

IUT Lyon 1

IUT A - Année Spéciale

Web Client Riche

Client APIs

2. Client APIs

Plan de la séance

- 3.1 DOM - Qu'est-ce ?
 - 3.1.1 Qu'est-ce que le DOM ?
 - 3.1.2 A quoi sert le DOM ?
 - 3.1.3 Les spécifications
- 3.2 DOM Core
 - 3.2.1 Structure d'un document HTML
 - 3.2.2 L'interface Node
 - 3.2.3 L'objet Document
 - 3.2.4 L'interface Node (le retour)
 - 3.2.5 L'objet DocumentFragment
 - 3.2.6 L'objet Text
 - 3.2.7 L'objet Element
- 3.3 DOM HTML
 - 3.3.1 L'interface HTMLDocument
 - 3.3.2 L'interface HTMLElement
- 3.4 DOM Events
 - 3.4.1 Événements HTML 4.0 vs. DOM
 - 3.4.2 Déroulement des événements
 - 3.4.3 Divers types d'événements
 - 3.4.4 Exercices
- 3.5 AJAX
 - 3.5.1 Qu'est-ce qu'AJAX ?
 - 3.5.2 L'objet XMLHttpRequest
 - 3.5.3 Historique et spécifications
 - 3.5.4 Un problème de sécurité
 - 3.5.5 CORS
 - 3.5.6 Exemples de requêtes
 - 3.5.7 Exercice
- 3.6 Autres APIs
 - 3.6.1 APIs client
 - 3.6.2 Services

2.1 DOM - Qu'es acò ?

2.1.1 Qu'est-ce que le DOM ?

Le DOM est un ensemble d'interfaces (*ou APIs*), standardisé par le W3C, indépendant de tout langage ou plateforme, et permettant l'accès au contenu, à la structure et au style de documents HTML ou XML.

Via le DOM, le document peut être traité, et le résultat du traitement peut être réincorporé dans la page présentée.

Il existe un assez grand nombre de spécifications que l'on peut regrouper par familles :

- DOM Core - objets et méthodes génériques de navigation et de modification d'un document,
- DOM HTML - objets et méthodes spécifiques aux documents HTML,
- DOM Events - traitement des événements, dont notamment les interactions avec l'utilisateur,
- DOM Style - lecture et modification du style des éléments.

2.1.2 A quoi sert le DOM ?

Les pages web comportant une certaine interactivité sont apparues avec Javascript (*Netscape 2.0, fin 1995*) et JScript (*Internet Explorer 3.0, 1996*) moyennant souvent d'improbables contorsions (*browser sniffing*) car leurs APIs respectives comportaient de nombreuses différences.

En 1997 apparaît la possibilité de modifier le document alors qu'il est affiché par le navigateur. Cette technique connue sous le nom de **DHTML** (*Dynamic HTML*) constitue une combinaison de HTML, CSS et Javascript, mais nécessite toujours du code spécifique en fonction du navigateur.

Fin 1998, le **W3C** (*World Wide Web Consortium*) a recommandé une interface standard connue sous le nom de **DOM Level 1**, afin de faciliter le développement de pages dynamiques en limitant les problèmes d'interopérabilité entre navigateurs.

Le DOM est donc une API standard pour la modification dynamique de documents HTML, issue d'un organisme indépendant regroupant les acteurs concernés, permettant de développer du code directement compatible d'un navigateur à l'autre.

2.1.3 Les spécifications

L'API DOM est décrite par de nombreuses spécifications [DOM specs] dont la première [DOM Level 1] a été recommandée par le W3C en 1998. Elle comporte deux chapitres : [DOM Core] (*fonctionnalités génériques*), et [DOM HTML] (*spécificités HTML*).

[DOM Level 2] est un ensemble de spécifications recommandées en novembre 2000. Les avancées majeures concernent la gestion des événements (*DOM Level 2 Events*), et des documents de style (*DOM Level 2 Style*).

Le niveau d'implémentation de l'API DOM Level 2 est à l'heure actuelle plutôt satisfaisante. La suite est plus confuse.

Certains constituants de [DOM Level 3] ont été recommandés entre 2003 et 2004, d'autres sont toujours à l'état de brouillons (*Draft*) comme [DOM Level 3 Events], et il existe déjà une recommandation de [DOM Level 4].

