propriété JavaScript value qui correspond à l'attribut Html value des éléments de saisie ; la propriété **value** des boutons de commande correspond en Html au libellé affiché sur le bouton ; elle n'est en général exploitée par JavaScript que pour changer dynamiquement le libellé d'un bouton (ex : Start ↔Stop).

La propriété **name** reprend la valeur de l'attribut Html **name**.

Les attributs Html **disabled** et **readonly** peuvent être consultés ou modifiés par JavaScript (propriétés booléennes **disabled** et **readOnly**) de manière à activer/désactiver des zones de saisie selon le contexte.

Tous les composants affichés d'un formulaire Html disposent de méthodes JavaScript **focus**() et **blur**() pour respectivement prendre ou rejeter le curseur de saisie. De plus, les zones de saisie de texte peuvent automatiquement sélectionner leur contenu affiché grâce à la méthode **select**().

Les **listes déroulantes**, éléments Html <select>, disposent en JavaScript d'un tableau options correspondant aux différents éléments Html <option> contenus. Bien entendu la propriété **value** de l'objet **select** prend la valeur de la propriété **value** de l'objet **option** choisi par l'utilisateur.

Pour les **boutons radio** (qui portent la même valeur d'attribut **name** en Html) JavaScript permet d'accéder à chacun des boutons pour vérifier sa propriété booléenne **checked** et sa **value** éventuelle.

De même pour les **cases à cocher** Html, JavaScript donne accès aux propriétés **name, value** et **checked.**

À noter

Même si Html préconise d'écrire une valeur textuelle pour les attributs de pré-sélection par l'utilisateur (**checked='checked'** ou **selected='selected'**), JavaScript considère les propriétés **checked** et **selected** comme étant de type **boolean** (valeur **true** ou **false)**.

Enfin, un **formulaire** dispose des propriétés **action** et **method**, correspondant aux attributs Html similaires, et d'une méthode **submit**() qui permet de reproduire par programmation la soumission d'un formulaire telle que la réalise l'utilisateur en cliquant un bouton Html **type="submit**" (le chapitre 7 détaille la gestion des formulaires par JavaScript).

Pour aller plus loin...

Pour aller plus loin dans l'exploration des propriétés et méthodes des objets JavaScript, consultez le document complémentaire « **Résumé des objets JavaScript.pdf** » ainsi que la documentation de référence sur https://developer.mozilla.org/en-US/docs/Web/API/HTMLFormElement

2. L'interactivité avec les événements

2.1 Généralités

Avec les événements et surtout leur gestion par JavaScript, nous abordons le côté "magique" de JavaScript.

En Html classique, il y a des événements que vous connaissez bien : ce sont le clic de la souris sur un lien pour vous transporter sur une autre page Web et le clic d'un bouton de formulaire ; ce sont hélas les seuls événements que gère Html. Heureusement, JavaScript va permettre de gérer tout ce qui peut se passer pendant la vie d'une page Web dans un navigateur, pour votre plus grand plaisir.

Les événements JavaScript, associés aux fonctions, aux méthodes et aux formulaires, ouvrent grand la porte pour une réelle **interactivité** de vos pages.

JavaScript peut générer toutes sortes d'événements sur les objets du DOM Html comme le clic sur une image, le survol de la souris sur une div, le changement de valeur ou de focus d'un champ de saisie, l'appui sur une touche du clavier, ou encore la soumission d'un formulaire par l'utilisateur, ...

Chaque événement JavaScript sera associé à une fonction dite **asynchrone**. La fonction ne s'exécutera que lorsque l'événement sera déclenché !

2.2 Les types d'événements

Passons en revue différents événements implémentés en JavaScript.

