

Introduction Système et Réseaux - FLIN302

Michel Meynard

22 octobre 2008

Table des matières

1	Introduction	5
1.1	Objectifs	5
1.2	Finalités	5
1.3	HTTP et architecture du Web	5
1.3.1	Côté Serveur	6
1.3.2	Côté Client	6
2	Le langage HTML et XHTML	7
2.1	Introduction	7
2.1.1	Vocabulaire	7
2.2	Syntaxe de XHTML	7
2.2.1	Généralités	7
2.2.2	Définition de la DTD	8
2.2.3	L'élément html	8
2.2.4	L'en-tête du document	8
2.3	Le corps du document	8
2.3.1	Attributs de base	9
2.3.2	Éléments indispensables	9
2.4	Les formulaires	10
2.4.1	Éléments de formulaires	10
2.4.2	Inclusion d'autres objets	11
3	JavaScript	13
3.1	Modèle événementiel	13
3.2	Variables	13
3.3	Types de données	13
3.3.1	Nombres (objet Number)	14
3.3.2	Chaînes de caractères (objet String)	14
3.3.3	Tableaux (objet Array)	14
3.3.4	Objets (objet Object)	15
3.3.5	Booléen (objet Boolean)	16
3.4	Fonctions (objet Function)	16
3.5	Opérateurs	16
3.6	Structures de contrôle	16
3.6.1	Alternatives	16
3.6.2	Itératives	17
3.6.3	Parcours des propriétés (et/ou des indices) d'un objet	17
3.6.4	Utilisation d'objet implicite (with)	17
3.6.5	Exceptions	17
3.7	Divers	18
4	Le Document Object Model (DOM 2) et JavaScript	19
4.1	Introduction	19
4.2	Interface du Document HTML	19
4.3	document	20
4.4	Window	21
4.5	Événements	21
4.5.1	Gestionnaire d'événement	21

4.5.2	L'objet Event	22
4.6	Formulaire	22
4.7	Un exemple d'arbre DOM	23
5	Les feuilles de styles CSS	27
5.1	Introduction	27
5.2	Objectifs	27
5.3	Syntaxe	27
5.4	Quelques exemples simples	28
5.5	Localisation des styles	29
5.6	Sélecteur	29
5.6.1	Correspondance de modèle (pattern matching)	29
5.6.2	Sélecteur multiple <input type="checkbox"/>	30
5.6.3	classes de style <input type="checkbox"/>	30
5.7	Modèle de visualisation	31
6	HTTP et Application côté serveur	33
6.1	Apache	33
6.1.1	Configuration	33
6.1.2	Débogage	33
6.1.3	Authentification Apache	33
6.1.4	Création des mots de passe	34
6.1.5	Désauthentification	34
6.2	Paramètres de RezUFR	34
7	Conclusion	35
8	Erreurs fréquentes	37
8.1	Javascript	37
8.2	DOM	37

Chapitre 1

Introduction

1.1 Objectifs

Ce cours est destiné à des étudiants néophytes en technologies du Web mais ayant des connaissances de base en programmation et une expérience utilisateur de la navigation sur le web. Son ambition est d'inculquer les bases du langage HTML ainsi que du protocole HTTP. Les notions de serveur HTTP, par exemple Apache, et de client léger, navigateur Web, ainsi qu'un minimum de programmation en Javascript seront acquises. L'ouvrage [1] est indiqué pour s'initier à ces techniques de base.

L'étudiant se familiarisera avec la conception de sites statiques à l'aide de l'outil de base "emacs". En TP, il devra être capable de rechercher des références et de la documentation concernant HTML, Javascript, les feuilles de style CSS.

1.2 Finalités

Les objectifs de ce cours sont nombreux :

- apprentissage du langage HTML ;
- séparation contenu et forme ;
- apprentissage des feuilles de style CSS ;
- apprentissage d'un langage de programmation côté client javascript ;
- étude de la dynamique des applications Web : initialisation, saisie, traitement, ...
- étude de la dynamique côté client avec le langage JavaScript ;

1.3 HTTP et architecture du Web

Quelques définitions de base sont nécessaires :

HTTP **H**yper**T**ext **T**ransfer **P**rotocol est un protocole de communication **client-serveur** développé pour le World Wide Web (www). HTTP a été inventé par Tim Berners-Lee avec les adresses web (URL) et le langage HTML pour créer le World Wide Web. Il utilise généralement le port 80 et est un protocole **non orienté connexion**. Schématiquement, le client envoie une requête au serveur qui lui retourne une réponse généralement sous la forme d'un fichier HTML. Le client affiche alors la réponse. Cette réponse contient généralement des **liens** hypertextes ou des formulaires, qui une fois cliqués ou soumis génèrent une requête au serveur ...

serveur HTTP par exemple, Apache, IIS, ... sont des serveurs installés sur des machines.

client HTTP appelé aussi client léger ou navigateur Web, les plus connus sont Firefox, Internet Explorer, Opera, ...

URL de l'anglais *Uniform Resource Locator* est une chaîne de caractères codée en ASCII pour adresser les ressources du World Wide Web : document HTML, image, son, forum Usenet, boîte aux lettres électroniques, etc. Elle est informellement appelée une **adresse web**. Exemples : <http://www.google.fr/>, <mailto:toto@titi.fr>, http://www.lirmm.fr/~meynard/Ens2/rubrique.php3?id_rubrique=36&x=1, ...

URL relative à l'intérieur d'une page HTML, des liens vers d'autres pages du même site peuvent être définis avec une écriture relative au répertoire courant de la page : par exemple, [../images/toto.jpg](#) est une URL relative.

Ce qu'il faut retenir absolument, c'est que HTTP n'est pas orienté connexion c'est-à-dire que le serveur ne mémorise aucune information à propos du client et qu'entre 2 requêtes du client, il peut se passer 2 secondes ou un temps infini ! Par conséquent, l'écriture des applications côté serveur doit prendre en compte cette absence de mémoire.

D'autre part, une part non négligeable du traitement dans une application peut être effectué côté client afin d'effectuer des vérifications (contrôles) qui éviteront de surcharger le serveur. Cependant, des règles de sécurité interdisent aux scripts côté client d'accéder aux ressources du poste client (fichiers, imprimante, ...).

1.3.1 Côté Serveur

L'application côté serveur utilise deux technologies possibles :

cgi le programme est un script (python, bash) ou un binaire (compilé par g++) qui est chargé dans un processus externe au serveur http. La sortie standard de ce processus est redirigé dans la réponse envoyée par le serveur au client. L'inconvénient principal des cgi est la perte de temps nécessitée par la création d'un processus pour chaque nouvelle requête;

module certains serveurs dont Apache ont intégré des modules interprétant des langages de programmation tels que Perl, Python, PHP. Ainsi, l'interprétation des scripts est beaucoup plus rapide.

1.3.2 Côté Client

L'application côté client peut utiliser plusieurs technologies possibles :

javascript langage de script interprété par le navigateur et permettant de manipuler l'arborescence du document(DOM);

applet Java mini application Java permettant d'utiliser toute la puissance du langage Java (API, structures de données, ...)

ActionScript langage de script compatible avec JavaScript et permettant de réaliser des animations Flash ...

Chapitre 2

Le langage HTML et XHTML

2.1 Introduction

“HyperText Markup Language”, abrégé HTML, est le langage conçu pour représenter les pages web. En août 1991, Tim Berners-Lee (CERN) annonce publiquement le web sur Usenet, en donnant l’URL d’un document de suffixe .html. A l’origine, HTML est basé sur SGML jugé trop complexe. “eXtensible HyperText Markup Language”, abrégé XHTML, est le langage destiné à remplacer HTML 4.0 et est une application XML. Par la suite, nous utiliserons toujours du XHTML.

A l’origine, contenu et format de document étaient mélangés dans un même fichier HTML. C’est-à-dire que l’on peut trouver dans un document HTML, des éléments de :

- structuration du contenu tels que `div`, `h1`, `h2`, ..
- mise en forme tels que `b` (bold), `i` (italic), `center`, ...

Actuellement, on utilise des feuilles de style (en cascade) “Cascading Style Sheets” pour le format. La plupart du temps, la feuille de style est externe c’est-à-dire stockée dans un fichier différent référencé (lié) par le fichier HTML. Dans certains cas, un style en-ligne (au fichier) peut être associé.

Actuellement, c’est le W3C (*World Wide Web Consortium*) qui est chargé de rédiger des recommandations (sorte de norme) concernant les technologies du web. L’adresse web de leur site est <http://www.w3.org/>. On peut notamment valider ses documents en ligne à l’adresse <http://validator.w3.org/>. Un autre site d’importance et qui n’a aucun lien avec le W3C est <http://www.w3schools.com/> qui est un site pédagogique très pratique même s’il est commercial.

