Licence 1

Informatique, Développement d'Application

Cours : Introduction au JavaScript

Séquence 3 : JavaScript et les navigateurs

UVS
UNIVERSITÉ VIRTUELLE DU SÉNÉGAL

3. Chapitre III : JavaScript et les navigateurs

Nous passons dans ce chapitre à JavaScript comme utilisé dans les navigateurs Web, communément appelé JavaScript côté client. La plupart des exemples que nous avons vus jusqu'à présent n'a pas de contexte particulier; ce sont des fragments JavaScript qui s'exécutent dans un environnement non spécifié. Ce chapitre fournit ce contexte.

Avant de commencer à parler de JavaScript, il est utile de penser aux pages Web que nous affichons dans les navigateurs Web. Certaines pages présentent des informations statiques et peuvent être appelées des documents. D'autres pages Web ressemblent plus à des applications qu'à des documents. Ces pages peuvent charger dynamiquement de nouvelles informations selon les besoins, elles peuvent être graphiques plutôt que textuelles, et elles peuvent fonctionner hors ligne et enregistrer des données localement pour qu'elles puissent restaurer votre état lorsque vous les visitez à nouveau. D'autres pages Web se situent quelque part au milieu du spectre et combinent les fonctionnalités des documents et des applications.

3.1. JavaScript côté client

L'objet **Window** est le point d'entrée principal de toutes les fonctionnalités et API JavaScript côté client. Il représente une fenêtre ou un cadre de navigateur Web, et vous pouvez vous y référer avec la fenêtre d'identification. L'objet Window définit des propriétés telles que **location**, qui fait référence à un objet Location qui spécifie l'URL actuellement affichée dans la fenêtre et permet à un script de charger une nouvelle URL dans la fenêtre:

```
// Définit la propriété location pour naviguer vers une nouvelle page
Web window.location = "http://www.google.com/";
```

L'objet Window définit également des méthodes telles que **alert ()**, qui affiche un message dans une boîte de dialogue, et **setTimeout ()**, qui enregistre une fonction à appeler après une durée spécifiée:

```
// Attendre 2 secondes afficher « hello world »
setTimeout(function() { alert("hello world"); }, 2000);
```

Notez que le code ci-dessus n'utilise pas explicitement la propriété window. Dans le JavaScript côté client, l'objet Window est également l'objet global. Cela signifie que l'objet Window est en haut de la chaîne de portée et que ses propriétés et méthodes sont effectivement des variables globales et des fonctions globales. L'objet Window a une propriété nommée **window** qui se réfère toujours à lui-même. Vous pouvez utiliser cette propriété si vous avez besoin de faire référence à l'objet fenêtre lui-même, mais il n'est

généralement pas nécessaire d'utiliser la fenêtre si vous souhaitez simplement faire référence aux propriétés d'accès de l'objet fenêtre global.

L'une des propriétés les plus importantes de l'objet Window est **document**: elle fait référence à un objet Document qui représente le contenu affiché dans la fenêtre. L'objet Document possède des méthodes importantes telles que **getElementById ()**, qui renvoie un seul élément de document (représentant une paire de balises HTML ouverte / fermée et tout le contenu entre elles) en fonction de la valeur de son attribut id:

```
// Trouver l'élément avec id = "timestamp"
var timestamp = document.getElementById ("timestamp");
```

L'objet Element renvoyé par getElementById () a d'autres propriétés et méthodes importantes qui permettent aux scripts d'obtenir son contenu, de définir la valeur de ses attributs, et ainsi de suite:

```
// Si l'élément est vide, insérez la date et l'heure actuelle if
(timestamp.firstChild == null){
timestamp.appendChild(document.createTextNode(new Date().toString()));
}
```

Chaque objet Element possède des propriétés style et **className** qui permettent aux scripts de spécifier des styles CSS pour un élément de document ou de modifier les noms de classes CSS qui s'appliquent à l'élément. La définition de ces propriétés CSS modifie la présentation de l'élément de document:

```
// Modifier explicitement la présentation de l'élément de
titre timestamp.style.backgroundColor = "yellow";

// Ou modifiez simplement la classe et laissez la feuille de style spécifier les
détails: timestamp.className = "highlight";
```

Un autre ensemble de propriétés importantes sur les objets Window, Document et Element sont les propriétés du gestionnaire d'événements. Ils permettent aux scripts de spécifier des fonctions qui doivent être appelées de manière asynchrone lorsque certains événements se produisent. Les gestionnaires d'événements permettent au code JavaScript de modifier le comportement des fenêtres, des documents et des éléments qui composent ces documents. Les propriétés du gestionnaire d'événements ont des noms qui commencent par le mot **"on",** et vous pouvez les utiliser comme ceci:

```
// Mettre à jour le contenu de l'élément timestamp lorsque l'utilisateur clique
dessus timestamp.onclick = function () {this.innerHTML = new Date (). ToString (); }
```

L'un des gestionnaires d'événements les plus importants est le gestionnaire **onload** de l'objet Window. Il est déclenché lorsque le contenu du document affiché dans la fenêtre est

stable et prêt à être manipulé. Le code JavaScript est généralement encapsulé dans un gestionnaire d'événement onload.

