LE LANGAGE JAVASCRIPT

TODO :

-

v1.2.0.0 – 06/05/2010
peignotc(at)arqendra(dot)net / peignotc(at)gmail(dot)com

TABLE DES MATIÈRES

TABLE DES ANNEXES

TABLE DES ILLUSTRATIONS

0 PROLOGUE

Le présent document n'a pas pour but de présenter de manière exhaustive tous les secrets du langage JavaScript, mais d'offrir un aperçu rapide du champ d'application, des possibilités, et de la syntaxe de ce langage.

Il tient pour acquis un certain nombre de concepts et d'éléments de base de la programmation et communs à différents langages. La maîtrise de notions comme les principes de « mot-clef », d'« instruction » et de « variable », les tableaux, les structures de contrôle et les fonctions, telles qu'on peut les appréhender notamment dans l'étude du langage C, C++, Java, PHP, ou C#, etc. est impérative pour comprendre les informations de ce document.

Il tient également pour acquis l'ensemble des éléments du langage HTML dans le cadre de la programmation multimédia. Là aussi, la maîtrise des fondamentaux et de la syntaxe spécifiques à ce langage (par ailleurs très différents du JavaScript) est impérative pour bien saisir l'intérêt du langage JavaScript et comment et pourquoi les deux langages s'associent afin de produire du contenu multimédia dynamique (nda : voir le cours *Le langage HTML* du même auteur).

Enfin, veuillez noter que le contenu technique du présent document date du 2nd semestre 2004 [1]. L'essor fulgurant de la planète Internet ainsi que l'avancée inéluctable des technologies informatiques rendent une partie de ces informations obsolètes. Les éléments décrits demeurent fonctionnels, mais certains sont déconseillés dans le cadre d'activités à caractère professionnel, notamment par souci de sécurité et de respect des standards.

On veillera donc à compulser d'autres documents dédiés au langage JavaScript afin de parfaire ses connaissances quant aux us et coutumes du moment du bon développeur web : évolution du HTML 4 vers le XHTML [2], association du JavaScript avec les CSS [3] afin de créer des éléments dynamiques, évolutions attendues de JavaScript 2, etc.

---

[1] La mise en page peut être plus récente. La date spécifiée en page de garde correspond à la date de dernière modification du contenu et/ou du contenant.

[2] XHTML : eXtensible HyperText Markup Language (eng) ≡ Langage de Description de l'HyperTexte Extensible (fr), successeur du HTML qui mélange les syntaxes du langage HTML et du langage XML nécessitant plus de rigueur dans son écriture.

[3] CSS : Cascading Style Sheets (eng) ≡ Feuilles de Style en Cascade (fr), définition de styles d'écriture (mise en forme, police, couleur, etc.).

1 INTRODUCTION AU LANGAGE JAVASCRIPT

Le langage **JavaScript** est un langage de script développé par la société Netscape [1]. Il s'agit d'un langage interprété, qui permet de réaliser de la programmation multimédia dynamique.

Contrairement à ce que pourrait laisser penser son nom, JavaScript n'est pas apparenté avec le langage Java. Afin d'éviter les amalgames et confusions, il vaut mieux considérer qu'il s'agit de deux langages complètement différents [2].

Les avantages de ce langage sont :

- traitement de la mise en page en partenariat avec le langage HTML ;
- intégré par défaut dans la plupart des navigateurs (Firefox, Internet Explorer, Opera, Safari, Chrome, etc.) ;
- syntaxe intégrant la notion de programmation objet (limité aux objets simples).

JavaScript est un **langage de script client**, et permet donc de réaliser des traitements au niveau du client.

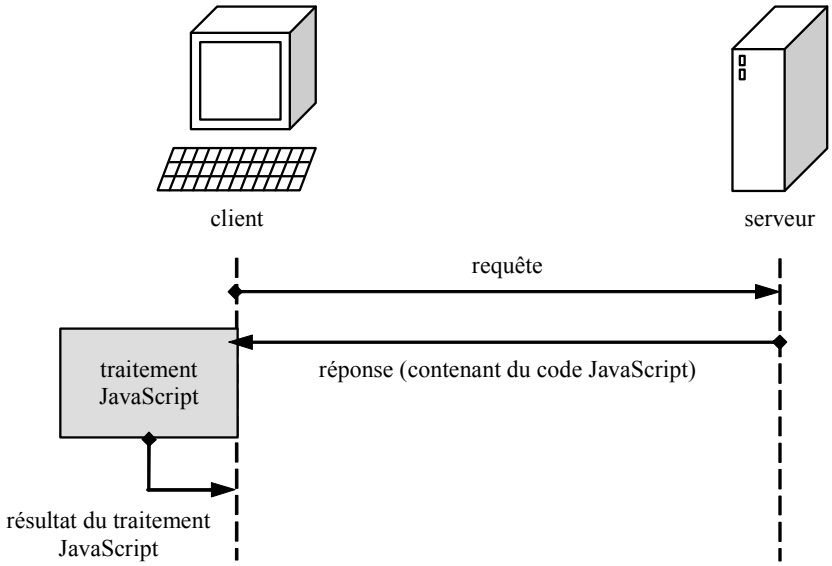

Figure 1.1 : communication client-serveur avec traitement JavaScript

Le langage JavaScript est un langage interprété, il n'a donc pas besoin d'être compilé ; en revanche, il est nécessaire de disposer d'un interpréteur JavaScript pour pouvoir exécuter des scripts JavaScript. Comme c'est la machine cliente qui se charge du traitement JavaScript, c'est elle qui doit donc disposer d'un interpréteur JavaScript [3].