2.1.1 Vocabulaire

- document : l’ensemble du texte composé du fichier principal et des fichiers inclus ou référencés (script, feuille de style, image, ...);
- langage à balisage : `<balise>contenu ...</balise>`;
- élément : composé d’une balise ouvrante puis d’un contenu (possédant éventuellement d’autres éléments) puis de la balise fermante correspondante;
- élément vide : ne possédant pas de contenu, la balise ouvrante est également fermante comme dans `
`;
- attribut : information de la forme `nom='valeur'` présente uniquement dans une balise ouvrante;
- validité : un document HTML est valide s’il correspond aux règles édicté par le “World Wide Web Consortium” (w3c);
- URI : *Uniform Resource Identifier* adresse d’une ressource Internet composé d’un protocole, d’un nom de domaine complètement qualifié (FQDN), d’un chemin, et d’autres paramètres possibles; Par exemple, `http://www.lirmm.fr/~meynard/ProjetInfoL3/index.php?rubrique=Projet&action=liste` est une URI. Autre exemple, `mailto:toto@tutu.fr` désigne une adresse email;

2.2 Syntaxe de XHTML

2.2.1 Généralités

- les noms d’attribut sont en minuscules;
- les valeurs d’attribut doivent être entre apostrophes ou bien entre guillemets;
- les éléments vides sont définis par une seule balise, comme dans `
` et dans ``;
- la déclaration d’une DTD est obligatoire;

Exemple 1 (Exemple de document XHTML minimum : modele.html)

```

<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<meta http-equiv="content-type" content="text/html; charset=ISO-8859-1" />

<title>mon premier document</title>
</head>
<body>
<h1>Titre de niveau 1</h1>
<p>hello World !</p>
</body>
</html>

```

2.2.2 Définition de la DTD

Une DTD (Document Type Definition) définit la syntaxe d'un type de document : c'est une grammaire formelle indiquant les imbrications possibles d'éléments, les attributs obligatoires, optionnels, ...

La déclaration de la DTD `<!DOCTYPE ...` est obligatoire afin que le navigateur sache quel type de document gérer. Cette déclaration localise le fichier de DTD. Remarquons que plusieurs DTD pour XHTML 1 existent (strict, transitional, frameset), et que nous choisissons la transitoire qui permet certains éléments de présentation (center, applet, ...).

2.2.3 L'élément html

C'est la racine du document XML et doit par conséquent définir l'espace de nom XML grâce à l'attribut `xmlns`. Il doit contenir à sa suite 2 éléments :

head l'en-tête du document ;

body le corps du document qui sera affiché sur la page ;

2.2.4 L'en-tête du document

L'élément `head` peut et **doit** contenir d'autres éléments que le titre du document `title` :

- l'élément `meta` fournit des méta-informations sur la page en cours qui pourront être utilisées par les moteurs de recherche ou par le navigateur. L'attribut `content` est obligatoire et définit la valeur de la méta-information. L'attribut `name` est optionnel et peut prendre les valeurs `author`, `description`, `keywords`. Par exemple,


```
<meta name="keywords" content="fromage, camembert, produit frais" />
```

Un autre attribut optionnel très important est `http-equiv` puisqu'il permet d'envoyer des en-têtes HTTP au navigateur pour l'aider à interpréter le document. Par exemple, `<meta http-equiv="refresh" content="5" />` permet de rafraîchir la page au bout de 5 secondes. Autre exemple important qui définit le codage utilisé, `<meta http-equiv="content-type" content="text/html; charset=UTF-8" />` indique le type de document à afficher au navigateur.

- l'élément `script` permet de référencer un fichier script externe et/ou de définir des fonctions JavaScript locales. Par exemple,

```
<script type="text/javascript" charset="iso-8859-1" src="javascript.js"></script>
```

Attention à ne pas créer un élément vide `<script ... />` lorsque tout le code est dans un fichier externe.

- l'élément `link` permet de lier un autre document. Par exemple, pour définir la feuille de style associée :


```
<link rel="stylesheet" href="stylesheet.css" type="text/css" />
```

2.3 Le corps du document

Le corps (*body*) du document HTML constitue ce qui va être affiché sur l'écran. Par défaut, les éléments sont affichés selon leur type :

- les éléments en ligne (*inline*) sont placés de gauche à droite jusqu'à ce qu'il n'y ait plus de place dans leur bloc et un retour à la ligne automatique est géré par le navigateur ; ainsi le texte d'un paragraphe est-il affiché avec des coupures de lignes dépendant de la taille du navigateur ; de même les ancres sont des éléments en ligne ;

- les éléments de type bloc sont affichés avec une coupure de ligne avant et après eux ; les paragraphes sont de type bloc ainsi que les en-têtes.

2.3.1 Attributs de base

Un certain nombre d'attributs de base communs à quasiment tous les éléments (*core attributes*) peuvent être utilisés.

class classe CSS de l'élément. Cela permet de formater l'élément selon un style défini dans une classe déjà définie ; par exemple, `class='monvert'` où `monvert` est une classe de style ;

style style en ligne ; par exemple, `style='color :green ;'`

id identifiant **unique** de l'élément permettant sa manipulation en JavaScript grâce à la méthode :

```
document.getElementById(id) ;
```

title *tooltip* affiché lors du survol de l'élément ;

De plus, une gestion des événements souris et clavier permettent d'associer des scripts JavaScript. Cette association est réalisée grâce à des attributs tels que `onclick` ou `onkeypress`.

2.3.2 Éléments indispensables

a *anchor* les ancres mettent en oeuvre l'hypertexte et peuvent être utilisées de 2 façons :

1. lien hypertexte référant une autre page web grâce à l'attribut `href`. Par exemple, pour aller sur le site du `w3schools` :

```
<a href='http://www.w3schools.com/'>CLIQUEZ ICI</a>
```

2. marque-page indiquant une cible potentielle d'un lien :

```
<a id='toto'>Texte cible</a>
```

Plus loin dans le même document, on pourra référencer cette cible par le lien suivant :

```
<a href='#toto'>aller au texte cible</a>
```

Bien entendu, on peut faire une référence externe sur une cible en suffixant le chemin de l'url par `#toto` ;

Enfin, la valeur d'un `href` peut-être une pseudo-URL contenant du code JavaScript qui sera exécuté lors de l'activation du lien comme dans l'exemple suivant :

```
<a href="javascript:alert('Bonjour le monde !')">Mon Lien</a>
```

Remarquons que le code est préfixé par `javascript` : afin d'indiquer

p paragraphe contenant une suite de phrases ;

br *break* élément vide qui permet de passer à la ligne suivante ; en HTML les espaces, tabulations et retour ligne (CR et/ou LF) multiples ne sont perçus que comme un espace séparateur lorsqu'ils sont situés entre deux mots.

h1 *header* de niveau 1 est un titre de section ; on trouve également **h2** ...

ul *unordered list* liste d'items non numérotés ;

ol *ordered list* liste d'items numérotés ;

li *list item* élément d'une liste numérotée ou non ;

div *division* est une section logique du document permettant de regrouper plusieurs blocs dans un même formatage ; on l'utilise souvent dans la mise en page du document pour scinder les différentes parties (bandeau haut, menu de gauche, page centrale, bandeau du bas) ;

table les tables HTML sont un moyen d'afficher les tableaux mais aussi de mettre en page un document (comme les `div`). Les tables peuvent ou doivent contenir :

caption la légende de la table ;

tr *table row* une ligne ;

td *table delimiter* une case ; `colspan` et `rowspan` sont des attributs indiquant le nombre de cases à fusionner avec la case courante vers la droite et vers le bas ;

th *table header* une case de titre de colonne ;

hr *horizontal rule* ligne de séparation horizontale ;

img *image* en format gif, png ou jpg ; les attributs indispensables :

src *source* chemin du fichier image ;

alt *alternative* texte utilisé si le navigateur ne sait pas afficher les images ;

width largeur en pixels ;

height hauteur en pixels ;

form les formulaires sont décrits dans la section suivante.

2.4 Les formulaires

Les formulaires constituent des éléments indispensables dans la saisie d'informations à destination d'une application web. Un ou plusieurs formulaires peuvent être présents dans une même page. L'attribut `name` de chaque champ de saisie correspond au nom du "paramètre" qui contiendra l'information. Plusieurs champs peuvent avoir le même nom, dans ce cas le paramètre sera de type tableau. L'attribut `tabindex` permet d'ordonner les champs de saisie quand l'utilisateur appuiera sur la touche de tabulation.