Exemple :

```
<!DOCTYPE html>
<html>
<head>
  <script>
    window.onload = function () {
      // Afficher le message quand la page est complètement chargée
      alert("La page est chargée");
    };
  </script>
</head>

<body>
</body>
</html>
```

3.2. Manipulation du DOM (Document Object Model)

JavaScript côté client existe pour transformer des documents HTML statiques en applications Web interactives. Manipuler le contenu des pages Web est l'objectif central de JavaScript.

Chaque objet Window a une propriété **document** qui fait référence à un objet Document. L'objet Document représente le contenu de la fenêtre, et c'est l'objet de cette section. L'objet Document n'est cependant pas seul. C'est l'objet central d'une API plus grande, appelée Document Object Model ou DOM, pour représenter et manipuler le contenu d'un document.

3.2.1. Aperçu du DOM

Le DOM (Document Object Model) est l'API de base pour représenter et manipuler le contenu des documents HTML et XML. L'API n'est pas particulièrement compliquée, mais vous devez comprendre un certain nombre de détails architecturaux. Tout d'abord, vous devez comprendre que les éléments imbriqués d'un document HTML ou XML sont représentés dans le DOM en tant qu'arborescence d'objets. La représentation arborescente d'un document HTML contient des nœuds représentant des balises ou des éléments HTML, tels que <body> et <p>, et des nœuds représentant des chaînes de texte. Un document HTML peut également contenir des nœuds représentant des commentaires HTML. Considérez le document HTML simple suivant:

```
<html>
        <head>
                <title>Sample Document</title>
        </head>
        <body>
                <h1>An HTML Document</h1>
                <p>This is a <i>simple</i>
        document.</p> </body>
</html>
```

La représentation DOM de ce document est la suivante :

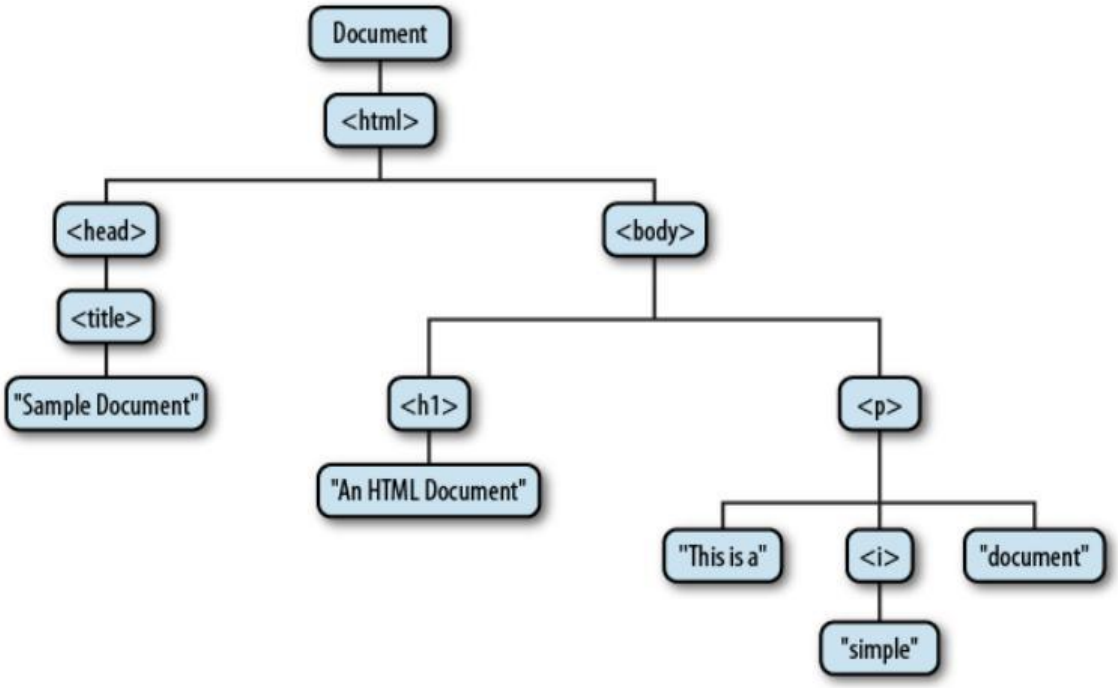

Figure 3.2.1 : Représentation DOM

Il est utile de savoir que DOM emprunte la terminologie des arbres généalogiques. Le nœud directement au-dessus d'un nœud est le parent de ce nœud. Les nœuds un niveau directement au-dessous d'un autre nœud sont les enfants de ce nœud. Les nœuds au même niveau et avec le même parent sont frères et sœurs. L'ensemble des noeuds n'importe quel nombre de niveaux au-dessous d'un autre noeud sont les descendants de ce noeud. Et le parent, le grand-parent et tous les autres nœuds au-dessus d'un nœud sont les ancêtres de ce nœud.

Chaque case de la figure 3.2.1 est un nœud du document et est représentée par un objet Node.