Au contrario des langages de scripts qui s'exécutent au niveau du serveur, un langage de script client n'engendre pas de surcharge de traitements au niveau du serveur. Cependant, les versions de l'interpréteur JavaScript client [4] et du code JavaScript doivent correspondre [5] ; sinon, il y a un risque d'exécution erronée [6].

---

[1] Celle-là même qui a développé le navigateur web du même nom.
[2] Certes on pourrait opérer des points de rapprochement entre ces 2 langages, mais finalement pas plus qu'entre deux autres langages distincts.
[3] L'interpréteur JavaScript est constitué d'un module additionnel (plugin) à rajouter au navigateur web (généralement inclus à l'installation).
[4] En novembre 2004, nous en sommes à la version 1.5 de JavaScript.
[5] Généralement l'interpréteur JavaScript a une compatibilité ascendante et peut donc exécuter du code d'une ancienne version de JavaScript.
[6] Cas pratique : les programmeurs web confondent parfois le JavaScript (standard) avec le JScript (JavaScript spécifique à Internet Explorer (nda : politique ultra propriétaire de Micro$oft …)), dont le code n'est bien souvent que partiellement exploitable par les autres navigateurs.

2 NOTIONS DE BASE

2.1 GÉNÉRALITÉS

Généralement on écrit des scripts JavaScript en association avec le langage HTML. C'est-à-dire qu'un même fichier contiendra du code HTML ainsi que du code JavaScript.

Pour que l'interpréteur JavaScript puisse reconnaître du code JavaScript, on l'insère dans la page HTML à l'aide de la balise `<script>`.

```
<script language="javascript">
   <!-- code JavaScript -->
</script>
```

La valeur de l'attribut `language=` peut aussi préciser la version de JavaScript utilisée :

Ex. :
```
<script language="javascript1.2">
   <!-- code JavaScript -->
</script>
```

Notez que l'on insère en général le code JavaScript à l'intérieur de la balise de commentaire `<!-- -->`, elle-même insérée dans la balise `<script>` ; ceci permet aux navigateurs ne comprenant pas JavaScript de ne pas chercher à interpréter le contenu de la balise `<script>`.

On peut aussi enregistrer le code JavaScript dans un fichier portant l'extension *.js* et mentionner ce fichier avec l'attribut `src=` de la balise `<script>`. Mais un fichier contenant du code JavaScript mélangé à du HTML garde son extension *.html* (contrairement au langage PHP).

Ex. :
```
<script language="javascript" src="test.js"></script>
```

Le langage JavaScript permet d'effectuer des traitements sur les variables, mais sa finalité principale est d'améliorer la mise en page, l'ergonomie et l'interactivité de la page HTML.
On assimile le JavaScript plus à un langage de **programmation événementielle** qu'à un langage de programmation classique [1]. C'est-à-dire que l'on programme la réaction du navigateur en fonction des actions de l'utilisateur, lesquelles constituent des **événements**.

Ex. :
- modification du menu lors du « survol » de celui-ci par le curseur de souris ;
- affichage d'un nouvel élément visuel lors d'un clic de souris sur un bouton ;
- gestion de menus déroulants ;
- …

Pour procéder à la gestion des ces événements, on a rajouté de nouveaux attributs aux balises. Tous ces attributs commencent par le préfixe `on-` en suivant la syntaxe `onEvénement=` (ex. : `onMouseOver=`, `onClick=`, etc.).

[1] Langage procédural.

Ex. : La page HTML va se fermer dès que le curseur de souris sera positionné « au-dessus » du texte *test JavaScript*.

```
<html>
   <head>
      <script language="javascript">
         function fin() {
            window.close()
         }
      </script>
   </head>
   <body>
      <font size=4>
         <a href="" onMouseOver="fin()">test JavaScript</a>
      </font>
   </body>
</html>
```

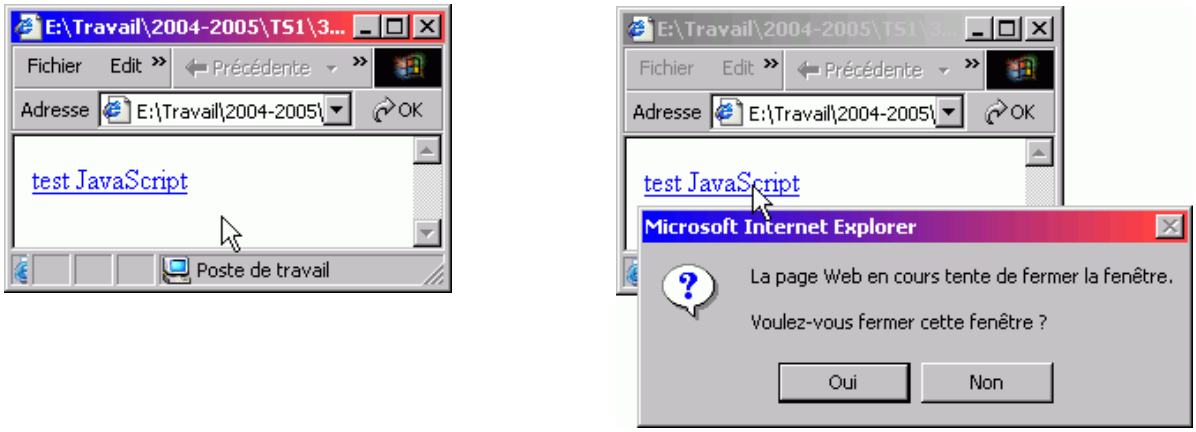

Figure 2.1 : exemple de gestion d'un événement JavaScript avec la balise <a> (1)

L'utilisation de la balise `<a>` n'est prétexte ici qu'à pouvoir utiliser un attribut JavaScript, en l'occurrence `onMouseOver=`. En effet l'attribut `href=` n'a pas de valeur définie (`href=""`) ; cependant le fait de préciser cet attribut, même vide, permet d'afficher *test JavaScript* comme un lien.