2.4.1 Éléments de formulaires

form élément racine d'un formulaire contenant les attributs suivants :

action est l'url où sera soumis le formulaire (généralement un script php où un programme cgi) ;

method soit `get`, soit `post` ; la première (`get`) insère les champs saisis par l'utilisateur à la fin de l'url après un point d'interrogation sous la forme `http://localhost/index.php?nom1=val1&nom2=val2` ; les noms sont les valeurs des attributs `name` des champs tandis que les valeurs sont les contenus saisis par l'utilisateur. La seconde méthode `post` "cache" les contenus saisis ;

target avec la valeur `_blank`, une nouvelle fenêtre sera ouverte, avec `_self` (par défaut) on recouvrira la fenêtre actuelle.

onsubmit code javascript à exécuter avant la soumission ; Si le code JavaScript retourne faux, la soumission est annulée !

name ou **id** nom ou id du formulaire pouvant être utile pour le référencement des champs de saisie dans le code javascript de vérification ;

input élément **vide** définissant un champ de saisie ayant :

- un attribut **name**, le nom du champ ;
- un attribut **type** permettant d'indiquer la forme du champ de saisie parmi :

text champ de type texte libre sur une ligne ; autres attributs : **size** la taille du champ de saisie à l'écran, **maxlength** le nombre maximum de caractères à saisir, **value** valeur par défaut ;

password champ de mot de passe caché ;

button bouton permettant de déclencher un script javascript ; autres attributs : **value** valeur affichée à l'écran, **onclick** code javascript à déclencher ;

checkbox case à cocher ; autres attributs : **value** valeur affectée au nom dans le cas où cette case est cochée, **checked="checked"** si on veut que la case soit cochée par défaut ;

radio case à cocher **exclusive** ; tous les boutons radio mutuellement exclusifs doivent avoir le même attribut **name** ; autres attributs : **value** valeur affectée au nom dans le cas où cette case est cochée, **checked="checked"** si on veut que la case soit cochée par défaut ;

hidden champ caché n'apparaissant pas dans le formulaire ; historiquement, les champs cachés permettaient de faire transiter l'information passée de proche en proche lors de la navigation ; autres attributs : **value** ; Actuellement, le mécanisme de session est plus pratique à utiliser.

submit bouton de soumission permettant d'exécuter l'**action** du formulaire ; plusieurs boutons de soumission peuvent coexister dans un même formulaire ;

image bouton de soumission utilisant une image comme visuel ; autres attributs : **src**, **alt**, **width**, **height** ;

reset réinitialisation des champs du formulaire ;

file pour envoyer au serveur un fichier localisé chez le client ; le formulaire doit posséder un attribut **enctype** ayant la valeur `multipart/form-data` et sa méthode doit être `post` ; Il est également possible de limiter la taille du fichier à envoyer en ajoutant un champs caché nommé `max_file_size` et de valeur la taille maximal acceptée en octets.

textarea élément non vide contenant une zone de texte multi-ligne ; attributs : **name**, **rows** nb de lignes, **cols** nb de colonnes ;

select élément non vide contenant une liste déroulante d'options ; attributs : **name**, **size** nb de lignes visibles, **multiple** de valeur **multiple** indique que plusieurs options sont possibles ;

option élément non vide contenu dans **select** ; attributs : **value** valeur qui sera associée au **name** du **select** conteneur, **selected** de valeur **selected** indique si cette option est présélectionnée ; le contenu de l'élément option sera le texte affiché dans la liste déroulante ;

Exemple 2 (Un formulaire de login)

```
<form action="login.php" method="post" >
Nom : <input type="text" size="20" name="login"><br />
Mot de passe : <input type="password" size="20" name="passwd"><br />
<input type="submit" value="Valider">
</form>
```

2.4.2 Inclusion d'autres objets

L'élément `Object` permet d'inclure un objet typé dans une page HTML. Ses attributs sont :

type indique le type de l'objet tel que : "text/html", "application/x-shockwave-flash", "video/x-msvideo", ...

data url du fichier tel que : ../commun.html, ma_video.avi, animflash.swf, ...

width et **height** indique la taille de l'inclusion;

Cet élément n'est pas vide et peut ou doit contenir des sous-éléments **paramètres** utilisés par le navigateur pour un type d'objet particulier.

Exemple 3

```
<param name="autostart" value="true" />
<param name="loop" value="true" />
Texte alternatif
```


Chapitre 3

JavaScript

JavaScript est un langage orienté objet, interprété par le navigateur côté client. C'est un langage orienté objet à **prototype**, c'est à dire que les bases du langage et son API est fournies par des objets prédéfinis (Number, String, Document) qui ne sont pas instanciés au sein de classes. Chaque objet contient des **propriétés** (slot), et des **méthodes**. Une des méthodes qui porte le même nom que l'objet est le **constructeur** permettant de générer d'autres objets. Il est sensible à la casse!

JavaScript (Firefox) et JScript (Internet Explorer) sont des implémentations de la norme ECMAScript. Il peut donc y avoir quelques différences subtiles entre plusieurs navigateurs!

3.1 Modèle événementiel

Les instructions JavaScript sont exécutées suite au déclenchement d'un événement auquel elles avaient été attaché dans le code HTML grâce à un attribut `onEVENT` où `EVENT` peut être un `click` souris, un `load` chargement du document, un `submit` de formulaire, ...

Les éléments cliquables ont une action par défaut : par exemple, un lien permettra de se rendre sur une nouvelle URI. Si un élément cliquable possède un attribut `onclick`, le code javascript associé à cet événement sera exécuté au préalable et l'action par défaut ne sera exécuté que si le code javascript ne retourne pas `false`.

```
<a href="somewhere.html" onclick="return doSomething()">
```

3.2 Variables

Les Variables en JavaScript n'ont pas de type défini, et n'importe quelle valeur peut être stockée dans n'importe quelle variable. Les variables **peuvent** être déclarées avec `var`. Ces variables ont une portée lexicale et une fois que la variable est déclarée, on peut y accéder depuis la fonction où elle a été déclarée (variable **locale**). Les variables déclarées en dehors d'une fonction et les variables utilisées sans avoir été déclarées en utilisant `var`, sont **globales** (peuvent être utilisées par tout le programme).

Exemple 4 (déclaration de variables)

```
x = 0; // Une variable globale
var y = 'Hello!'; // Une autre variable globale

function f(){
  var z = 'foxes'; // Une variable locale
  twenty = 20; // Globale car le mot-clef var n'est pas utilisé
  return x; // Nous pouvons utiliser x ici car il s'agit d'une variable globale
}
// La valeur de z n'est plus accessible à partir d'ici
```

3.3 Types de données

Les mots-clés `null` et `undefined` sont utilisés pour représenter une valeur nulle et une variable déclarée et non définie. Toutes les valeurs sont des objets mais il n'existe pas de classe. Les objets prédéfinis de Javascript (`string`, `number`, `array`, `object`, ...) peuvent être utilisés comme prototype d'autres objets créés :

- soit en utilisant l'opérateur `new` ;
- soit en utilisant une notation simplifiée JSON ;

Attention à l'affectation d'objets dans des variables : les objets simples (`Number`, `Date`, ...) sont copiés tandis que les objets complexes (`Array`, `Object`, ...) sont passés par référence !

Exemple 5

```
var i=new Number(5);j=i;j++; document.write("i=5 :"+i+" j=6 :"+j+"<br />");
tab.msg="hello world";var t2=tab;t2.msg="message de t2";
document.write("tab.msg :"+tab.msg); // affiche message de t2 !
```

3.3.1 Nombres (objet `Number`)

Les nombres en JavaScript sont représentés en virgule flottante comme des IEEE-754 Doubles, ce qui permet une précision de 14 à 15 chiffres significatifs. Comme ce sont des nombres binaires, ils ne représentent pas toujours exactement les nombres décimaux, en particulier les fractions.

Ceci pose problème quand on formate des nombres pour les afficher car JavaScript n'a pas de méthode native pour le faire. Par exemple :

```
alert(0.94 - 0.01); // affiche 0.9299999999999999
```

En conséquence, l'arrondi doit être utilisé dès qu'un nombre doit être affiché. La méthode `toFixed(nbDecimales)` de l'objet `Number` permet de le faire.

```
var num = new Number(13.37); // ou var num = 13.37;
document.write (num.toFixed(1)); // affiche 13.4
```

Les littéraux peuvent être spécifiés dans l'une des notations habituelles : `345` ; `34.5` ; `3.45e2` ; `0377` (octal) ; `0xFF` (hexa) ;

Le constructeur `Number` peut être utilisé pour réaliser une conversion numérique explicite :

```
var myString = "123.456"
var myNumber = Number( myString );
```

Inversement, pour convertir un nombre en chaîne, il faut utiliser la méthode `toString()` de `Number`. Cette méthode est également disponible dans de nombreux objets et permet leur affichage.

3.3.2 Chaînes de caractères (objet `String`)

Une chaîne de caractères en JavaScript est immuable et peut être directement créée en plaçant des caractères entre quotes (double ou simple).

```
var s = "Hello, world!";document.write(s.charAt(1)); // e
var ch = 'Bonjour';
```

Propriétés de l'objet `String` :

length longueur ;

Méthodes de l'objet `String` :

charAt(i) retourne caractère à la *i* ème position (0 à *n-1*) ;

opérateur == sensible à la casse ;

opérateur + concaténation ;

indexOf(searchvalue,(fromindex)) retourne -1 si introuvable, la position si trouvé ;

substring(start,(stop)) crée une sous-chaîne depuis un indice ;

3.3.3 Tableaux (objet `Array`)

Un `Array` est un tableau "creux" indicé par des entiers (0 à *n-1*) où seuls les éléments définis réservent de la mémoire.

```
tab = ["titi",1,,4.2,'tutu']; // longueur 6 et 4 elements
t2 = new Array(0,1,2,3,4,5); // longueur 6 et 6 elements entiers
t3 = new Array(36); // longueur 36 pour un tableau vide
```

Attention, un tableau est également un objet JavaScript : il peut également posséder des propriétés nommées par des noms et accessibles par notation pointée ou indexée.