Pour finir, certains comme [mozilla] ont parfois recommandé de consulter plutôt le [DOM Living Standard] édité par le WHATWG (*Web Hypertext Application Technology Working Group*) qui consolide l'état de l'art à partir des différentes spécifications en cours de validité.

N.B. Dans la suite du cours, les fonctionnalités du DOM seront présentées en mentionnant dans la mesure du possible la spécification dont elles sont issues.

2.2 DOM Core

2.2.1 Rappel : structure d'un document HTML

Un document HTML est la [sérialisation] d'une arborescence d'éléments qui se matérialise sous la forme de balises imbriquées (*dont certaines peuvent être implicites*).

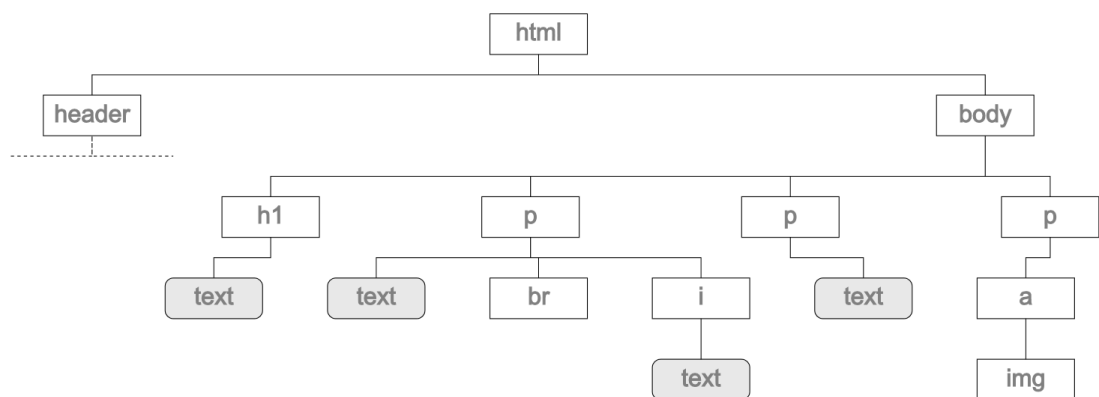
```
<!DOCTYPE html>
<title>Structure d'un document html</title>
<meta charset='utf-8'>
<style>
body { padding: 0 1em; font-family: 'Open Sans', Arial, sans-serif; color: #808080; }
</style>

<h1>Structure d'un document html</h1>
<p>Voici un exemple de document html5 au contenu minimaliste...<br>
<i>(cf. code source - remarquer les balises manquantes i.e. implicites)</i>

<p>La conformité de ce document aux standards HTML et CSS peut être vérifiée...

<p><a href="http://validator.w3.org/check?uri=referer">
</a>
```


Exemple de document html5 (code source)



Représentation d'une partie de l'arborescence du document html5 de la figure précédente.

[démonstration]

N.B. les attributs aussi ont leur place dans l'arbre, mais ne sont pas représentés sur la figure ci-dessus.

 Rappel : les outils pour développeurs de la plupart des navigateurs permettent de visualiser et de travailler avec cet arbre.

Noeuds de l'arbre

Les noeuds (*Nodes*) de l'arbre peuvent être de différente nature :

Document

représente le noeud racine de l'arbre.

Element

correspond à tous les noeuds matérialisés par une balise (*a, p, em, span ou div sont autant d'éléments*).

Text

est un noeud terminal qui matérialise le contenu textuel d'un élément.

L'API DOM Core fournit des objets (*Document*, *Element*, *Text*) pour chacun de ces types de noeuds, ainsi qu'un objet *Node* qui regroupe les attributs (*au sens objet*) communs à tous les types, et un objet *Attribute* pour représenter les attributs (*au sens HTML*) des éléments.

Ces divers objets possèdent les méthodes nécessaires pour naviguer dans l'arbre et le modifier dynamiquement alors qu'il est affiché par le navigateur.