Si on ne précise pas l'attribut `href=`, le texte s'affiche alors de manière simple ; s'il est toujours possible de cliquer dessus et d'exécuter le code JavaScript associé, aucune information visuelle ne permet à l'utilisateur d'identifier que l'élément propose ce comportement.

Ex. : `<a onMouseOver="fin()">test JavaScript</a>`

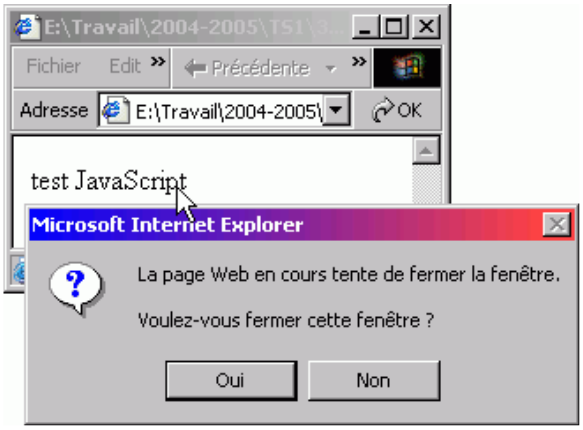

Figure 2.2 : exemple de gestion d'un événement JavaScript avec la balise <a> (2)

Nous pouvons remarquer que les fonctions JavaScript sont généralement écrites dans la zone d'en-tête du document HTML [1].

[1] À l'intérieur de la balise `<head>`.

Dans l'exemple ci-dessus, on aurait pu se passer de l'écriture d'une fonction en écrivant directement :

```
<html>
   <head></head>
   <body>
      <a href="" onMouseOver="window.close()">test JavaScript</a>
   </body>
</html>
```

On peut aussi utiliser les instructions JavaScript en tant que pseudo-protocole.

Ex. : On se trouve alors dans le cas du fonctionnement classique de la balise `<a>` (le lien est donc souligné et de couleur).

```
<body>
   <a href="javascript:window.close()">test JavaScript</a>
</body>
```

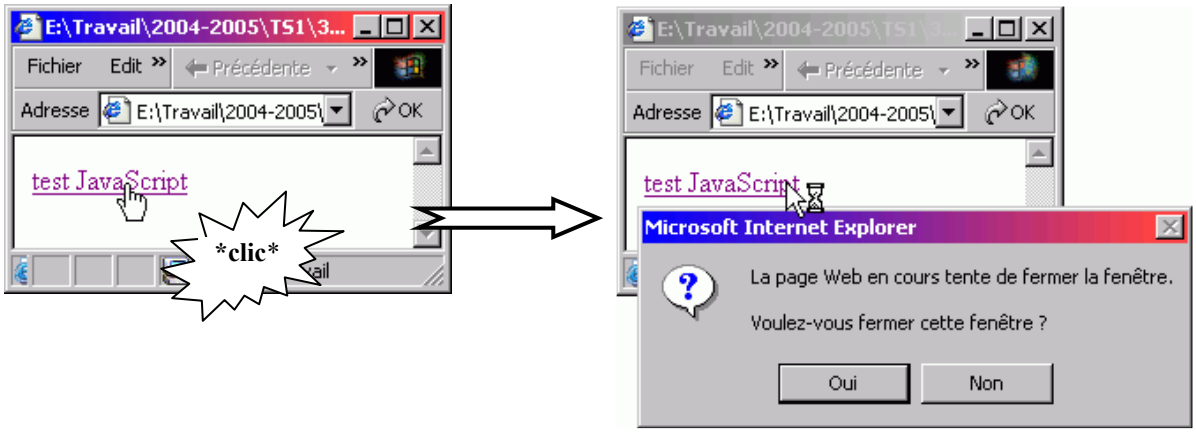

Figure 2.3 : exemple de gestion d'un événement JavaScript avec la balise <a> (3)

Le langage JavaScript a une syntaxe qui se rapproche des langages C/C++ :

- le caractère de fin de ligne de code '`;`' (point-virgule) n'est pas obligatoire [1] ;
- les niveaux d'imbrication de boucle sont indiquées avec les caractères '`{`' et '`}`' (accolades) ;
- les expressions sont évaluées en étant encadrées par les caractères '`(`' et '`)`' (parenthèses) ;
- les opérateurs (opération, affectation, comparaison) ont été repris du langage C ;
- on insère des commentaires avec les caractères '`//`' (une seule ligne) ou bien `/* */` (plusieurs lignes) ;

Il faut cependant noter que JavaScript différencie la casse [2]. Il convient donc d'être extrêmement rigoureux dans ses écritures de scripts JavaScript.

2.2 LES VARIABLES

En JavaScript, le type d'une variable n'est pas explicité. En revanche, il est conseillé de déclarer la variable [3] avec le mot-clef `var`.