Propriétés de l'objet `Array` :

`length` longueur du tableau ;

Méthodes de l'objet `Array` :

`push(newelement1(newelement2, ..., newelementX))` ajout d'un élément en queue ;

`Object pop()` supprime et retourne le dernier élément ;

`unshift(newelement1(newelement2, ..., newelementX))` ajoute en tête ;

`Object shift()` supprime et retourne le premier élément ;

3.3.4 Objets (objet `Object`)

Un objet JavaScript est un ensemble de couple (clé, valeur) ou tableau associatif, appelées **propriétés**, dont les clés sont des chaînes. Un objet peut être créé littéralement :

```
var o = {nom: 'dupont', prenom: 'Paul', age: 42};
```

Un objet peut également être créé grâce au constructeur d'`Object` :

```
b=new Object();
b.firstname="John";
b.lastname="Doe";
b.age=50;
```

Les propriétés peuvent être créées, modifiées et lues par la notation pointée ou par la notation indexée (crochets) ; Cette seconde façon permet d'accéder à des propriétés non connues à l'écriture du script (`o[x]`) !

```
var n = o.nom; var a = o['age'];
```

Constructeur et objets génériques

Si l'on veut construire plusieurs objets similaires (mêmes propriétés), il faut créer un constructeur puis l'appeler :

```
function person(nom, prenom, age){
  this.nom=nom; // this référence l'objet en cours de création
  this.prenom=prenom;
  this.age=age;

  this.toString=function(){return this.nom+this.prenom+this.age.toString();}
}
var toto=new person('Toto', 'Pierre', 25);
document.write(toto+'<br />'); // TotoPierre25
```

On peut détruire des propriétés d'un objet ou l'objet lui-même en utilisant l'opérateur `delete` :

```
delete o.nom;
delete o;
```

Objets prédéfinis

JavaScript possède des objets prédéfinis tels que `Array`, `Boolean`, `Date`, `Function`, `Math`, `Number`, `Object`, `RegExp` et `String`. L'environnement d'exécution (le navigateur) apporte d'autres objets prédéfinis dans le DOM (Document Object Model) tels que `window`, `document`, `form`, `link`.

Héritage et prototype

On peut dynamiquement modifier la structure d'un constructeur en lui ajoutant de nouvelles propriétés et méthodes. Les objets instanciés auparavant ne sont pas affectés mais toute nouvelle instance du constructeur héritera des propriétés et méthodes ainsi rajoutées. La propriété **prototype** est possédée par tout objet ...

Exemple 6 (ajout de la fonctionnalité de recherche d'un élément dans un tableau)

```

Array.prototype.in_array = function(p_val) {
  for(var i = 0, l = this.length; i < l; i++) {
    if(this[i] == p_val) {
      return true;
    }
  }
  return false;
}

```

3.3.5 Booléen (objet Boolean)

Deux littéraux `true` et `false` définissent le type booléen. Lors des conversions en booléen (conditionnelle, répétitive, expression utilisant des opérateurs booléens, ...), `0`, `-0`, `null`, `""`, `false`, `undefined`, et `NaN` sont transformés en `false` tandis que toute autre valeur est vraie!

3.4 Fonctions (objet Function)

Le nombre d'arguments réels ne correspond pas forcément au nombre de paramètres définis. La propriété `arguments` est un tableau des paramètres passés.

```

function function-name(arg1, arg2, arg3) {
  statements;
  return expression;
}

```

3.5 Opérateurs

Les opérateurs de Java ou de C++ fonctionnent sans grande différence en JavaScript. Le langage étant faiblement typé, il faut éviter d'ajouter (+) des nombres à des chaînes (cela donne des chaînes). De même, dans les comparaisons, certaines conversions de type sont implicites et l'opérateur `===` permet de vérifier l'égalité de type et de valeur! Idem pour `!==`.

Les opérateurs logiques à court-circuit `&&` et `||` retourne le premier ou le second opérande quel qu'il soit (booléen ou non).

3.6 Structures de contrôle

Très classiques.

3.6.1 Alternatives

```

if (expr) {
  statements;
} else if (expr) {
  statements;
} else {
  statements;
}

```

```

switch (expr) {
  case VALUE:
    statements;
    break;
  case VALUE:
    statements;
    break;
  default:
    statements;
}

```

```

    break;
}

```

3.6.2 Itératives

```

for (initial-expr; cond-expr; expr evaluated after each loopround) {
    statements;
}
while (cond-expr) {
    statements;
}
do {
    statements;
} while (cond-expr);

```

3.6.3 Parcours des propriétés (et/ou des indices) d'un objet

Il faut noter que le parcours n'est effectué que pour les **propriétés définies par le programmeur** : les propriétés prédéfinies de l'objet (constructor, prototype, length, ...) ne sont pas parcourues.

```

for (var p in o) {
    ... o[p] ...;
}

```

Exemple 7

```

var tab=["un",2,'trois',[1,2,4,8]];
tab.msg="hello world";
tab.push(5);
for (var i in tab){
    document.write(i+" : "+tab[i]+'<br />');
}
document.write("longueur : "+tab.length);

```

```

0 : un
1 : 2
2 : trois
3 : 1,2,4,8
msg : hello world
4 : 5
longueur : 5

```

Cet exemple montre que `tab` possède 5 cases de tableau indicées de 0 à 5 et une propriété d'objet `msg`.

3.6.4 Utilisation d'objet implicite (with)

```

with(document) {
    var a = getElementById('a'); // méthode de document
    var b = getElementById('b');
};

```

3.6.5 Exceptions

```

try {
    // instructions pouvant conduire à une exception
} catch(e) {
    // e étant une chaîne du type "InvalidNameException"
} finally {
    // toujours exécuté
}

```

Un au moins des blocs `catch` ou `finally` doit être présent.

3.7 Divers

JavaScript est sensible à la casse. La syntaxe des commentaires est la même qu'en C++. JSON (JavaScript Simple Object Notation) permet d'initialiser des objets {} et/ou des tableaux [] de manière simple directement en JavaScript :

```
tab=[{'nom':'Dupont', 'age':35, 'hobbies':['golf', 'lecture']},  
     {'nom':'Durand', 'age':45, 'hobbies':[]},  
     {'nom':'Martin', 'age':40, 'hobbies':['poker']}]
```

Chapitre 4

Le Document Object Model (DOM 2) et JavaScript

4.1 Introduction

Le modèle objet de document HTML est destiné à décrire une interface de programmation d'applications (API) indépendante du langage (JavaScript, perl, C++, ...). La manipulation du document HTML est réalisée via une arborescence de noeuds (Node) représentant des éléments (Element) HTML ainsi que les noeuds texte ou les noeuds attributs. La description complète de l'API est sur le site [6].

Cette API est décrite dans le langage IDL "Interface description language". C'est un langage voué à la définition d'interfaces de composants logiciels, permettant de faire communiquer les modules implémentés dans des langages différents. IDL est défini par l'OMG et utilisé notamment dans le cadre d'applications CORBA.

On peut voir dans la figure 4.1, un exemple d'arbre DOM.

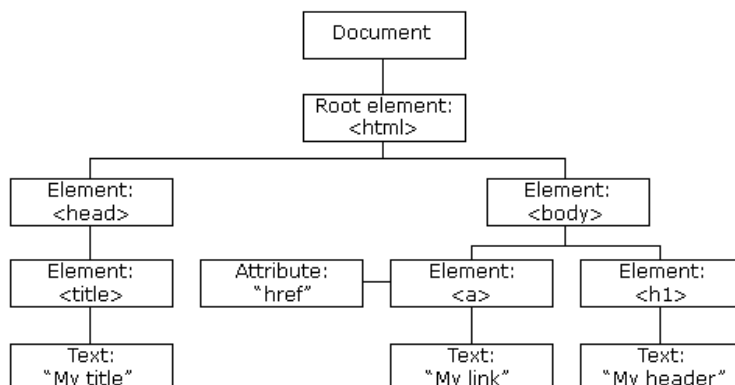


FIGURE 4.1 – arbre DOM HTML

4.2 Interface du Document HTML

Il constitue la racine de l'arborescence, a comme nom `document` et possède des attributs (propriétés d'objet) tel que `forms` et des méthodes telle que `write()`. Voici l'interface IDL d'un document HTML, tel que décrit par le W3C :

```
interface HTMLDocument : Document {
    attribute DOMString        title;
    readonly attribute DOMString    referrer;
    readonly attribute DOMString    domain;
    readonly attribute DOMString    URL;
    attribute HTMLCollection    body;
    readonly attribute HTMLCollection    images;
    readonly attribute HTMLCollection    applets;
```

```

readonly attribute HTMLCollection  links;
readonly attribute HTMLCollection  forms;
readonly attribute HTMLCollection  anchors;
        attribute DOMString        cookie;
                                // raises(DOMException) on setting
void          open();
void          close();
void          write(in DOMString text);
void          writeln(in DOMString text);
NodeList      getElementsByName(in DOMString elementName);
};

```

Nous remarquons que la description de l'interface peut être implémentée de diverses façons :

- les attributs peuvent ne pas exister en tant qu'objet mais être obtenus avec des accesseurs (get/set);
- des ajouts de méthodes et d'attributs peuvent exister : par exemple, en JavaScript la méthode `getElementById()` permet de récupérer un élément grâce à son identifiant ;
- la construction de nouveaux objets étant très dépendante du langage d'implémentation, elle n'est pas spécifiée dans l'interface : par exemple, en JavaScript la méthode `createElement()` permet de créer un élément.