Notez que la figure contient trois types différents de nœuds. À la racine de l'arborescence se trouve le nœud Document qui représente le document entier. Les nœuds qui représentent

les éléments HTML sont des nœuds d'élément et les nœuds qui représentent le texte sont des nœuds de texte**. Document, Element et Text** sont des sous-classes de la classe Node. Document et Element sont les deux classes DOM les plus importantes.

Node et ses sous-types forment la hiérarchie de types illustrée à la figure 3.2.2. Notez qu'il existe une distinction formelle entre les types génériques Document et Element, et les types HTMLDocument et HTMLElement. Le type de document représente un document HTML ou XML et la classe Element représente un élément d'un tel document. Les sous-classes HTMLDocument et HTMLElement sont spécifiques aux documents et éléments HTML. Dans ce livre, nous utilisons souvent les noms de classes génériques Document et Element, même en référence à des documents HTML.

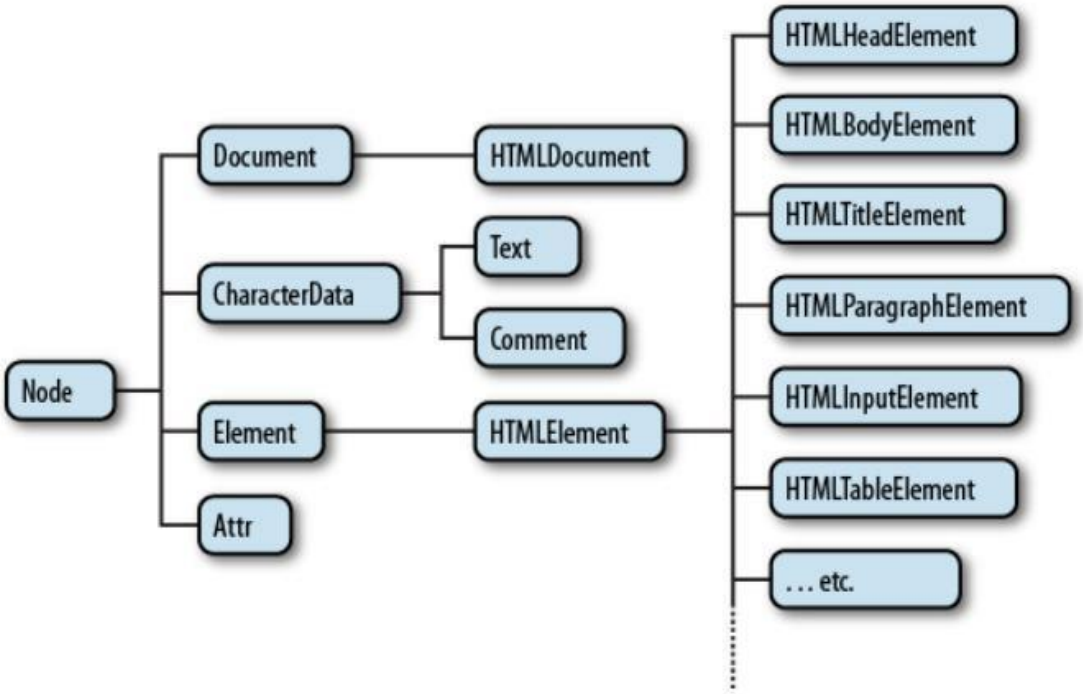

Figure 3.2.2 : Hiérarchie partielle des classes de Node de Document

3.2.2. Sélection d'éléments de Document

La plupart des programmes JavaScript côté client fonctionnent en manipulant d'une manière ou d'une autre un ou plusieurs éléments du document. Lorsque ces programmes démarrent, ils peuvent utiliser la variable **document** pour faire référence à l'objet Document.

Cependant, pour manipuler les éléments du document, ils doivent obtenir ou sélectionner les objets Element qui se réfèrent à ces éléments. Le DOM définit un certain nombre de façons de sélectionner des éléments; vous pouvez interroger un document pour un ou plusieurs éléments:

- avec un attribut **id** défini;
- avec un attribut **name** défini;
- avec le nom d'une balise défini;
- avec la ou les classes CSS;

correspondant à un sélecteur CSS

3.2.2.1. Sélection des éléments par ID

Tout élément HTML peut avoir un attribut id. La valeur de cet attribut doit être unique dans le document: aucun élément d'un même document ne peut avoir le même ID. Vous pouvez sélectionner un élément basé sur cet ID unique à l'aide de la méthode getElementById () de l'objet Document.

```
var section1 = document.getElementById("section1");
```

C'est la manière la plus simple et la plus utilisée de sélectionner des éléments. Si votre script doit manipuler un certain ensemble spécifique d'éléments de document, attribuez-lui ces attributs, et recherchez les objets Element en utilisant cet identifiant.

3.2.2.2. Sélection des éléments par nom

L'attribut HTML **name** était à l'origine destiné à affecter des noms aux éléments de formulaire, et la valeur de cet attribut est utilisée lorsque des données de formulaire sont soumises à un serveur. Comme l'attribut id, name attribue un nom à un élément. Contrairement à id, cependant, la valeur d'un attribut name n'a pas besoin d'être unique: plusieurs éléments peuvent avoir le même nom, ce qui est commun dans le cas des boutons radio et des cases à cocher dans les formulaires. De même, contrairement à id, l'attribut name n'est valide que sur une poignée d'éléments HTML, y compris les formulaires, les éléments de formulaire, les éléments <iframe> et <img>.