Ex. :
```
var A;
var B="test";
```

Le type de la variable est donc fixé en fonction de son contenu, et est donc susceptible d'évoluer au cours de l'exécution du script.

Lorsque la variable est déclarée, elle a le même genre de portée qu'en langage de programmation procédural ; elle est donc locale [4]. Lorsque la variable n'est pas déclarée, elle est globale.

[1] Mais il est fortement conseillé par souci de lisibilité.
[2] Casse = minuscule ou majuscule.
[3] Mais ce n'est pas obligatoire.
[4] La portée de la variable est limitée au bloc `{ }` dans lequel elle a été définie (fonctions, ou bloc d'instructions dans une structure de contrôle).

2.2.1 Les types de variables

Les types disponibles en JavaScript sont classiques :
- `boolean` : booléen, ex. : `var isOK = true ;`
- `number` : nombre entier ou réel, ex. : `var nb = 15 ;`
- `string` : chaîne de caractère, ex. : `var nom = "Jean" ;`
- `Array` : tableau, liste de valeur, ex. : `var tab = new array(12,4,2) ;`
- `objet` : objet (au sens programmation orientée objet).

2.2.2 Les booléens

Ils ne peuvent prendre que 2 valeurs : `true` ou `false`.

2.2.3 Les variables numériques

Les opérateurs pour les variables numériques sont les opérateurs classiques :
- opérateurs arithmétiques : `+`, `-`, `*`, `/`, `%` ;
- opérateurs d'incrémentation/décrémentation (pré– et post–) : `++`, `--` ;
- opérateurs combinés : `+=`, `-=`, `*=`, `/=`, `%=`, `&=`, `|=`, `^=`, `<<=`, `>>=` ;
- opérateurs logiques : `&` (ET), `|` (OU), `^` (XOR), `<<` (décalage à gauche), `>>` (décalage à droite).

2.2.4 Les chaînes de caractères

Les chaînes peuvent être délimitées aussi bien par le caractère '`'`' (apostrophe [1]) que par le caractère '`"`' (guillemets [2]).

On peut concaténer les chaînes facilement avec l'opérateur '`+`' (plus).

Les caractères spéciaux connus du langage C peuvent aussi être utilisés : `\r` (retour chariot), `\n` (saut de ligne), `\t` (tabulation), `\b` (retour arrière), `\f` (saut de page).

Le caractère `\` est le caractère d'échappement [3] et permet par exemple d'insérer dans une chaîne le caractère '`'`' (en tant qu'apostrophe et non en tant que délimiteur de chaîne).

Ex. :
```
var str = 'en écrivant \\\', le caractère cote : \' s\'affiche';
```

2.2.5 Les tableaux

On peut définir des tableaux grâce au mot-clef `Array`. Plus qu'un mot-clef, il s'agit en fait d'un objet. Pour définir un tableau, on effectue donc une instanciation de la classe `Array` (avec le mot-clef `new` par conséquent).

Ex. :
```
var tab1 = new Array();
var tab2 = new Array('Jean','Michel');
var tab3 = new Array(2);
tab3[0] = 'Jean';
```

Les tableaux sont dynamiques, c'est-à-dire que la taille s'adapte automatiquement en fonction du nombre de valeurs enregistrées.

Comme en PHP, on peut aussi définir des tableaux associatifs, ainsi que des tableaux multi-dimensions.

2.3 LES STRUCTURES DE CONTRÔLE

Elles sont identiques au langage C.

[1] Appelée aussi *cote* ou *simple cote*, en opposition à l'appellation *double cote* qui correspond tout simplement aux guillemets.
[2] Très utile pour insérer une chaîne dans une instruction JavaScript elle-même insérée dans un attribut HTML (entre guillemets donc).
[3] Appelé *quote* en anglais (to quote (eng) ≡ citer (fr)).

2.3.1 Les opérateurs

- opérateurs de tests : ==, !=, >, <, >=, <= ;
- opérateurs logiques : `AND` ou `&&`, `OR` ou `||`, `NOT` ou `!`, `XOR` ou `^^`.

Le résultat obtenu est un booléen.

2.3.2 Les tests

2.3.2.1 Test simple

On réalise un test simple avec l'instruction `if ()` éventuellement associée avec l'instruction `else`, suivant la syntaxe :

```
if (condition) {
   /* instructions si condition validée */
}
else {
   /* instructions si condition invalidée */
}
```

2.3.2.2 Test multiple

On réalise un test multiple avec les instructions `switch` et `case` éventuellement associées avec l'instruction `default`, suivant la syntaxe :

```
switch (variable) {
   case valeur1 :  /* instructions si variable vaut valeur1 */
           break
   case valeur2 :  /* instructions si variable vaut valeur2 */
           break
   ...
   default :  /* instructions si variable ne vaut aucune des valeurs */
}
```

2.3.3 Les boucles

2.3.3.1 Boucle « tant que… faire… »

On réalise une boucle « tant que » (répétition avec aucune exécution au minimum) avec l'instruction `while ()`, suivant la syntaxe :

```
while (condition) {
   /* instructions */
}
```

2.3.3.2 Boucle « répéter… tant que »

On réalise une boucle « répéter tant que » (répétition avec aucune exécution au minimum) avec les instructions `do` et `while ()`, suivant la syntaxe :

```
do {
   /* instructions */
}
while (condition);
```

2.3.3.3 Boucle « pour… faire »

On réalise une boucle « pour » (répétition contrôlée) avec l'instruction `for ()`, suivant la syntaxe :

```
for (initialisation ; condition ; incrémentation ou décrémentation) {
   /* instructions */
}
```

2.4 LES FONCTIONS

Comme déjà mentionné, les fonctions JavaScript sont généralement écrites dans la zone d'en-tête du document HTML. En effet, cette zone du document est chargée avant toute autre ; ainsi toutes les fonctions JavaScript sont chargées avant leur utilisation.