Dans la suite de ce chapitre, nous allons présenter l'implémentation JavaScript de la manipulation du DOM. Malheureusement, seule l'interface EcmaScript étant normalisée, nous allons présenter les attributs et méthodes communs aux navigateurs courants (Firefox, IE, Opera). De plus, l'inventaire exhaustif étant trop important, il est préférable de se référer à la documentation en ligne du site [7]. Nous ne décrivons ici que quelques objets fondamentaux.

Enfin, HTML évoluant vers XML (avec XHTML), nous décrivons aussi le DOM XML [8] qui est une évolution du DOM HTML et qui contient de nombreuses améliorations en ce qui concerne la gestion de l'arborescence (noeuds).

De plus, certains objets JavaScript tels Window ou Navigator sont très utiles même s'ils ne font pas partie du DOM et ne sont référencés que dans la documentation du DOM HTML [7]! **D'autres objets JavaScript tel que document implémentent les méthodes du DOM HTML (comme write()) mais implémentent aussi les méthodes du DOM XML (comme createTextNode()).**

4.3 document

L'objet document contient :

des collections telles que `forms[]` permettant d'accéder aux formulaires du document :

- soit par un indice entier compris entre 0 et `document.forms.length-1`; par exemple : `document.forms[0]` ;
- soit par une chaîne de caractères correspondant au nom du formulaire ; par exemple : `document.forms['monform']` ;

des propriétés telle que URL qui indique l'adresse du site courant ;

des méthodes telles que `getElementById('id')` qui retourne un élément (ou null), `getElementsByName('nom')` qui retourne un tableau des éléments ayant ce nom.

collections forms collection des formulaires accessibles par leur nom (attribut name) :

- soit par `document.monform`; attention à cette manière "historique" d'accéder au formulaires, images, input : en effet, `monform` n'est pas une propriété de l'objet document mais de l'objet forms de document ! Aussi, chaque navigateur peut prendre en charge cette fonctionnalité de manière différente...
- soit par `document.forms['monform']`
- soit par `getElementById('monid')` (unicité) ou `getElementsByName('monform')`

images collection des images du document ;

- soit par `document.monimage`
- soit par `document.images['monimage']`
- soit par `getElementBy...`

Remarquons que l'accès aux éléments par leur nom (en notation pointée ou par la clé du tableau associatif) nécessite que ce nom soit **unique** ce qui n'est absolument pas certain. S'il y a conflit de nom, la référence sera `undefined` mais on pourra toujours accéder aux éléments en utilisant des indices entiers sur le tableau collection ou par un identifiant.

propriétés body accès à l'élément body ;

url url du document ;

méthodes `getElementById()` `getElementsByTagName()`, `getElementsByName()` retourne un ou une collection d'éléments

`write()` ou `writeln()` écrit dans le document ;

`createElement()` , `createTextNode()` , `createAttribute(name)` créent des noeuds qu'il faudra ensuite "accrocher" à l'arborescence grâce à : `p.appendChild(f)` qui permet d'ajouter, un élément ou un texte ou un attribut, fils `f` à un élément parent `p` ;

4.4 Window

L'objet `window` est le parent de `document`. C'est donc la racine de toute la hiérarchie du DOM. Il contient de nombreuses méthodes indispensables ! A noter que l'on peut ne pas citer l'objet `window` : il est toujours en `with` !

collections `frames[]` est la collection des frames ;

propriétés `document` le document

location possédant une propriété `href` qui est l'url de la page ;

status permet d'écrire dans la barre d'état du navigateur ;

toolbar et **statusbar** permettent de cacher les barres du navigateur ;

méthodes `alert('msg')` ouvre une boîte d'avertissement avec un bouton OK ;

`close()` ferme la fenêtre courante ;

`confirm('msg')` ouvre une boîte de questionnement avec OK et Annuler, retourne un booléen indiquant le choix de l'utilisateur ;

`open(...)` ouvre une nouvelle fenêtre (pop up) ;

`prompt('msg','defaut')` ouvre un dialogue avec

- un message demandant de saisir une chaîne ;
- un champ de saisie (initialisée à 'defaut')
- deux boutons OK et Annuler ;

Cette méthode retourne `null` si le bouton Annuler a été cliqué ou bien la chaîne saisie si OK a été cliqué.

`setTimeout("alert('5 secondes écoulées !')",5000)` ; permet de lancer du code après un certain nombre de millisecondes ;

`setInterval("alert('Assez !')",10000)` ; permet de lancer du code tous les x millisecondes ; utile pour afficher l'heure !

4.5 Evénements

Les événements sont créés par l'utilisateur qui agit sur l'interface du navigateur en cliquant sur la souris ou en tapant au clavier. HTML permet d'associer à certains événements survenant sur des éléments du DOM, du code JavaScript qui sera déclenché automatiquement. Cette association est réalisée par des **gestionnaires d'événement** qui sont des **attributs** d'élément commençant par **on...** Dans l'exemple 8, le code HTML permettra d'afficher un bouton qui une fois cliqué ouvrira une boîte d'avertissement !

Exemple 8 (Gestion simple d'événement)

```
<button onclick="alert('BOUTON CLIQUE !')">Cliquez-moi</button>
```

L'attribut `onclick` est le gestionnaire d'événement et sa valeur est une chaîne contenant le code javascript à exécuter lorsque l'événement survient. Attention, certains gestionnaires d'événement ne sont utilisables que sur certains éléments !

4.5.1 Gestionnaire d'événement

Une liste des gestionnaires d'événements les plus couramment utilisés suit :

onfocus (resp. **onblur**) lorsque le focus arrive (resp. part) sur un élément ;

onmouseover (resp. **out**, **down up**, **move**) gestionnaires des événements souris ;

onkeypress (resp. **down up**) gestionnaires des événements clavier ;

onload lorsque l'élément est complètement chargé sur la page : utile notamment pour l'élément `body` afin de déclencher une action lorsque la page entière est affichée ;

onsubmit spécifique à l'élément `form`, permet d'exécuter des vérifications sur les champs du formulaire avant de soumettre celui-ci. **Attention, si le code JavaScript retourne faux le formulaire ne sera pas soumis** (voir modèle événementiel de JavaScript) !

4.5.2 L'objet Event

Un événement est un objet JavaScript qui est créé lors de l'action de l'utilisateur et qui peut être transmis au code JavaScript par l'intermédiaire du mot-clé **event**. Chaque action de l'utilisateur crée un événement différent et les propriétés de celui-ci informent sur l'action de l'utilisateur. Par exemple, la propriété **target** de l'objet **event** indique l'élément sur lequel l'événement s'est produit. L'exemple 9 montre comment afficher l'id de l'élément cible. Attention aux différences entre navigateurs en ce qui concerne l'objet **event** !

Exemple 9 (Gestion de la cible de l'événement)

```
<script type="text/javascript">
function indiqueCible(e){
    var cible=e.target;
    alert("La cible est l'élément d'id : " + cible.id);
}
</script>
</head>
<body >
<p id="p1" onmousedown="indiqueCible(event)">
Cliquez sur ce paragraphe identifié par p1 !
</p>
<p id="p2" onmousedown="indiqueCible(event)">
Cliquez sur ce paragraphe identifié par p2 !
</p>
```

Liste des propriétés d'event

Voici les principales propriétés de l'objet **event** :

target élément activé par l'utilisateur ;

button entier indiquant le bouton de la souris pour les événements souris (attention aux différences de codage des navigateurs (0,1,2 pour Firefox, 1,4,2 pour IE) ;

clientX position horizontale en pixels de la souris dans la fenêtre du navigateur (0 à gauche, xxx à droite) ;

clientY position verticale en pixels de la souris dans la fenêtre du navigateur (0 en haut, xxx en bas) ;

screenX (resp. **Y**) position en pixels de la souris dans l'écran ;

keyCode code ASCII de la touche du clavier ; attention aux différences entre navigateurs (voir l'url <http://unixpapa.com/js/testkey.html>).

4.6 Formulaire

Un objet **Form** correspond à chaque formulaire du document HTML.

collections **elements**[] est le tableau des éléments du formulaires : champs, boutons, ...

propriétés **action** url de soumission ;

id id du formulaire ;

name nom du formulaire ;

méthodes **reset()** remet à 0 les champs du formulaire ;

submit() soumet le formulaire ; attention, la soumission par cette méthode ne prend pas en compte le gestionnaire d'événement **onsubmit** qui est sensé vérifier les champs du formulaire !

L'exemple 10 illustre deux fonctions permettant de vérifier si un champ ou un tableau de champ est vide. De plus, une fois vérifié que les champs sont remplis, on demandera confirmation à l'utilisateur !