Pour sélectionner des éléments HTML en fonction de la valeur de leurs attributs de nom, vous pouvez utiliser la méthode getElementsByName () de l'objet Document:

```
var radiobuttons = document.getElementsByName("favorite_color");
```

getElementsByName () est défini par la classe HTMLDocument et non par la classe Document. Il n'est donc disponible que pour les documents HTML et non pour les documents XML. Il renvoie un objet **NodeList** qui se comporte comme un tableau en lecture seule des objets Element. Dans IE, getElementsByName () retournera également les éléments qui ont un attribut id avec la valeur spécifiée. Pour des raisons de compatibilité, veillez à ne pas utiliser la même chaîne que le nom et l'ID.

La définition de l'attribut name d'un élément <form>, <img>, <iframe>, <applet>, <embed> ou <object> crée une propriété de l'objet Document dont le nom est la valeur de l'attribut (en supposant, bien entendu, que le document ne possède pas déjà une propriété portant ce nom). S'il n'y a qu'un seul élément avec un nom donné, la valeur de la propriété de document créée automatiquement est l'élément lui-même. S'il existe plusieurs éléments, la

valeur de la propriété est un objet NodeList qui agit comme un tableau d'éléments. Cela signifie que certains éléments peuvent être sélectionnés par nom en utilisant simplement le nom en tant que propriété Document:

```
// Récupère l'objet Element pour l'élément <form name = "shipping_address">
var form = document.shipping_address;
```

3.2.2.3. Sélection des éléments en utilisant la balise

Vous pouvez sélectionner tous les éléments HTML ou XML d'un type spécifié à l'aide de la méthode getElementsByTagName () de l'objet Document. Pour obtenir un objet semblable à un tableau en lecture seule contenant les objets Element pour tous les éléments d'un document, par exemple, vous pouvez écrire:

```
var spans = document.getElementsByTagName("span");
```

Comme getElementsByName (), getElementsByTagName () retourne un objet NodeList. Les éléments du NodeList retourné sont dans l'ordre du document, vous pouvez donc sélectionner le premier élément <p> d'un document comme ceci:

```
var firstpara = document.getElementsByTagName("p")[0];
```

Les balises HTML ne sont pas sensibles à la casse, et lorsque getElementsByTagName () est utilisé sur un document HTML, il effectue une comparaison de nom de balise insensible à la casse. La variable **spans** ci-dessus, par exemple, inclura tous les éléments écrits comme ceci : <SPAN>.

La classe Element définit également une méthode getElementsByTagName (). Il sélectionne uniquement les éléments qui sont les descendants de l'élément sur lequel il est appelé. Donc, pour trouver tous les éléments dans le premier élément <p> d'un document, vous pouvez écrire:

```
var firstpara = document.getElementsByTagName("p")[0];
var firstParaSpans = firstpara.getElementsByTagName("span");
```

Pour des raisons historiques, la classe HTMLDocument définit des propriétés de raccourci pour accéder à certains types de nœuds. Les propriétés images, formulaires et liens, par exemple, font référence à des objets qui se comportent comme des tableaux en lecture seule des éléments <img>, <form> et <a> (mais uniquement des balises <a> qui ont un attribut href). Ces propriétés font référence aux objets **HTMLCollection**, qui ressemblent beaucoup aux objets **NodeList**, mais ils peuvent en outre être indexés par l'ID ou le nom de l'élément. Auparavant, nous avons vu comment vous pourriez vous référer à un élément nommé <form> avec une expression comme celle-ci:

document.shipping_address

Avec la propriété **document.forms**, vous pouvez également vous référer plus spécifiquement au nom (ou à l'id) comme ceci:

document.forms.shipping_address;

3.2.2.4. Sélection d'éléments par classe CSS

L'attribut class d'un HTML est une liste de zéro ou plusieurs identifiants séparés par des espaces. Il décrit un moyen de définir des ensembles d'éléments de documents associés: tous les éléments qui ont le même identificateur dans leur attribut de classe font partie du même ensemble. **class** étant un mot réservé en JavaScript, le JavaScript côté client utilise la propriété **className** pour contenir la valeur de l'attribut de classe HTML.

HTML5 définit une méthode, **getElementsByClassName (),** qui nous permet de sélectionner des ensembles d'éléments de document basés sur les identifiants de leur attribut de classe.