La structure de définition d'une fonction est classique, si ce n'est que le type de retour n'est pas mentionné :

```
function nom_fonction(paramètre1, paramètre2, ...)
{
   var valeur_retour;

   /* instructions */

   return valeur_retour;
}
```

Nb : Une fonction peut ne renvoyer aucune valeur de retour, tout comme elle peut ne pas avoir de paramètres d'en entrée.

3 GESTION DES ÉVÉNEMENTS

En général, on considère JavaScript comme un langage de **programmation événementielle**. C'est-à-dire que l'on programme la réaction du navigateur en fonction des actions de l'utilisateur :

- un clic ;
- position spécifique du curseur de souris (au-dessous ou en-dehors d'une zone du document HTML) ;
- activation d'une touche du clavier ;
- …

Ces actions constituent ce qu'on appelle des **événements** [1].

Pour procéder à la gestion des ces événements, on a rajouté de nouveaux attributs [2] aux balises. Tous ces attributs commencent par le préfixe `on-` en suivant la syntaxe `onEvénement=` (ex. : `onMouseOver=`, `onClick=`, etc.).

Les événements sont généralement associés à des éléments graphiques, lesquels sont donc affichés à l'écran. On peut classifier ces événements suivant le type d'objets graphiques auquel ils sont associés :

- fenêtre ;
- souris ;
- clavier ;
- formulaire.

3.1 ÉVÉNEMENTS ASSOCIÉS AUX FENÊTRES

- `onLoad=` : déclenché une fois l'intégralité des éléments de la page téléchargés ;
- `onUnload=` : déclenché lorsque l'on quitte la page web ;
- `onResize=` : déclenché lorsque la fenêtre est redimensionnée ;
- `onMove=` : déclenché lorsque la fenêtre est déplacée ;
- `onAbort=` : déclenché lorsque le téléchargement d'une image est annulée ;
- `onError=` : déclenché lorsqu'une erreur JavaScript se produit ;
- `onFocus=` : déclenché lorsque la fenêtre passe au premier plan ou devient active ;
- `onBlur=` : déclenché lorsque la fenêtre passe en arrière-plan ou devient inactive.

3.2 ÉVÉNEMENTS ASSOCIÉS À LA SOURIS

- `onMouseDown=` : déclenché lorsqu'un bouton de la souris est enfoncé ;
- `onMouseUp=` : déclenché lorsqu'un bouton de la souris est relâché (nda : il était de fait enfoncé avant) ;
- `onClick=` : déclenché lorsque l'utilisateur clique sur l'objet (bouton enfoncé puis relâché) ;
- `onDblClick=` : déclenché lorsque l'utilisateur double-clique sur l'objet ;
- `onMouseMove=` : déclenché lorsque la souris est déplacée ;
- `onMouseOver=` : déclenché lorsque le curseur de souris se positionne au-dessus de l'objet ;
- `onMouseOut=` : déclenché lorsque le curseur de souris se positionne hors de la zone d'un objet.

[1] Toutes les actions de l'utilisateur sont des événements, en revanche la réciproque est fausse : en effet, par exemple, lorsqu'une erreur survient dans l'exécution d'un script JavaScript, un événement est déclenché ; or il ne s'agit pas d'une action de l'utilisateur (action directe s'entend).

[2] Appelés alors *attributs JavaScript*.

3.3 ÉVÉNEMENTS ASSOCIÉS AU CLAVIER

- `onKeyDown=` : déclenché lorsqu'une touche est enfoncée ;
- `onKeyUp=` : déclenché lorsqu'une touche est relâchée ;
- `onKeyPress=` : déclenché lorsqu'une touche est appuyée (enfoncée puis relâchée).

3.4 ÉVÉNEMENTS ASSOCIÉS AUX FORMULAIRES

- `onSubmit=` : déclenché lorsque le formulaire est validé (clic sur le bouton de validation) ;
- `onReset=` : déclenché lorsque le formulaire est ré-initialisé (clic sur le bouton reset) ;
- `onChange=` : déclenché lorsque la valeur d'un champ du formulaire est modifié ;
- `onSelect=` : déclenché lorsque du texte est sélectionné dans un champ (types `<input>` et `<textarea>`) ;
- `onClick=` : déclenché lorsque l'utilisateur clique sur un champ (types `<input type="radio">` et `<input type="checkbox">`) ;
- `onFocus=` : déclenché lorsque le curseur de souris se positionne au-dessus d'un champ ;
- `onBlur=` : déclenché lorsque le curseur de souris se positionne hors de la zone d'un champ.

4 LES OBJETS

Le langage JavaScript intègre et utilise la notion d'objet que l'on retrouve dans la programmation objet (C++, Java, etc.). Cependant on ne manipule que des objets « simples » ; la notion de classe est simpliste et le concept d'héritage n'est pas implémenté.

4.1 NOTIONS D'OBJET EN JAVASCRIPT

En JavaScript, on manipule donc des variables, mais aussi des objets. Un objet possède des variables associées et des fonctions associées ; en terminologie objet JavaScript, on parle respectivement de **propriétés** [1] (variables) et **méthodes** (fonctions).