Exemple 10 (Fonctions testant la vacuité des champs)

```
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" />
```

```

<meta http-equiv="Content-Language" content="fr" />
<title>Exemple de formulaire de login</title>
<script type="text/javascript">
function isEmpty(form,element){
  if (typeof(element)=="string"){ // un seul element à tester
    if (document.forms[form].elements[element].value.length==0){
      document.forms[form].elements[element].focus(); //positionner
      return true;
    } else {
      return false;
    }
  } else{ /* element est un tableau new Array('nom',prenom) */
    for(var i=0;i<element.length;i++){
      var vide=document.forms[form].elements[element[i]].value.length==0;
      if (vide) {
        document.forms[form].elements[element[i]].focus();
        return true;
      }
    }
    return false;
  }
}
function alertIfEmpty(form,element){
  if (isEmpty(form,element)){
    alert("Au moins un champ de saisie obligatoire n'est pas renseigné !");
    return false;
  }
  else{
    return true;
  }
}
</script>
</head>
<body>
<form action="login.php" method="post" name="monform"
  onsubmit="return alertIfEmpty('monform', new Array('login','passwd')) &&
    confirm('Voulez-vous vraiment soumettre ?')">
Nom : <input type="text" size="20" name="login"><br />
Mot de passe : <input type="password" size="20" name="passwd"><br />
<input type="submit" value="Valider">
</form>
</body>
</html>

```

4.7 Un exemple d'arbre DOM

Afin d'illustrer le parcours récursif de l'arbre DOM d'un document HTML, les exemples suivants sont à étudier.

Exemple 11 (Le fichier HTML arbreDomHtml.html)

```

<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
<meta http-equiv="content-type" content="text/html; charset=ISO-8859-1" />

<title>Arbre DOM HTML</title>

<script language="Javascript" type="text/javascript" src="arbreDomHtml.js" >

```

```

</script>
</head>
<body onload='lanceAnalyseDOM()>

<h1>Arbre DOM HTML</h1>
<p class="resume">
hello World !
<a href=".">lien dans un p </a>
<h2>section b</h2>
abcde
<button onclick="alert('BOUTON CLIQUE !')">Cliquez-moi</button>
</body>
</html>

```

Exemple 12 (Le fichier JavaScript arbreDomHtml.js)

```

function analyseDOM(noeud, indent, dest) {
  if (noeud == null || noeud == dest) return; /* evite recursion infinie ! */
  var type = 'NiElemNiAttribNiText';
  switch (noeud.nodeType) {
    case 1 : type = 'Element';
      break;
    case 2 : type = 'Attribut';
      break;
    case 3 : type = 'Text';
      break;
  }
  var t=document.createTextNode('');
  for (var i=0; i < indent; i++) {
    t.appendData("___ ");
  }
  t.appendData(type+"-"+noeud.nodeName+" ");
  t.appendData(noeud.nodeValue?"'+noeud.nodeValue+'":");
  t.appendData(" Enfants : "+noeud.childNodes.length);
  dest.appendChild(t);
  dest.appendChild(document.createElement("br"));
  for (var enfant=0; enfant < noeud.childNodes.length; enfant++) {
    analyseDOM(noeud.childNodes.item(enfant),indent+1,dest);
  }
}

function lanceAnalyseDOM() {
  var divDOM = document.createElement("div");
  /* division dans la quelle on va afficher le DOM */
  divDOM.appendChild(document.createTextNode(''));
  /* texte à afficher */
  divDOM.style.cssText = 'left:1500px; top:1500px; background:yellow;';
  /* style des textes dans divDOM (jaune) */
  document.body.appendChild(divDOM); /* ajout au body */
  analyseDOM(document, 0, divDOM); /* analyse de tout le doc */
}

// setTimeout('lanceAnalyseDOM()', 100);
// ne lance l'analyse qu'après 1s (chargement de la page et du script)

```

Exemple 13 (Le résultat produit)

```

NiElemNiAttribNiText-#document Enfants : 2
___ NiElemNiAttribNiText-HTML Enfants : 0
___ Element-HTML Enfants : 2
___ ___ Element-HEAD Enfants : 8
___ ___ ___ Text-#text " " Enfants : 0
___ ___ ___ Text-#text " " Enfants : 0

```

```
--- --- --- Element-META Enfants : 0
--- --- --- Text-#text " " Enfants : 0
--- --- --- Element-TITLE Enfants : 1
--- --- ---   ___ Text-#text "Arbre DOM HTML" Enfants : 0
--- --- ---   Text-#text " " Enfants : 0
--- --- --- Element-SCRIPT Enfants : 1
--- --- ---   ___ Text-#text " " Enfants : 0
--- --- ---   Text-#text " " Enfants : 0
--- --- Element-BODY Enfants : 9
--- --- --- Text-#text " " Enfants : 0
--- --- --- Element-H1 Enfants : 1
--- --- ---   ___ Text-#text "Arbre DOM HTML" Enfants : 0
--- --- ---   Text-#text " " Enfants : 0
--- --- --- Element-P Enfants : 3
--- --- ---   ___ Text-#text " hello World ! " Enfants : 0
--- --- ---   Element-A Enfants : 1
--- --- ---     ___ Text-#text "lien dans un p " Enfants : 0
--- --- ---     Text-#text " " Enfants : 0
--- --- --- Element-H2 Enfants : 1
--- --- ---   ___ Text-#text "section b" Enfants : 0
--- --- ---   Text-#text " abcde " Enfants : 0
--- --- --- Element-BUTTON Enfants : 1
--- --- ---   ___ Text-#text "Cliquez-moi" Enfants : 0
--- --- ---   Text-#text " " Enfants : 0
```


Chapitre 5

Les feuilles de styles CSS

5.1 Introduction

Le langage CSS (Cascading Style Sheets : feuilles de style en cascade) sert à décrire la présentation des documents HTML et XML. Les standards définissant CSS sont publiés par le World Wide Web Consortium (W3C). Introduit au milieu des années 1990, CSS 2.1 est couramment utilisé dans la conception de sites web et est bien pris en charge par les navigateurs web actuels. La documentation de référence absolue est à l'adresse <http://www.w3.org/TR/CSS21/>.

La spécification de CSS 2.1 est basée sur le modèle arborescent de document html : **Document Object Model (DOM)**.

5.2 Objectifs

- séparer la structure d'un document de ses styles de présentation ;
- définir le rendu d'un document en fonction du média de restitution et de ses capacités (type de moniteur ou de dispositif vocal), de celles du navigateur textuel, ...
- permettre la cascade des styles, c'est-à-dire un héritage multiple entre les origines (auteur, utilisateur), le média, la localisation des définitions de style (externe, en ligne, ...);

5.3 Syntaxe

Un style est défini par : `selecteur {prop1:val1; prop2:val2 val3 val4; prop3: v1, v2, v3; ...}`.

Le **sélecteur** définit l'ensemble des balises affectées par le style. Chacune des 53 **propriétés** est répartie dans un groupe parmi les 6 suivants :

font contenant font-family, font-size, font-weight, ...

color, background contenant color, background-color, background-repeat, background-image, ...

text contenant text-align, text-indent, ...

box, layout contenant padding, border, margin, width, height, display, float, ...

list contenant list-style-type, list-style-position, ...

tag contenant

Les **valeurs** sont réparties en plusieurs catégories :

mot-clé non sensible à la classe tel que : red, underline, bold, top, ...

longueur nombre décimal absolu ou relatif (démarrant par un signe) suivi par une unité sur 2 lettres. Les unités suivantes sont possibles :

cm, mm centimètre, millimètre ;

in inch (2.54cm) ;

pt point (1/72 inch) ;

pc pica (12pt ou 4.2mm) ;

px pixel (dépend de la résolution de l'écran) ;

em hauteur de la police courante ;

ex hauteur d'un x de la police courante ;

Par exemple, +0.25em, -3em, 100px, sont des longueurs correctes dont les deux premières indiquent un agrandissement et un rétrécissement par rapport à la valeur par défaut ;

pourcentage longueur relative telle que width :50%, line-height :120%, ...

couleur exprimée en notation RGB (Red, Green, Blue) par :

6 chiffres hexadécimaux tels que #FFFFFF pour le blanc ;

3 chiffres hexadécimaux tels que #000 pour le noir ; #F00 pour le rouge (chaque chiffre est répété) ;

rgb(128,0,128) pour le pourpre (également rgb(50%,0,50%)) ;

url avec la notation url(<http://toto.fr/chemin>) ;

5.4 Quelques exemples simples

Exemple 14 (styles dans l'en-tête)

Ce premier exemple utilise des styles définis dans l'en-tête html du document. Remarquons que pour p, la liste des polices est séparée par des virgules indiquant qu'un seul des éléments de la liste sera choisi. Pour la propriété border de div en revanche, la liste de valeur est une concaténation (sans virgule) de valeur (color, width, style).

```
<style type="text/css">
<!--
p {
  font-family : cursive, fantasy, monospace;
  line-height : 100%;
}
h1 {
  text-align: center;
  font-variant : small-caps;
}
div {
  float: left;
  width: 30%;
  border: blue 1px solid;
}
-->
</style>
</head>
<body>
...
```

Les commentaires html entourant la définition des styles est destinée aux navigateurs ne comprenant pas les styles CSS. Il est utile de valider sa feuille de style en faisant appel au validateur du W3C : <http://jigsaw.w3.org/css-validator/>. Les divisions flottantes permettent de placer les divisions dans le flot des boîtes affichées. Ici, trois divisions (3*30%<100%) pourront être placées horizontalement dans la page.