Comme getElementsByTagName (), getElementsByClassName () peut être invoqué à la fois sur des documents HTML et des éléments HTML, et renvoie une NodeList en direct contenant tous les descendants correspondants du document ou de l'élément. getElementsByClassName () prend un seul argument de chaîne, mais la chaîne peut spécifier plusieurs identifiants séparés par des espaces. Seuls les éléments qui incluent tous les identifiants spécifiés dans leur attribut de classe sont mis en correspondance. L'ordre des identifiants n'a pas d'importance. Notez que l'attribut class et les méthodes getElementsByClassName () séparent les identifiants de classe d'espaces, pas de virgules. Voici quelques exemples de getElementsByClassName ():

```
// Trouver tous les éléments qui ont "warning" dans leur attribut de
classe var warnings = document.getElementsByClassName ("warning");

// Trouver tous les descendants de l'élément nommé "log" qui ont la classe
// "error" et la classe "fatal"
var log = document.getElementById ("log");
var fatal = log.getElementsByClassName ("erreur fatal");
```

3.2.2.5. Sélection d'éléments avec des sélecteurs CSS

Les feuilles de style CSS ont une syntaxe très puissante, appelée sélecteurs, pour décrire des éléments ou des ensembles d'éléments dans un document. Les éléments peuvent être décrits par ID, nom de tag ou classe:

```
#nav // Un élément avec id = "nav"
div // Tout élément <div>
.warning // Tout élément avec "warning" dans son attribut de classe
```

Plus généralement, les éléments peuvent être sélectionnés en fonction des valeurs d'attribut:

p [lang = "fr"] // Un paragraphe rédigé en français: <p lang = "fr"> * [name = "x"] // Tout élément avec un attribut name = "x"

Ces sélecteurs de base peuvent être combinés:

span.fatal.error // Tout avec "warning" et "fatal" dans sa classe warning
span [lang = "fr"] // Tout avertissement en français

Les sélecteurs peuvent également spécifier la structure du document:

#log span // N'importe quel enfant de l'élément avec id = "log"
#log> span // Tout descendant de l'élément avec id = "log"
body> h1: first-child // Le premier <h1> enfant du <body>

Les sélecteurs peuvent être combinés pour sélectionner plusieurs éléments ou plusieurs ensembles d'éléments:

div, #log // Tous les éléments plus l'élément avec id = "log"

Comme vous pouvez le voir, les sélecteurs CSS permettent de sélectionner des éléments de toutes les façons décrites ci-dessus: par ID, par nom, par nom de balise, et par nom de classe. Parallèlement à la standardisation des sélecteurs CSS3, un autre standard W3C, appelé "API de sélection", définit les méthodes JavaScript permettant d'obtenir les éléments correspondant à un sélecteur donné. La clé de cette API est la méthode Document **querySelectorAll ()**. Il prend un seul argument de chaîne contenant un sélecteur CSS et retourne un NodeList qui représente tous les éléments du document qui correspondent au sélecteur. Contrairement aux méthodes de sélection d'éléments décrites précédemment, la liste de nœuds retournée par querySelectorAll () n'est pas active: elle contient les éléments qui correspondent au sélecteur au moment de l'appel de la méthode, mais elle ne se met pas à jour lorsque le document change. Si aucun élément ne correspond, querySelectorAll () renvoie un NodeList vide. Si la chaîne de sélecteur n'est pas valide, querySelectorAll () renvoie une exception.

En plus de querySelectorAll (), l'objet document définit également querySelector (), qui est similaire à querySelectorAll (), mais renvoie uniquement le premier élément de correspondance (dans l'ordre du document) ou null s'il n'y a pas d'élément correspondant.

3.2.3. Contenu d'un élément

Reprenons la figure 3.2.1 (voir la figure 3.2.3 ci-dessous) qui illustrait un arbre de Document en début de paragraphe. La question qu'on peut se poser est a suivant : quel est le type du contenu de l'élément <p>.
On peut y répondre en disant que le contenu de l'élément <p> est :

Un texte HTML « This is a simple document.
» Un texte brut « This is a simple document. »

un nœud Text, un nœud Element ayant un nœud enfant de type Text et un autre noeud Text.