2.2.2 L'interface Node

Tous les objets correspondant à des noeuds de l'arbre (*Document*, *Element*, *Text*...) implémentent cette interface. Ils exposent notamment les attributs suivants : [\[WHATWG Node\]](#) [\[DOM3 Node\]](#)

Introspection du noeud :

```
DOMString    nodeName;      // DOM1 type de l'élément (i.e. nom de la balise)
DOMString    nodeValue;     // DOM1 contenu d'un noeud texte
unsigned short    nodeType;  // DOM1 code donnant la nature du noeud
NamedNodeMap    attributes;  // DOM1 liste des attributs du noeud
DOMString      textContent;  // DOM3 concaténation des nodeValue des enfants
```

[démonstration]

Exemple d'utilisation :

```
<div><span>nodeName : </span><span id="nodeName"> </span></div>
<div><span>nodeValue : </span><span id="nodeValue"></span></div>
<div><span>nodeType : </span><span id="nodeType"> </span></div>
<script>
  ['nodeName', 'nodeValue', 'nodeType'].forEach( function(prop) {
    window[prop].firstChild.nodeValue = e.target[prop];
  });
</script>
```

☞ Parmi les attributs ci-dessus, *nodeValue* et *textContent* sont les seuls dont il soit possible de modifier la valeur. Les autres sont en lecture seule.

Accès aux parents :

```
Node    parentNode;      // DOM1 noeud parent
Document    ownerDocument; // DOM2 document parent
```

Parcours de l'arbre :

```
NodeList    childNodes;      // DOM1 liste des noeuds enfants
Node    firstChild;          // DOM1 premier noeud enfant (dans l'ordre du document)
Node    lastChild;           // DOM1 dernier noeud enfant
Node    previousSibling;     // DOM1 frère aîné (noeud précédent avec le même parent)
Node    nextSibling;         // DOM1 frère cadet (noeud suivant avec le même parent)
```

De proche en proche, ces attributs permettent de parcourir le document :

```
(function process_tree(children) {
  for ( var n = 0; n < children.length; n++ ) {
    if ( children[n].nodeType !== 1 ) continue; // ELEMENT_NODE
    console.log(children[n].nodeName);
    process_tree(children[n].childNodes);
  }
})(document.childNodes);
```

[démonstration]

☞ En phase de développement il est souvent utile de prévoir un compteur global pour éviter les boucles de récursion infinies...

2.2.3 L'objet Document

L'objet Document correspond au **noeud** racine d'un document et implémente l'interface Node.

Il possède un attribut de confort qui donne directement accès à son **élément** racine :

```
Element documentElement; // DOM1 élément racine
```

Dans le contexte d'un navigateur, on obtient l'objet Document qui représente le document affiché via la variable globale document :

```
console.log(document.nodeType); // 9 # DOCUMENT_NODE
```

L'extrait ci-dessous détermine que l'élément racine est le second enfant du document :

```
console.log(document.documentElement.nodeName); // HTML
console.log(document.childNodes); // NodeList [<!DOCTYPE html>, html]
console.log(document.documentElement == document.childNodes[1]); // true
```

[démonstration]

🔧 Où l'on comprend la différence entre le **noeud** racine (*document*) et l'**élément** racine (*document.documentElement*), et la raison d'être du premier.

L'objet Document expose des méthodes qui permettent de récupérer un noeud ou une liste de noeuds, décrites ci-après.

Méthodes de requêtage de l'arbre

```
NodeList getElementsByTagName(in DOMString tagname); // DOM1 via le nom de balise...
Element getElementById(in DOMString elementId); // DOM2 via l'identifiant...
```

Le programme ci-dessous affiche le contenu de toutes les balises script de la page :

```
<pre id="dropzone"></pre>
<script>
  var scripts = document.getElementsByTagName('script');
  for ( var n=0; n < scripts.length; n++ ) {
    dropzone.innerHTML += scripts[n].textContent;
  }
</script>
```

[démonstration]

🔧 les spécifications de **HTML5** préconisent que les éléments identifiés soient directement accessibles par leur nom dans l'espace global. On a donc (*cf. ci-dessus*) :

```
window.dropzone === document.getElementById('dropzone'); // true
```

En général l'objet Document possède également des méthodes issues de spécifications non finalisées, ou de spécifications autres que le DOM :

Nouvelles méthodes de requêtage de l'arbre

```
HTMLCollection getElementsByClassName(DOMString classNames); // DOM4WHATWG
Element? querySelector(DOMString selectors); // Selectors-API
NodeList querySelectorAll(DOMString selectors); // Selectors-API
```

[démonstration]

La première permet de récupérer des éléments en fonction de la valeur de leur attribut class, les deux suivantes utilisent un sélecteur CSS.