Le caractère '`.`' (point) est utilisé pour marquer la relation de type objet.

Ex. : Soit la classe `window`, correspondant à une zone d'affichage du navigateur.
La méthode `close()` ferme la fenêtre.
La propriété `innerHeight` correspond à la hauteur de la fenêtre.
La syntaxe `window.close()` permet donc d'exécuter la méthode `close()` de la classe `window`, et `window.innerHeight` permet d'accéder à la variable `innerHeight` de la classe `window`.

`window.close()` ferme la fenêtre du navigateur ;
`window.innerHeight = 200` modifie la hauteur de la fenêtre à 200 pixels.

Une propriété d'un objet peut être un autre objet lui-même, possédant donc des propriétés et des méthodes.

Ex. : Soit la classe `document`, correspondant à un document HTML.
Définissons-en un objet comme propriété de la classe `window` : `window.document` correspond au document HTML chargé dans la fenêtre du navigateur.
La propriété `bgColor` de la classe `document` correspond à la couleur de fond du document HTML.
La méthode `write(texte)` de la classe `document` permet d'écrire le texte dans le document HTML.

`window.document.bgColor = 'blue'` modifie en bleu la couleur de fond du document HTML chargé dans la fenêtre du navigateur ;
`window.document.write('modification du texte')` affiche le texte indiqué dans le document HTML chargé dans la fenêtre du navigateur.

objet.javascript.1.htm

```
<html>
   <head>
      <script language="javascript">
         function change()
         {
            nouvwin = window.open('objet.javascript.2.htm');
            nouvwin.document.bgColor = 'blue';
            nouvwin.document.fgColor = 'white';
            window.document.write('modification du texte');
         }
      </script>
   </head>
```

[1] Appelées *attributs* en POO classique. Dans les langages comme Java et C# la notion de propriété est une extension de la notion d'attribut.

```
   <body>
      <h1 align="center"><a onMouseOver="change()">test JavaScript 2</a></h1>
   </body>
</html>
```

objet. javascript.2.htm

```
<html>
   <body>
      <font size=4>
         <a onMouseOver="javascript:window.close()">texte affiché en blanc
         <br>sur fond bleu</a>
      </font>
   </body>
</html>
```

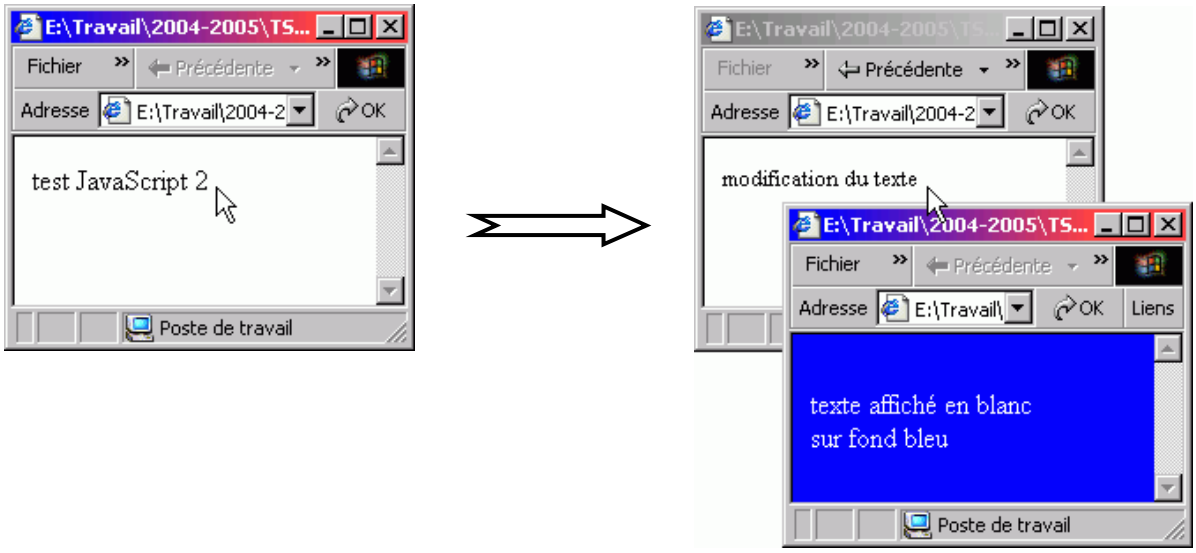

Figure 4.1 : exemple d'utilisation des objets en JavaScript

4.2 LES OBJETS DU NAVIGATEUR

Lorsqu'on lance un navigateur, JavaScript crée [1] automatiquement un certain nombre d'objets appartenant à l'une des 3 classes suivantes :

- `window` : toute zone d'affichage HTML : 1 objet par fenêtre du navigateur ou frame ;
- `navigator` : navigateur utilisé : 1 seul objet possible ;
- `screen` : caractéristiques d'affichage de la machine utilisée : 1 seul objet possible.

Les objets de la classe `window` sont les plus utilisés parmi les 3 types d'objets créés automatiquement.

4.2.1 La classe window

Lorsqu'on n'utilise pas de frames, on dispose d'un objet de la classe `window` [2] par fenêtre du navigateur. Dans le document HTML chargé dans l'une de ses fenêtres, la référence `window` correspond donc à la fenêtre qui contient ce document.