Exemple 15 (positionnement fixe de l'en-tête)

Voici un ensemble de styles permettant de définir une division d'en-tête fixée en haut du navigateur et une division principale qui lorsqu'elle défilera, passera sous l'en-tête. La propriété z-index définit l'empilement des boîtes à l'affichage (par défaut 0).

```
div.entete {
  position: fixed;
  z-index: 1;
  background-color:aquamarine;
  top : 0px;
  height: 100px;
  width: 800px;
  border: red 1px solid
}
div.principale {
  position: absolute;
```

```

width: 800px;
top : 100px;
border: red 1px solid
}
div {
float: left;
width: 30%;
border: blue 1px solid
}

```

Dans le corps du document, on aura une structure en division telle que ce qui suit :

```

<div class="entete">
<h1>Titre de niveau 1</h1>
</div>

<div class="principale">

<div>
<p>hello World !</p>
</div>

<div>
<p>abcde abcde abcde abcde abcde abcde abcde a...
</div>

</div> <!-- principale -->

```

5.5 Localisation des styles

Les styles peuvent être définis de différentes façons et avec une priorité **décroissante** :

en ligne à éviter car le style est un attribut de balise et la séparation contenu et forme n'est plus assurée ;

dans l'en-tête comme dans le premier exemple, l'élément style peut être placé dans l'élément `head` du document html ; sa portée concerne **seulement** le document où il est défini ;

dans un autre fichier de style d'extension `.css` qui sera lié aux documents html par un élément `link`. C'est la meilleure méthode ! On peut également lier le document html à plusieurs fichiers de styles : en cas de conflit, c'est le dernier fichier lié qui est prépondérant ;

Exemple 16 (un exemple de liaison à deux fichiers de style)

```

<head>
<meta http-equiv="content-type" content="text/html; charset=ISO-8859-1" />
<title>Fichier CSS externe</title>
<link rel="stylesheet" type="text/css" href="css3.css">
<link rel="stylesheet" type="text/css" href="css32.css">
</head>

```

Supposons que dans le fichier `css3.css`, on a `h1 {color: red;}` et que dans le fichier `css32.css`, on a `h1 {color: blue;}`. Alors, les éléments `h1` seront bleu.

5.6 Sélecteur

Dans un style, le sélecteur définit les éléments affectés par les propriétés. Dans les exemples précédents les sélecteurs simples tels que `p` ou `div` ou `h1` désignent tous les éléments de cette catégorie. Mais on peut raffiner la sélection en utilisant de nombreux procédés.

5.6.1 Correspondance de modèle (pattern matching)

Avec CSS, des expressions régulières permettent de sélectionner certains éléments du DOM. Ces expressions régulières utilisent des opérandes (les éléments notés E, F tels que `div` ou `p`) et des opérateurs que nous allons décrire succinctement :

- * correspond à tous les éléments de n'importe quel type ;
- E F** correspond aux élément F descendant de E ; par exemple : `h1 a {color : red}` correspond aux liens situés dans un titre de niveau 1 qui seront en rouge ;
- E > F** correspond aux éléments F fils de E ; `ol > li {font-weight: bolder;}` met en gras les listes numérotées mais pas les autres (ul) ;
- E + F** correspond aux F qui sont un frère suivant d'un E ; par exemple : `h2 + p {color: yellow}` met en jaune le paragraphe qui suit un titre de niveau 2 ;
- E[nom="val"]** correspond aux éléments E ayant un attribut `nom` de valeur `val` ; par exemple :
`input[type="text"] {color: blue}` met les champs de saisie en bleu ; On peut ne pas mentionner la valeur afin de sélectionner tous les éléments ayant un attribut : `*[id]` sélectionne tous les éléments possédant l'attribut `id`.

5.6.2 Sélecteur multiple ,

Pour avoir tous les éléments de titre centrés :

```
h1, h2, h3 {text-align: center}
```

5.6.3 classes de style .

Une classe de style permet d'affecter différents styles à un même élément HTML selon le contexte où il est utilisé. Par exemple, un paragraphe de résumé pourra être écrit en italique alors que les paragraphes "normaux" ne le seront pas. Pour cela, il suffit de définir un attribut `class` de l'élément à styliser.

```
p {
  font-family : monospace;
}
p.resume {
  font-style : italic;
  line-height : 80%;
}
```

Remarquons que les paragraphes de résumé héritent de la famille de police monospace. Voici l'utilisation des deux styles.

```
<p class="resume">
  hello World ! abcde abcde abcde abcde abcde abcde abcde abcde abcde abcde
</p>
<p>abcde abcde abcde abcde abcde abcde abcde abcde abcde abcde
abcde abcde abcde abcde abcde abcde abcde abcde abcde abcde abcde
</p>
```

Cette façon de procéder est très fréquente pour définir des divisions sémantiques du document avec des `div` en leur affectant des classes différentes (voir l'exemple 15).

Classe générique

On peut créer une classe sans l'associer à un élément html. Dans ce cas, le sélecteur est composé d'un point suivi du nom de classe. Cette classe peut être affectée à différents éléments :

```
.italique {font-style: italic;}
...
<p class="italique">bla bla </p>
<h4 class="italique">titre</h4>
```

Identifiant de classe

On peut utiliser (avec `#`) l'attribut `id` des éléments html pour définir une classe mais ce style ne pourra être affecté qu'à l'unique élément qui possèdera cet identifiant !

```
#valider { color : yellow;}
h1#premier { font-style : italic;}
```

Pseudo-classe :

Les pseudo-classes et pseudo-éléments permettent de sélectionner des objets ou des classes qui ne sont pas des noeuds du DOM :

pseudo-élément la première ligne d'un paragraphe, ou un élément généré par la feuille de style ;

pseudo-classe classe acquise dynamiquement ou déduite d'un parcours du DOM ;

:first-child sélectionne seulement les paragraphes premiers fils : `p:first-child { text-indent: 0 }`

:link sélectionne les liens avant qu'ils aient été visités : `a:link { color: blue }`

:visited sélectionne les liens après qu'ils aient été visités : `a:visited { color: blue }`

:focus sélectionne l'élément ayant le focus : `input[type="text"]:focus {color: red}`. Durant la frappe du texte, celui-ci est rouge ;

:hover sélectionne l'élément ayant le pointeur dessus (souvent utilisé pour les liens) :

`input[type="text"]:hover {color: yellow}`. Durant le survol, le texte est jaune ;

:active sélectionne l'élément en train d'être activé (clic souris) : `input[type="text"]:active {color: black}`.

:first-line première ligne d'un paragraphe par exemple ;

:first-letter première lettre ;

:before pour générer du texte avant un certain élément :

```
body {
    counter-reset: chapter;      /* Crée un compteur de chapitre */
}
h1:before {
    content: "Chapitre " counter(chapter) ". "; /* contenu généré */
    counter-increment: chapter; /* chapter++ */
    counter-reset: section;     /* Crée un compteur de section */
}
h2:before {
    content: counter(chapter) "." counter(section) ". ";
    counter-increment: section;
}
}
```

Ce style permettra de précéder chaque titre de niveau 1 d'un "Chapitre 12. " et chaque titre de niveau 2 d'un "Section 12.3."

:after pour générer un contenu (texte, image, ...) après un élément ;

```
body:after {
    content: "Fin du body";
    display: block;
    margin-top: 2em;
    text-align: center;
}
}
```

Dans cette exemple, le texte "Fin du body" sera affiché en fin de page.

A noter que les 4 derniers types de sélecteurs sont des pseudo-éléments et non des pseudo-classes et qu'ils doivent donc être à la fin d'un sélecteur.

5.7 Modèle de visualisation

L'arbre DOM du document est affiché selon un modèle de visualisation utilisant le principe des "boîtes" TeX. Chaque élément de type bloc (p ou li) génère une boîte bloc tandis que les éléments "inline" génèrent des boîtes inline. Le calcul des dimensions et de la position de ces boîtes est dynamique au fur et à mesure du chargement de la page et/ou de la dynamique interactive.

L'étude de ce modèle est en dehors des limites de ce cours mais peut-être étudié sur le Web en <http://www.w3.org/TR/CSS21/visuren.html>.