N.B. `querySelector` renvoie toujours un seul élément (*au plus*) tandis que `querySelectorAll` renvoie une liste (*cf. ci-dessous*) :

```
var cells = document.querySelectorAll('table td');
for ( var n=0; n < cells.length; n++ ) {
    cells[n].textContent = n;
}
```

☞ Ces méthodes sont supportées par la plupart des navigateurs.
(A noter tout de même : `querySelector` / `querySelectorAll` = IE 8+, `getElementsByClassName` = IE9+).

C'est également l'objet `Document` qui implémente les méthodes permettant de créer dynamiquement des noeuds qui pourront ensuite être insérés dans l'arbre.

Méthodes de création de noeuds

<code>DocumentFragment</code>	<code>createDocumentFragment()</code> ;	// DOM1 # <i>fragment</i>
<code>Element</code>	<code>createElement(in DOMString tagName)</code> ;	// DOM1 # <i>élément</i>
<code>Text</code>	<code>createTextNode(in DOMString data)</code> ;	// DOM1 # <i>noeud texte</i>
<code>Attr</code>	<code>createAttribute(in DOMString name)</code> ;	// DOM1 # <i>attribut</i>

Exemples d'utilisation :

```
var frag = document.createDocumentFragment()
, img = document.createElement('IMG')
, txt = document.createTextNode('hello, world')
, attr = document.createAttribute('HREF')
;
```

A noter: HTML n'étant pas sensible à la casse, le nom des éléments et des attributs passés en argument aux méthodes `createElement()` et `createAttribute()` peuvent être aussi bien en majuscules qu'en minuscules.

☞ L'utilité d'un fragment sera explicitée par la suite.

2.2.4 L'interface `Node` – (le retour)

Les noeuds (*`DocumentFragment`, `Element`, `Text`*) une fois créés doivent être insérés dans l'arbre pour être vus. Les méthodes de modification dynamique de l'arbre sont implémentées par l'interface `Node` :

Méthodes de modification dynamique de l'arbre

<code>Node</code>	<code>appendChild(in Node newChild)</code> ;	// DOM1
<code>Node</code>	<code>insertBefore(in Node newChild, in Node refChild)</code> ;	// DOM1
<code>Node</code>	<code>replaceChild(in Node newChild, in Node oldChild)</code> ;	// DOM1
<code>Node</code>	<code>removeChild(in Node oldChild)</code> ;	// DOM1

Ces méthodes permettent d'insérer un noeud parmi la liste existante des enfants d'un noeud parent, de remplacer un enfant existant, ou d'en supprimer un.

De cette manière il est possible de construire dynamiquement le contenu d'un élément (*voire de l'ensemble du document*) :

```
for ( i = 0; i < 3; i++ ) {  
  li = document.createElement('li'); // création d'un élément li  
  txt = document.createTextNode(1+i); // création d'un noeud texte  
  li.appendChild(txt); // ajout du texte au contenu de l'item  
  ul.appendChild(li); // ajout de l'item à la liste  
}
```

[démonstration]

Noter que l'ajout d'un noeud texte peut se faire indifféremment sous deux formes, la première préconisée par DOM1, la seconde introduite par DOM3 :

```
li.appendChild( document.createTextNode('hello') ); // DOM1
```

```
li.textContent = 'hello'; // DOM3
```

Finalement, l'interface Node propose encore deux méthodes de confort :

```
boolean hasChildNodes(); // DOM1 vrai si children.length > 0  
Node cloneNode(in boolean deep); // DOM1 renvoie une copie du noeud
```

Tous les objets correspondant à des noeuds de l'arbre (*Document, Element, Text*) ainsi que DocumentFragment implémentent l'interface Node.

Toutefois, bien que l'objet Text implémente l'interface Node, l'invocation de l'une des méthodes de modification de l'arbre (*appendChild, insertBefore, replaceChild, removeChild*) depuis un noeud texte provoquera une erreur.