Dans le cas de l'utilisation de frames, on dispose d'un objet de la classe `window` par fenêtre du navigateur, d'un objet de la classe `window` par cadre, ainsi que d'objets spécifiques :

- `window` : zone d'affichage courante, soit donc le cadre courant [3] ;
- `nom_cadre` : cadre dont le nom est `nom_cadre` (affecté par l'attribut `name=` de la balise `<frame>`) ;
- `parent` : zone d'affichage « parente » (découpée donc en cadres (dont le cadre courant)) ;
- `top` : zone d'affichage « principale » (le cadre le plus haut dans la hiérarchie, soit donc le cadre correspondant à la fenêtre du navigateur).

[1] En terminologie objet, le terme exact est *instancie* (on réalise une *instanciation*).
[2] Correspondant à la zone d'affichage courante du navigateur, soit donc l'intégralité de la zone couverte par l'une de ses fenêtres.
[3] Implicite ; de fait il est souvent omis, et on écrit alors par exemple `document.write()` à la place de `window.document.write()`.

Tous ces objets peuvent alors être appelés suivant la hiérarchie qu'ils mettent en œuvre.

Ex. : Soit le découpage en cadres suivant :

fenêtre du navigateur (cadre principal)
- cadre1
- cadre2
  - cadre21
  - cadre22

En JavaScript, cette hiérarchie s'écrit alors :

```
window.top
   ├ window.top.cadre1
   └ window.top.cadre2
      ├ window.top.cadre2.cadre21
      └ window.top.cadre2.cadre22
```

Considérons que le cadre courant est cadre2, il existe plusieurs manières de faire référence aux différents cadres, et d'utiliser leurs propriétés et méthodes (en l'occurrence, la propriété `document` [1]) :

- cadre2 (cadre courant) :
  `document` ⇔ `window.document` ⇔ `window.top.cadre2`
- cadre parent (identique au cadre principal dans cet exemple [2]) :
  `window.parent` ⇔ `window.top`
- cadre1 :
  `window.parent.cadre1` ⇔ `window.top.cadre1`

Toutes les syntaxes sont possibles [3], à partir du moment où la hiérarchie des cadres est respectée.

Ex. :

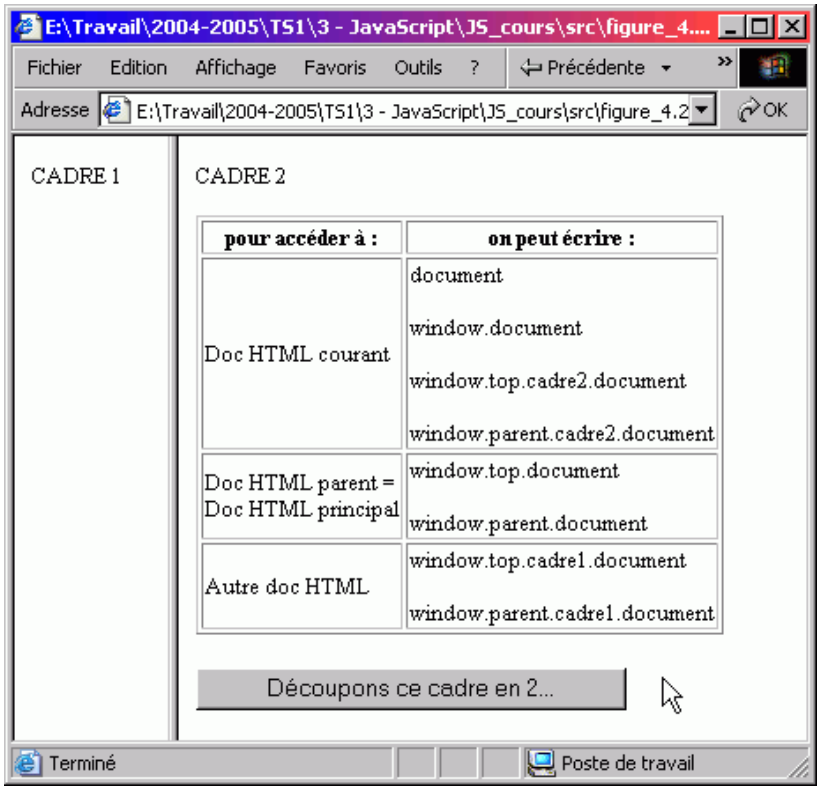

Figure 4.2 : exemple des différentes syntaxes d'accès aux cadres d'une page utilisant les frames (1)

[1] Qui est par ailleurs un objet (si vous suivez bien…).
[2] Ce qui est toujours le cas lorsqu'un seul découpage en cadres a été opéré dans le document HTML.
[3] On peut écrire par exemple (`cadre2` est le cadre courant) : `window.parent.cadre2.parent.cadre1` pour accéder au `cadre1` (sic).

```
<html>
   <body>
      CADRE 2<br><br>
      <table border="1"><tr>
         <th>pour accéder à :</th>
         <th>on peut écrire :</th>
      </tr><tr>
         <td>Doc HTML courant</td>
         <td>
         <a onClick="document.write('texte')">
         document</a><br><br>
         <a onClick="window.document.write('texte')">
         window.document</a><br><br>
         <a onClick="window.top.cadre2.document.write('texte')">
         window.top.cadre2.document</a><br><br>
         <a onClick="window.parent.cadre2.document.write('texte')">
         window.parent.cadre2.document</a>
         </td>
      </tr><tr>
         <td>Doc HTML parent =<br>Doc HTML principal</td>
         <td>
         <a onClick="window.top.document.write('texte')">
         window.top.document</a><br><br>
         <a onClick="window.parent.document.write('texte')">
         window.parent.document</a>
         </td>
      </tr><tr>
         <td>Autre doc HTML</td>
         <td>
         <a onClick="window.top.cadre1.document.write('texte')">
         window.top.cadre1.document</a><br><br>
         <a onClick="window.parent.cadre1.document.write('texte')">
         window.parent.cadre1.document</a>
         </td>
      </tr></table><br>
      <form action="figure_4.2_classe.window.20-2.htm">
         <input type="submit" value="Découpons ce cadre en 2...">
      </form>
   </body>
</html>
```