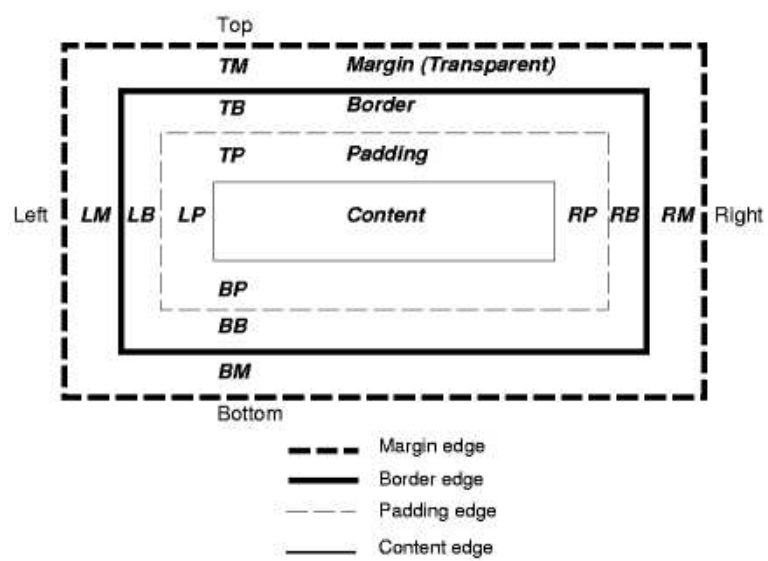


FIGURE 5.1 – une boîte et son entourage

Chapitre 6

HTTP et Application côté serveur

L'application côté serveur est généralement interprété par un module du serveur HTTP (souvent Apache).

6.1 Apache

Apache est un serveur http. Le serveur (démon) lance un processus qui génère des threads pour gérer chaque requête. Un serveur peut gérer plusieurs serveurs virtuels.

6.1.1 Configuration

Le fichier de configuration **httpd.conf** est lu au lancement du serveur. Ce fichier est situé dans : `${ServerRoot}/conf/httpd.conf`, avec `ServerRoot` valant habituellement `/etc/httpd/` sous Unix, `C:/Apache/` sous windows. Les **directives** de conf. sont de la forme `nomParamètre valeur` sans commentaire et sont réparties en 3 sections :

- contrôle du démon (environnement global) ;
- paramètres du serveur par défaut (ou principal) (variables par défaut pour les hôtes virtuels) ;
- paramètres pour les hôtes virtuels (recevant des requêtes à des adrs IP différentes mais étant gérées par le même processus serveur

Si les noms de fichiers sont préfixés par `/` ou `C:/` pour Win32 (et pas `C:\`), le serveur utilise ce nom absolu, sinon ils seront préfixés automatiquement par `$ServerRoot`.

La description détaillée de ces directives est en dehors des objectifs de ce cours.

6.1.2 Débogage

error.log journal des erreurs du serveur apache

access.log journal des accès datés et typés (GET, POST, ...) réussis ou non

6.1.3 Authentification Apache

La demande d'authentification par Apache est réalisée lorsque la requête traverse un répertoire contenant un fichier **.htaccess** spécifiant une authentification :

directive	description
AuthType Basic	type d'authentification
AuthUserFile "/home/pdupont/.../passwd"	référence absolue au fichier des mots de passe cryptés (md5) au format <code>nom :paswdcrypté</code>
AuthUserFile "/Mes Documents/.../passwd"	ou sous Win32
AuthName "Le Domaine Privé"	nom du domaine dans lequel veut entrer l'utilisateur
<Limit GET POST>	contrôle d'accès
require user admin	mot de passe de l'utilisateur <code>admin</code> requis
require valid-user	ou mot de passe d'un utilisateur du fichier <code>passwd</code> requis
</Limit>	

6.1.4 Création des mots de passe

Apache fournit un programme `htpasswd` qui permet d'ajouter un utilisateur et un mot de passe à un fichier de mots de passe :

<code>htpasswd -c passwd toto</code>	crée un fichier nommé <code>passwd</code> contenant l'utilisateur <code>toto</code> et son mot de passe crypté; ce dernier est demandé interactivement.
<code>htpasswd passwd riri</code>	ajoute une entrée pour l'utilisateur <code>riri</code> .

6.1.5 Désauthentification

Netscape et Internet Explorer effaceront le cache d'authentification client s'ils reçoivent une réponse 401. Cela permet de déconnecter un utilisateur, pour le forcer à ré-entrer son nom de compte et son mot de passe. Certains programmeurs l'utilisent pour donner un délai d'expiration, ou alors, fournissent un bouton de déconnexion. Par exemple, l'appel suivant à la fonction PHP Header permet de se désauthentifier :

```
<?php
Header("HTTP/1.0 401 Unauthorized");
?>
```

Cette désauthentification ne fonctionne pas toujours correctement !

6.2 Paramètres de RezUFR

Sur chaque machine du bâtiment 6, un serveur Apache tourne au moyen notamment d'un processus démon nommé `httpd`. Les paramètres actuels dans le domaine `info-ufr.univ-montp2.fr` sont :

- `~/public_html` : le répertoire de vos documents web accessibles depuis n'importe quel serveur http. En effet, `/auto-home/jdupont` est monté sur chaque machine. Vous y créez un sous-répertoire `ISR` pour y stocker les pages et scripts du TP. Les scripts python peuvent être situés n'importe où sous `public_html`, doivent avoir le suffixe `.py` et être **lisibles** par tous (`chmod a+r toto.py`).
- `http://localhost/~pdupont/ISR` : l'URL pour accéder à vos pages et scripts. La machine peut-être :
 - `localhost` : votre machine;
 - `127.0.0.1` : l'adresse IP de votre machine;
 - `a1.info-ufr.univ-montp2.fr` ou `a1` : nom de votre machine obtenue par la commande Unix `hostname`);
 - `a2` ou toute autre machine (`a3`, `a4`, ...) de la même salle.
- **ATTENTION à ne pas mettre `public_html` dans l'URL**
- `~/public_html/ISR/index.html` : page d'accueil par défaut accessible par : `http://machine/~pdupont/ISR`. Si cette page n'existe pas, alors la page `index.py` est recherchée;
- **DROITS** :
 - vos répertoires doivent être traversables et lisibles par tous : `chmod 755 public_html public_html/ISR`;
 - vos fichiers html doivent être lisibles : `chmod 644 public_html/index.html`;
 - vos scripts python doivent être lisibles : `chmod 644 public_html/ISR/hello.py`;
 - vos cgi doivent être lisibles et exécutables : `chmod 755 monrep/moncgi`.

Chapitre 7

Conclusion

Cette brève introduction aux technologies du web n'est pas exhaustive mais son ambition est d'apprendre à analyser une application web dans ses différentes composantes :

- séparation de la forme (styles CSS) et du contenu XHTML (et par la suite XML) ;
- répartition de la logique de l'application entre script côté client et côté serveur ;
- structuration d'un projet complexe nécessitant de multiples pages ;
- réutilisation de modules logiciels en observant certains sites et en picorant des scripts JavaScript.

La division des tâches entre serveur et client est primordiale à comprendre dans un environnement surchargé. En effet, toute tâche de vérification, de calcul ou de contrôle effectué chez le client allègera le travail du serveur et permettra donc une meilleure qualité de service.

Chapitre 8

Erreurs fréquentes

8.1 Javascript

- un élément script n'est jamais vide même si tout le contenu est dans un fichier externe, il faut écrire :
`<script language='javascript' src='tableau.js'></script>`
- JavaScript est sensible à la casse : `new Object()` et pas `new object()` ; `Document` et pas `document` ; `event` et pas `Event` !
- affectation par référence pour les objets complexes, par copie pour les simples ;

8.2 DOM

- tout texte situé entre deux balises est dans un noeud texte et n'est pas la valeur de l'élément ; attention, même un retour à la ligne génère un noeud texte ;
- tout attribut est dans un noeud attribut ;

8.3 Mes erreurs ...

Bibliographie

- [1] *Programmation HTML et Javascript*, de Chaléat, Charnay, Eyrolles (2000), “simple et rapide pour les bases HTML, CSS, JavaScript, 450 pages”
- [2] *Webmaster in a nutshell*, de S. Spainhour, R. Eckstein, O’Reilly (2003), “Manuel de référence rapide pour HTML, CSS, JavaScript, PHP, XML, 550 pages, en anglais”
- [3] *Un site de documentation sur les technologies du web*, <http://www.w3schools.com/>, “Très complet et en anglais (XHTML, CSS, JavaScript) ”
- [4] *Le site du W3C*, <http://www.w3.org/>, “rapports techniques de référence en anglais sur tous les sujets du Web”
- [5] *Spécification HTML*, <http://www.w3.org/TR/REC-html40/>, W3C (1999), “La spécification HTML 4.0 complète en anglais”
- [6] *Spécification DOM HTML*, <http://www.w3.org/TR/DOM-Level-2-HTML/html.html>, W3C (2003), “La spécification du DOM HTML complète en anglais”
- [7] *Référence DOM HTML en JavaScript*, <http://www.w3schools.com/html/dom/>, “référence de l’implémentation du DOM HTML en javascript complète en anglais : Navigator, Window, History, Location, ...”
- [8] *Référence DOM XML en JavaScript*, <http://www.w3schools.com/dom/>, “référence de l’implémentation du DOM XML en javascript complète en anglais : Document”
- [9] *Le site de référence sur CSS 2.1*, <http://www.w3.org/TR/CSS21/cover.html#minitoc>, “complet et en anglais”