2.2.5 L'objet DocumentFragment

A chaque fois que l'on insère un noeud (*Element ou Text*) dans l'arborescence effectivement affichée par le navigateur, ce dernier doit recalculer l'affichage (*reflow*) pour immédiatement le mettre en page.

Lorsqu'on a de nombreux éléments à insérer à la suite les uns des autres, il peut être judicieux de ne pas répéter inutilement le calcul de l'affichage qui peut être coûteux en temps utilisateur et en charge CPU.

L'objet DocumentFragment est précisément conçu pour cela.

```
frag = document.createDocumentFragment() // création du fragment  
for ( n = 0; n < 3; n++ ) {  
  p = document.createElement('p');  
  p.textContent = 'Paragraphe : ' + n;  
  frag.appendChild(p); // insertion d'éléments dans le fragment  
}  
p2.parentNode.insertBefore(frag, p2); // insertion du fragment dans le document
```

[démonstration]

DocumentFragment implémente l'interface Node et a été spécifié par DOM1.


2.2.6 L'objet Text

L'objet Text implémente l'interface Node, et possède les attributs et méthodes suivants :

```
unsigned long length; // DOM1
Text splitText(in unsigned long offset); // DOM1
DOMString wholeText; // DOM3
```

`splitText` découpe le noeud texte en deux noeuds consécutifs, dont le premier fait la longueur spécifiée, `wholeText` correspond à la concaténation de la valeur de tous les noeuds textes frères du noeud courant :

```
text.nodeValue = 'hello, world';
console.log(text.parentNode.childNodes.length); // 1 ... enfant
console.log(text.length); // 12 .. caractères
text.splitText(6);
console.log(text.parentNode.childNodes.length); // 2 ... enfants
console.log(text.length); // 6 ... caractères
console.log(text.nodeValue); // hello,
console.log(text.wholeText); // hello, world
```

 **Rappel** : l'objet Text implémente l'interface Node, mais l'invocation de l'une des méthodes de modification de l'arbre (*`appendChild`*, *`insertBefore`*, *`replaceChild`*, *`removeChild`*) depuis un objet Text provoquera une erreur.

2.2.7 L'objet Element

L'objet Element implémente l'interface Node et possède les attributs suivants :

```
DOMString tagName; // DOM1
boolean hasAttribute(in DOMString name); // DOM2
DOMString getAttribute(in DOMString name); // DOM1
void setAttribute(in DOMString name, in DOMString value); // DOM1
void removeAttribute(in DOMString name); // DOM1
NodeList getElementsByTagName(in DOMString name); // DOM1
```

La méthode `getElementsByTagName` reprend la méthode éponyme de l'objet Document en limitant la liste des éléments renvoyés aux enfants de l'élément.

Les autres méthodes permettent de gérer la présence et la valeur des attributs de l'élément :

```
var a = document.createElement('a');
a.setAttribute('title', 'World-Wide Web Consortium');
a.setAttribute('href', 'http://www.w3.org');
a.textContent = 'W3C home page';
p2.parentNode.insertBefore(a, p2);
```

2.3 DOM HTML

DOM Core est une API générique qui s'applique aussi bien à des documents HTML qu'à des documents XML dans le cas le plus général. DOM HTML est l'API qui décrit les éléments d'un document HTML.

DOM HTML fait l'objet de plusieurs spécifications du W3C. [\[DOM HTML\]](#) [\[DOM2 HTML\]](#)

On pourra également se référer au document "HTML - Living Standard" publié par le Web Hypertext Application Technology Working Group. [\[WHATWG DOM\]](#)

DOM HTML propose essentiellement des attributs (*au sens objet*) de confort, qui donnent notamment accès à l'ensemble des attributs (*au sens HTML*) d'un élément.

```
var img = document.createElement('img');
img.setAttribute('src', 'logo.png'); // DOM Core
img.src = "logo.png"; // DOM HTML
```

N.B. Les spécifications présentent tout un ensemble d'interfaces implémentées par les objets correspondants (cf. *HTMLDocument*, *HTMLElement*, *HTMLHtmlElement*, *HTMLHeadElement*, *HTMLBodyElement*, *HTMLHeadingElement*, *HTMLParagraphElement*, *HTMLDivElement*, ...) qui ne seront pas abordées ici de manière exhaustive.