Description	Attribut HTML
Cliquer sur un bouton, un lien ou tout autre élément du DOM.	onclick, ondblclick, onmousedown, onmouseup
Lorsqu'un élément prend ou perd le focus ou change de valeur.	onfocus, onblur, onchange
Déplacer le pointeur de la souris sur un lien ou tout autre élément.	onmouseover, onmouseout, onmousemove
Taper au clavier.	onkeyup, onkeydown, onkeypress
Charger ou quitter une page.	onload, onunload
Sélectionner un champ dans un élément de formulaire.	onselect
Envoyer un formulaire.	onsubmit
Modifier la taille de la fenêtre	onresize
En cas d'erreur (comme le chargement d'une image qui échoue).	onerror

2.3 Mettre en place des types d'événements

Vous avez 3 méthodes à votre disposition pour ajouter des événements sur des objets de votre page Html :

- Directement dans les balises Html en ajoutant l'attribut d'événement correspondant :

```
<input type="button" id="btn1" onclick="maFunctionJS();" />
```

- En JavaScript via le DOM-0 en passant par les propriétés événementielles des objets :

```
var btn = document.getElementById("btn1");
btn.onclick = function(e) { … };
```

- En JavaScript via le DOM-2, en ajoutant un handler d'événement à l'objet :

```
var btn = document.getElementById("btn1");
btn.addEventListener("click", function(e) { … }, false);
```

Remarque

Notez que l'argument baptisé ici 'e', de type event, correspond à la transmission à la fonction de la référence à l'événement déclencheur (voir plus loin).

La 1ère méthode est aujourd'hui dépréciée et à proscrire pour 2 raisons :

- Tout d'abord, l'appel à la fonction JavaScript est faite depuis le code Html, elle est donc **intrusive**. Elle ne respecte pas l'enrichissement progressif et la séparation des couches.
- De plus, le DOM-2 apporte beaucoup d'information via l'objet **event**. Hors, cet objet ne peut être créé via l'attribut HTML.

Avec la deuxième méthode, toute association d'une fonction à un événement est écrasée lorsque l'on associe de nouveau une fonction à ce même événement. Si on utilise des bibliothèques externes, on risque d'écraser un événement avec le DOM-0 !

Avec la dernière méthode, le DOM-2 permet d'associer à un même élément plusieurs fonctions liées au même type d'événement ; il suffit d'appeler successivement plusieurs fois la méthode **addEventListener().**

Le DOM-2 permet également de contrôler à quelle phase du cycle l'événement sera appelé. (via le 3ème paramètre booléen de la méthode **addEventListener()).**

Imaginez que vous ayez une image dans une **div** et que ces 2 éléments soient associés à des fonctions différentes pour le même événement « **onmouseover** ». Si vous passez votre souris sur l'image, quel événement sera déclenché en 1er ?

Quand un type d'événement est déclenché, il va se propager dans l'arbre du DOM depuis l'élément **document** jusqu'à l'élément le plus bas dans la hiérarchie du DOM (phase descendante ou dite de **capture**) puis il va remonter dans l'arbre jusqu'à l'élément document (phase montante ou dite de **bouillonnemen**t). Le dernier paramètre de la méthode **addEventListener()** permet de contrôler cela

Voir https://www.w3.org/TR/DOM-Level-3-Events/#event-flow

Remarque

La phase descendante est très peu utilisée. Les attributs Html, le DOM-0 et les anciennes versions des navigateurs ne gèrent que la phase montante (valeur "**false**").

2.4 L'objet Event

Avec la méthode **addEventListener()** DOM-2, on peut mentionner un paramètre pour la fonction appelée par l'événement. Ce paramètre sera un objet **event** qui contiendra différentes informations comme :

- le type d'événement déclencheur,
- l'élément source,
- si c'est la souris, ses coordonnées,
- si c'est le clavier, la touche frappée...

L'objet **event** permet également d'annuler les opérations liées à l'événement et d'empêcher sa propagation avec les méthodes **preventDefault()** et **stopPropagation()**.

2.5 Supprimer un événement

On peut supprimer une capture d'événement avec la méthode **removeEventListener()** :

```
var btn = document.getElementById("btn1");
btn.addEventListener("click", maFunctionJS, false);
btn.removeEventListener("click", maFunctionJS, false);
```

Glossaire

[1] **API**

> **Application Programmable Interface**, traduisez « interface de programmation » ou « interface pour l'accès programmé aux applications » ou ensemble de fonctions permettant d'accéder aux services d'une application.
>
> p. 3

[2] **W3C**

> World Wide Web Consortium. Organisme pour standardiser les différentes technologies du Web.
>
> p. 3

Crédits

OEUVRE COLLECTIVE DE L'AFPA

Sous le pilotage de la Direction de l'ingénierie

DATE DE MISE À JOUR

26/08/2021