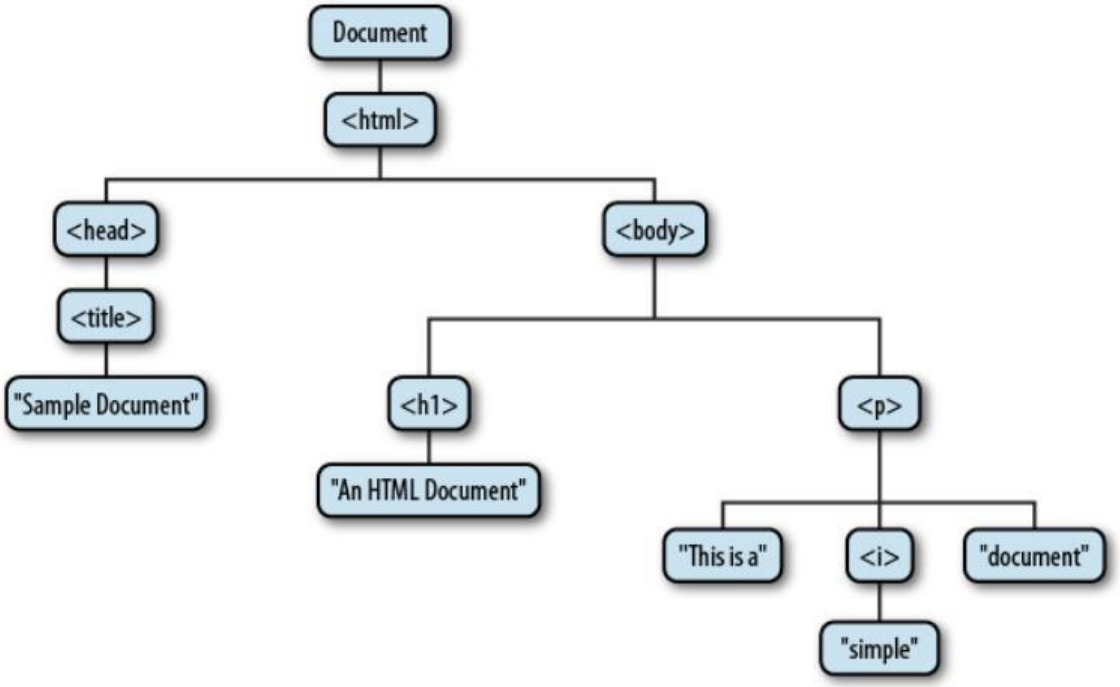

Figure 3.2.3 : Représentation DOM

Chacune de ces réponses est valide et chaque réponse est utile à sa manière. Les sections qui suivent expliquent comment travailler avec la représentation HTML, la représentation en texte brut et la représentation arborescente du contenu de l'élément.

3.2.3.1. Contenu de l'élément au format HTML

La propriété **innerHTML** d'un élément renvoie le contenu de cet élément en tant que chaîne de balises. La définition de cette propriété sur un élément appelle l'analyseur du navigateur Web et remplace le contenu actuel de l'élément avec une représentation analysée de la nouvelle chaîne. Malgré son nom, innerHTML peut être utilisé avec des éléments XML et HTML.

HTML5 standardise également une propriété nommée outerHTML. Lorsque vous utilisez outerHTML, la chaîne de balises HTML ou XML renvoyée inclut les balises ouvrantes et fermantes de l'élément sur lequel vous l'avez interrogé. Lorsque vous définissez outerHTML sur un élément, le nouveau contenu remplace l'élément lui-même. outerHTML est défini uniquement pour les nœuds Element, mais pas sur ceux de type Document.

Une autre caractéristique introduite par IE et normalisée par HTML5 est la méthode **insertAdjacentHTML** (), qui vous permet d'insérer une chaîne de code HTML arbitraire "adjacent" à l'élément spécifié. La balise est passée comme deuxième argument à cette méthode, et la signification précise de "adjacent" dépend de la valeur du premier argument. Ce premier argument doit être une chaîne avec l'une des valeurs

"**beforebegin**", "**afterbegin**", "**beforeend**" ou "**afterend**". Ces valeurs correspondent aux points d'insertion illustrés à la figure 3.2.5.1.

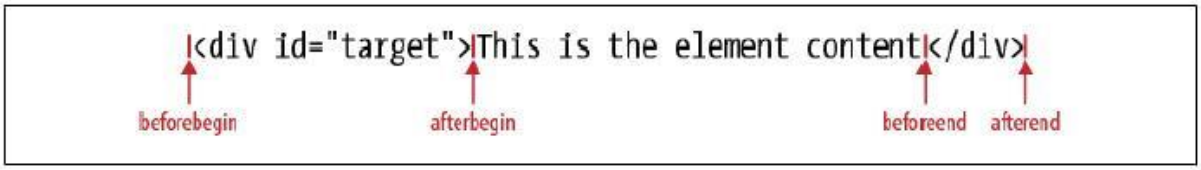

Figure 3.2.3.1 : Les points d'insertion pour **insertAdjacentHTML**

3.2.3.2. Contenu de l'élément en texte brut

Parfois, vous souhaiteriez interroger le contenu d'un élément en texte brut ou insérer du texte en clair dans un document .
La façon standard de faire cela est avec la propriété **textContent** du noeud:

```
var para = document.getElementsByTagName("p")[0]; // Premier <p> dans le
//documents
var text = para.textContent; // L e texte est "This is a simple document."
para.textContent = "Hello World!"; // Modifier le contenu
```

La propriété **textContent** est prise en charge par tous les navigateurs actuels sauf IE. Dans IE, vous pouvez utiliser la propriété Element **innerText** à la place. Microsoft a introduit **innerText** dans IE4, et il est supporté par tous les navigateurs actuels sauf Firefox.
Les propriétés **textContent** et **innerText** sont assez similaires que vous pouvez généralement les utiliser de façon interchangeable.

3.2.3.3. Contenu de l'élément en tant que nœuds de type Text

Une autre façon de travailler avec le contenu d'un élément est une liste de nœuds enfants, chacun pouvant avoir son propre ensemble d'enfants.
Dans les documents XML, vous devez également être préparé pour gérer les nœuds **CDATASection** , ils sont un sous-type de texte et représentent le contenu des sections CDATA.