2.3.1 L'interface HTMLDocument

Lorsque le document traité est un document HTML, l'objet document implémente l'interface HTMLDocument, qui hérite de Document et expose les attributs additionnels décrits ci-après.

Métadonnées relatives à la ressource

```
DOMString    URL; // DOM2 HTML - URL de la page
Location?    location; // WHATWG HTML - emplacement de la page
DOMString    domain; // nom de domaine de la page
DOMString    referrer; // URL de la page dont on vient
DOMString    lastModified; // WHATWG HTML - date et heure de dernière modification
DOMString    cookie; // cookies associés à la page
```

[démonstration]

```
var date = new Date(document.lastModified)
p.textContent = 'Dernière modification : '+date.toLocaleDateString('fr-FR');
```

Dernière modification : 14/02/2016

🔗 location est un objet qui possède quelques attributs intéressants pour disséquer l'URL de la page (*href*, *pathname*, *protocol*, *host*, *hostname*, *port*, *search*, *hash*).

Attributs pour l'accès aux éléments du DOM

```
DOMString    title; // textContent de l'élément title
HTMLElement? body; // raccourci vers l'élément body
HTMLHeadElement? head; // WHATWG HTML - raccourci vers l'élément head
HTMLCollection images; // liste des éléments img du document
HTMLCollection links; // liste des éléments a et area possédant un attribut href
HTMLCollection forms; // liste des éléments form du document
HTMLCollection scripts; // WHATWG HTML - liste des éléments script du document

NodeList getElementsByName(DOMString elementName); // par valeur de l'attribut name
```

🔗 La plupart de ces attributs apparaissent dans les deux spécifications (W3C et WHATWG), sauf head et scripts. Certains attributs cités dans DOM2 HTML mais déclarés obsolètes par WHATWG HTML ne sont pas mentionnés ici.

Exemple d'utilisation :

```
var images = document.images,
    options = document.querySelectorAll('#mySelect option');
for ( n=0; n < document.images.length; n++ ) {
    options[n].value = images[n].src;
    options[n].textContent = (images[n].alt || '#img'+n);
}
```



2.3.2 L'interface HTMLElement

Tous les éléments HTML implémentent l'interface HTMLElement, qui hérite lui-même de l'interface Element défini par DOM Core.

```
DOMString    title;          // DOM2 HTML
DOMString    id;             // DOM2 HTML correspond à la valeur de l'attribut id
DOMString    className;     // DOM2 HTML correspond à la valeur de l'attribut class
DOMTokenList classList;     // WHATWG DOM renvoie la liste des classes
HTMLCollection getElementsByTagName(DOMString localName); // WHATWG DOM
HTMLCollection getElementsByClassName(DOMString classNames); // WHATWG DOM
```

Les méthodes `getElementsByName` reprennent les méthodes éponymes de l'interface Document en limitant la liste des éléments renvoyés aux enfants de l'élément.

La liste correspondant à l'attribut `classList` est un objet avec des méthodes pour manipuler individuellement les classes de l'élément (`contains()`, `add()`, `remove()`, `toggle()`).

Exemple :

```
button.addEventListener('click', function(){ para.classList.toggle('blue') });
```



L'interface HTMLElement implémente également l'attribut `style` permettant de modifier le style CSS d'un élément : [DOM2 Style]

```
CSSStyleDeclaration    style;          // DOM2 Style
```

Cet objet permet de lire et de modifier la valeur des propriétés CSS spécifiées via l'attribut HTML `style` de l'élément considéré

Les propriétés de l'objet `style` sont nommées d'après les propriétés CSS en remplaçant les éventuels tirets par une majuscule. Ainsi la propriété CSS `background-color` devient par exemple `backgroundColor`

```
document.getElementById('tagada').style.backgroundColor = 'pink';
```

⚠ Attention, il n'est pas possible d'accéder de cette manière à l'ensemble des propriétés CSS d'un élément. Il faut pour cela recourir à la méthode `getComputedStyle()` [CSSOM]

```
s = getComputedStyle(document.getElementById('para'));
console.log(s.getPropertyValue('background-color')); // rgba(0,0,0,0)
console.log(s.getPropertyValue('color'));           // rgb(102,102,102)
console.log(s.getPropertyValue('width'));            // 748px
```

L'interface `HTMLElement` implémente également les gestionnaires d'événements via des attributs HTML "à la DOM Level 1" accessibles directement sous forme de d'attributs (*au sens objet*).