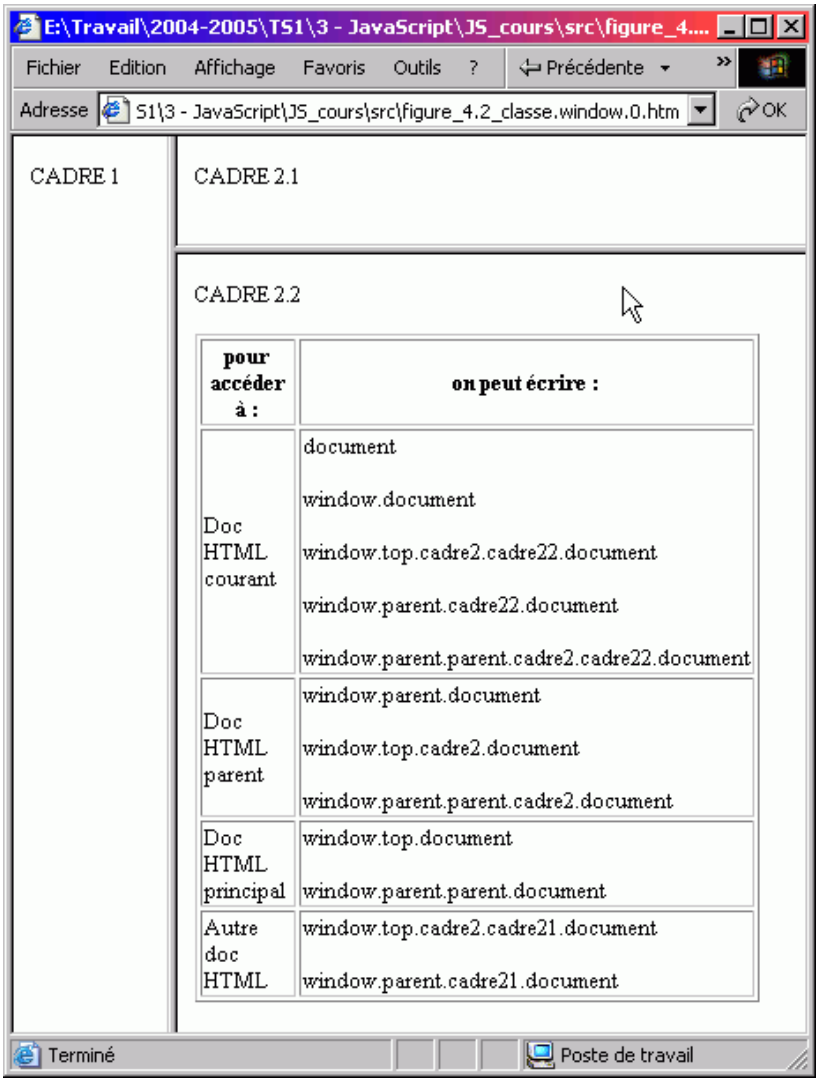

Figure 4.3 : exemple des différentes syntaxes d'accès aux cadres d'une page utilisant les frames (2)

4.2.2 Autres

Au-delà des objets `navigator`, `screen`, ainsi que les différents objets `window` pour toute fenêtre ou tout cadre du navigateur, tous les éléments d'une page sont accessibles comme des objets ; en effet JavaScript se combine avec le DHTML [1] qui permet une vue orientée objet de la page web ainsi que de tous ses éléments.

Ainsi chaque élément d'une page (titre, paragraphe, image, formulaire, etc.) a un objet associé, possédant des propriétés, et éventuellement des méthodes. Cette représentation objet a été standardisée [2], et on s'y réfère sous le nom de DOM [3].

En plus du DOM, le DHTML (très peu développé ici) introduit aussi la notion de *feuille de style* [4], qui permet de définir des styles d'écriture (incluant la mise en forme, le type de police, les couleurs utilisées, etc.) que l'on peut à loisir appliquer et modifier.

[1] DHTML : Dynamic HTML (eng) ≡ HTML Dynamique (fr).
[2] C'est le consortium W3C, organisme international, qui a défini ce standard du DHTML, qui est, en règle générale, relativement bien suivi par les développeurs de navigateurs.
[3] DOM : Document Object Model (eng) ≡ Modèle de Document Objet (fr).
[4] Appelées CSS : Cascading Style Sheets (eng) ≡ Feuilles de Style en Cascade (fr).

A BIBLIOGRAPHIE

Wazir Dino, *Cours JavaScript*, TS IRIS – LEGT Louis-Modeste Leroy – Évreux, 2002 ;

CommentÇaMarche.net, http://www.commentcamarche.net/, 2004 ;

Wikipedia – l'encyclopédie libre, http://fr.wikipedia.org/, 2005 ;

Peignot Christophe, *Langage HTML* et *Langage C*, http://info.arqendra.net/, 2010.