L'exemple suivant montre une fonction **textContent ()** qui traverse récursivement les enfants d'un élément et concatène le texte de tous les nœud Text descendants. Afin de comprendre le code, il faut rappeler que la propriété **nodeValue** (définie par le type Node) contient le contenu d'un noeud Text.

```
// Renvoie le contenu texte brut de l'élément e, en parcourant tous les
éléments enfants.
```

```
// Cette méthode fonctionne comme la propriété textContent vue dans la section
précédente
function myTextContent (e) {
    var child, type, s = ""; // s contient le texte de tous les
    enfants for (child = e.firstChild; child! = null; child =
    child.nextSibling) { type = child.nodeType;
    if (type === 3 || type === 4) // Noeuds Text et
    CDATASection s + = child.nodeValue;
    else if (type === 1) // Recursivité pour les noeuds
    Element s + = textContent (enfant);
    }
    return s;
}
```

3.3. Manipulation du CSS

Le Cascading Style Sheets (CSS) est une norme pour spécifier la présentation visuelle de Documents HTML. Il permet de spécifier précisément les polices, les couleurs, les marges, l'indentation, les bordures, et même la position des éléments du document. Manipuler le CSS en JavaScript permet une variété d'effets visuel intéressant: vous pouvez créer des transitions animées où le contenu du document "glisse" à partir de la droite par exemple; ou créer une liste de plan en expansion et effondrement dans lequel l'utilisateur peut contrôler la quantité d'informations affichées. Quand il a été introduit, les effets visuels scénarisés comme ceux-ci étaient révolutionnaires. Les techniques JavaScript et CSS qui les a produites étaient dénommé Dynamic HTML ou DHTML.

3.3.1. Manipulation avec l'attribut style

Le moyen le plus simple de créer un script CSS consiste à modifier l'attribut de style d'un élément du document. Comme la plupart des attributs HTML, style est également une propriété de l'objet **Element** et vous pouvez le manipuler en JavaScript. La propriété style est inhabituelle, sa valeur n'est pas une chaîne, mais un objet **CSSStyleDeclaration**. Les propriétés JavaScript de cet objet de style représente les propriétés CSS spécifiées par l'attribut de style HTML.

Pour rendre le texte d'un élément grand, gras et bleu, par exemple, vous pouvez utiliser le code suivant pour définir les propriétés JavaScript correspondant au poids de la police et les propriétés de style de couleur:

```
e.style.fontSize = "24pt";
e.style.fontWeight = "bold";
e.style.color = "blue";
```

De nombreuses propriétés de style CSS, telles que **font-size**, contiennent des traits d'union dans leurs noms. En JavaScript, un trait d'union est interprété comme un signe moins, il n'est donc pas possible d'écrire une expression comme:

e.style.font-size = "24pt"; // Erreur de syntaxe!

Par conséquent, les noms des propriétés de l'objet CSSStyleDeclaration sont légèrement différent des noms des propriétés CSS réelles. Si un nom de propriété CSS contient un ou plusieurs traits d'union, le nom de la propriété CSSStyleDeclaration est formé en supprimant les traits d'union et en capitalisant la lettre immédiatement après chaque trait d'union. Ainsi, la propriété CSS **border-left-width** est accessible via la propriété JavaScript **borderLeftWidth**, et vous pouvez accéder à la propriété CSS font-family ave le code suivant:

e.style.fontFamily = "sans-serif";

Lorsque vous travaillez avec les propriétés de style de l'objet CSSStyleDeclaration, souvenez-vous que toutes les valeurs doivent être spécifiées en tant que chaînes de caractères. Dans une feuille de style ou un attribut de style, vous pouvez écrire:

position: absolute; font-family: sans-serif; background-color: #ffffff;

Pour faire la même chose pour un élément e avec JavaScript, vous devez citer toutes les valeurs:

```
e.style.position = "absolute";
e.style.fontFamily = "sans-serif";
e.style.backgroundColor = "#ffffff";
```

Notez que les points-virgules sortent des chaînes. Ce sont juste des points-virgules JavaScript normaux ; les points-virgules que vous utilisez dans les feuilles de style CSS ne sont pas requis dans les valeurs de chaîne que vous définissez avec JavaScript.
De plus, rappelez-vous que toutes les propriétés de positionnement nécessitent des unités. Ainsi, ce n'est pas correct pour définir la propriété **left** comme ceci:

```
e.style.left = 300; // Incorrect: c'est un nombre, pas une chaîne
e.style.left = "300"; // Incorrect: les unités sont manquantes
```

La bonne façon de définir la valeur de la propriété left d'un élément e à 300 pixels est:

```
e.style.left = "300px";
```

3.3.2. Animations CSS

L'une des utilisations les plus courantes de CSS en JS est de produire des effets visuels animés.
Cela peut être réalisé en utilisant **setTimeout ()** ou **setInterval ()** à plusieurs reprises en invoquant une fonction qui modifie le style d'un élément.

La méthode **setTimeout** () appelle une fonction ou évalue une expression après un nombre spécifié de millisecondes. La fonction est seulement exécutée une fois.

La méthode **setInterval** () appelle une fonction ou évalue une expression à des intervalles spécifiés en millisecondes. La méthode **setInterval** () continuera d'appeler la fonction jusqu'à ce que **clearInterval** () soit appelé ou que la fenêtre soit fermée.