Exemples d'usage :

```
div.onmousedown = function() { this.className = "down"; };
div.onmouseup = function() { this.className = "up"; };
div.onmouseout = function() { this.className = "up"; };
pre.onmousemove = function(e) {
    display.textContent = e.clientX + ', ' + e.clientY;
}
input.onkeypress = function(e) { span.textContent = e.charCode; };
```

Hit me !

217, 359

✎ En HTML 5, la grande majorité des gestionnaires supportés par `HTMLElement` sont répertoriés via l'interface `GlobalEventHandlers` et sont également supportés par les objets `Document` et `Window`, sauf les gestionnaires liés à l'édition (*oncopy*, *oncut*, *onpaste*) uniquement supportés par `HTMLElement` et `Document`. [[DocumentAndElementEventHandlers](#)]

2.4 DOM Events

2.4.1 Événements HTML 4.0 vs. DOM

En HTML 4.0 les gestionnaires d'événements sont spécifiés sous la forme d'attributs. Il ne peut y avoir qu'un seul gestionnaire par événement :

```
<button onclick="alert('hello');"> Bonjour ! </button>
```

```
button.onclick = function() { alert('hello'); };
```

Le modèle de gestion des événements du DOM permet l'enregistrement de plusieurs gestionnaires pour le même événement d'un élément donné. Il n'est donc plus possible de spécifier ces événements via un attribut :

```
function hello() { alert('hello'); }
element.addEventListener('click', hello );
```

La compatibilité avec HTML 4.0 est assurée en considérant que pour un événement donné, le gestionnaire spécifié via l'attribut prend sa place parmi les autres dans la liste des gestionnaires enregistrés.

Si la valeur de l'attribut est modifiée en cours de route, le gestionnaire précédent est supprimé de la liste.

L'API [[DOM Level 2 Events](#)] ajoute à tous les nœuds (*Node*) des méthodes qui permettent de gérer une liste de gestionnaires d'événements.

`addEventListener(type, listener, useCapture)`

```
button.addEventListener('mousedown', function() {
    this.style.borderStyle = 'inset';
});
```

enregistre une fonction comme gestionnaire pour tout événement du type mentionné déclenché sur le nœud considéré.

node.removeEventListener(type, listener, useCapture)

```
button.removeEventListener('click');
```

supprime le ou les gestionnaires enregistrés pour un événement donné.

- ✎ Pour supprimer un gestionnaire donné il est nécessaire de mentionner la fonction de traitement. Il est par conséquent impossible de supprimer un gestionnaire déclaré avec une fonction anonyme.

```
<button id="test_button">test</button>
<script>
  test_button.addEventListener('click', buttonAlert);
  function buttonAlert() {
    alert('button clicked');
    this.removeEventListener('click', buttonAlert);
  }
</script>
```

node.dispatchEvent(event)

```
var event = new Event('demo');
var div = document.getElementById('testDispatchEvent');
div.addEventListener('demo', function(e) {
  console.log('hello from demo listener');
});
div.dispatchEvent(event);
```

permet de déclencher artificiellement un événement de type arbitraire ou existant sur un élément cible, comme s'il était effectivement intervenu.

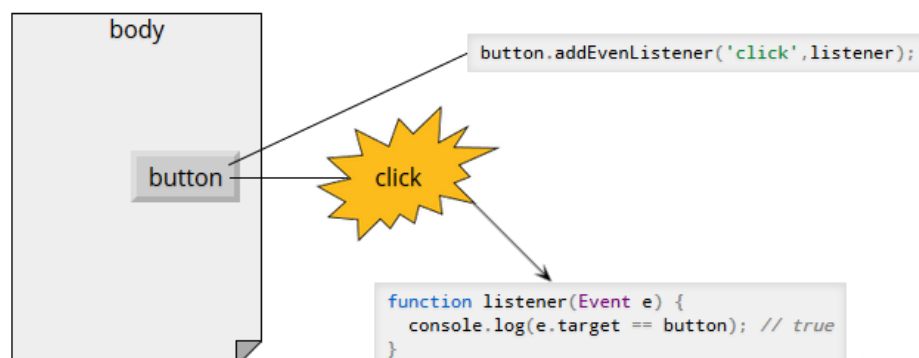
- ✎ Les fonctions gestionnaires d'événement sont appelées avec en argument un objet implémentant l'interface Event qui décrit l'événement déclencheur, automatiquement créé par le DOM.