L'exemple suivant démontre leur utilisation.

```
        function moveToRight() {
            var elem = document.getElementById("animatedElem");
            var actualPosition ="init";

            // ne plus afficher le box après 2000 milliscondes (2
            secondes) setTimeout(function(){
                elem.style.display = "none";
            },2000);

            animate ();

            function animate () {
                if(actualPosition=="init"){
                    actualPosition = "left";
                }
                else if(actualPosition=="left") {
                    // postionner au milieu avec la combinaison de margin :
auto
                    actualPosition = "none";
                }else {
                    actualPosition = "right";
                }

                elem.style.cssFloat = actualPosition;

                // ne plus exécuter la fonction animate
                // le box est à droite
                if(actualPosition != "right"){
                    // exécuter la fonction animate toutes les 500
millisecondes
                    setInterval(animate, 500);
                }
            }
        }
```

Dans cet exemple, l'élément est programmé pour être supprimé après 2000 millisecondes (2s) avec **setTimeout .**

Par la suite avec la fonction **setInterval** , l'élément est déplacé de la gauche vers la droite toutes les 500 millisecondes avec la propriété **cssFloat** .

Ce qui se résume en une animation où à 0ms l'élément est déplacé à gauche, à 500ms au milieu, à 1000ms à droite et supprimé à 2000ms.

3.4. Gestion des événements

Les programmes JavaScript côté client utilisent un modèle de programmation asynchrone piloté par des événements. Dans ce style de programmation, le navigateur Web génère un événement chaque fois que quelque chose d'intéressant arrive au document ou au navigateur ou à certains élément ou objet associé à celui-ci. Par exemple, le navigateur Web génère un événement quand il a fini de charger un document, lorsque l'utilisateur déplace la souris sur un lien hypertexte, ou lorsque l'utilisateur tape une touche sur le clavier. Si une application JavaScript veut utiliser
un type particulier d'événement, il peut enregistrer une ou plusieurs fonctions à invoquer dès que les événements de ce type se produisent.

3.4.1. Liste des événements

Il existe de nombreux événements, tous plus ou moins utiles. Parmi les évènements utiles que nous n'aborderons pas, sachez qu'il en existe des spécifiques pour les plateformes mobiles (smartphones, tablettes, etc...).
Voici la liste des événements principaux, ainsi que les actions à effectuer pour qu'ils se déclenchent :

Nom de l'événement	Action pour le déclencher
click	Cliquer (appuyer puis relâcher) sur l'élément
dblclick	Double-cliquer sur l'élément
mouseover	Faire entrer le curseur sur l'élément
mouseout	Faire sortir le curseur de l'élément
mousedown	Appuyer (sans relâcher) sur le bouton gauche de la souris sur l'élément
mouseup	Relâcher le bouton gauche de la souris sur l'élément
mousemove	Faire déplacer le curseur sur l'élément
keydown	Appuyer (sans relâcher) sur une touche de clavier sur l'élément
keyup	Relâcher une touche de clavier sur l'élément
keypress	Frapper (appuyer puis relâcher) une touche de clavier sur l'élément
focus	« Cibler » l'élément

Nom de l'événement	Action pour le déclencher
blur	Annuler le « ciblage » de l'élément
change	Changer la valeur d'un élément spécifique aux formulaires (input,checkbox, etc.)
input	Taper un caractère dans un champ de texte (son support n'est pas complet sur tous les navigateurs)
select	Sélectionner le contenu d'un champ de texte (input,textarea, etc.)
submit	Envoyer un formulaire
reset	Réinitialiser un formulaire

3.4.2. Utiliser les événements

Commençons par l'événement click sur un simple.

Cliquez-moi !

Comme vous pouvez le constater, il suffit de cliquer sur le texte pour que la boîte de dialogue s'affiche. Afin d'obtenir ce résultat nous avons ajouté à notre un attribut contenant les deux lettres **« on »** et le nom de notre événement « click » ; nous obtenons donc « onclick ».

Cet attribut possède une valeur qui est un code JavaScript, vous pouvez y écrire quasiment tout ce que vous souhaitez, mais tout doit tenir entre les guillemets de l'attribut.

3.4.3. Les événements au travers du DOM

Afin de pouvoir séparer un peu plus le traitement d'un événement du code HTML auquel il s'applique, il est possible d'accéder aux événements en ne modifiant que le code Javascript. Si un élément a été précédemment identifié (par une méthode getElementById, par exemple), alors l'événement visé est accessible comme une simple propriété de l'objet élément.
L'exemple suivant montre comment utiliser une propriété de l'élément pour définir un événement.

Exemple avec un événement souris.

et le code JavaScript associé :

```
var elm = document. getElementById('exemple1');

elm.onmouseover = function (){
        this .style.fontWeight="bold" ;
        this .style.color="red" ;
}

elm. onmouseout = normal;

function normal (event){
        this .style.fontWeight="normal" ;
        this .style.color="" ;
}
```

Il est à remarquer qu'on affecte à la propriété **onmouseout** la valeur **normal**, autrement dit, le nom de la fonction sans les parenthèses. Si on avait écrit **normal** (), au moment de l'affectation de l'événement à l'élément, la fonction aurait été évaluée.