Lorsqu'on veut simuler un événement, il faut créer manuellement un objet Event que l'on passe en argument à *dispatchEvent*. Les attributs de l'interface Event sont décrits dans la suite.

2.4.2 Déroulement des événements

Suivant le modèle du DOM, tout événement est associé à un élément cible, sur lequel est déclenché l'événement. Cet élément est mentionné par l'attribut target de l'objet Event.

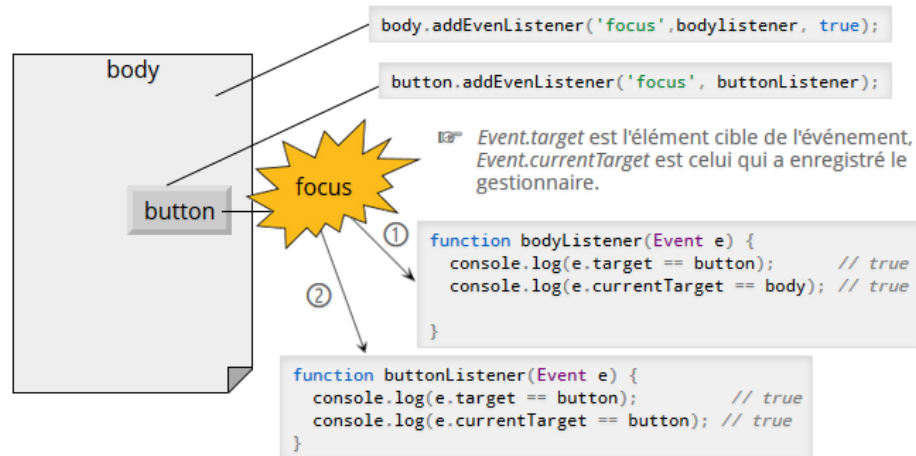
Lorsque l'événement intervient, tous les gestionnaires enregistrés pour cette cible et ce type d'événement sont appelés.



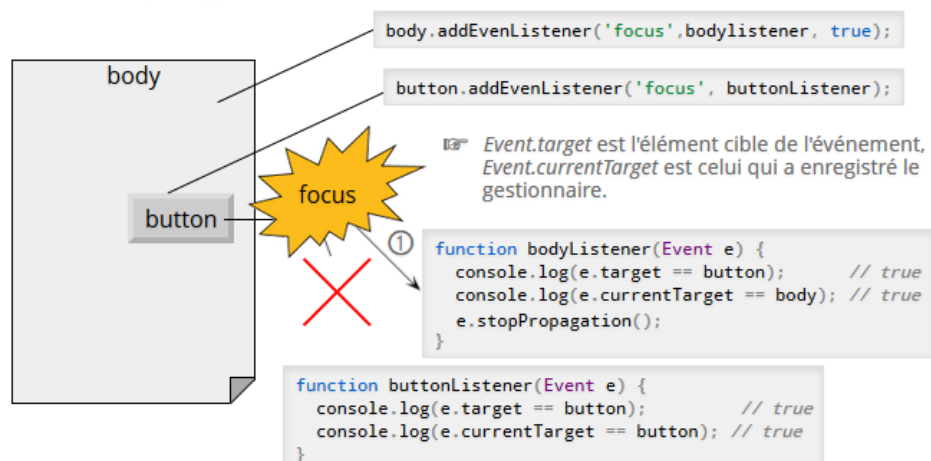
Capture

Si un **parent de la cible** enregistre un gestionnaire en précisant l'argument *capture=true*, alors ce gestionnaire est appelé.

Si la cible possède également un gestionnaire pour cet événement, alors ce gestionnaire est appelé **après** celui du parent.



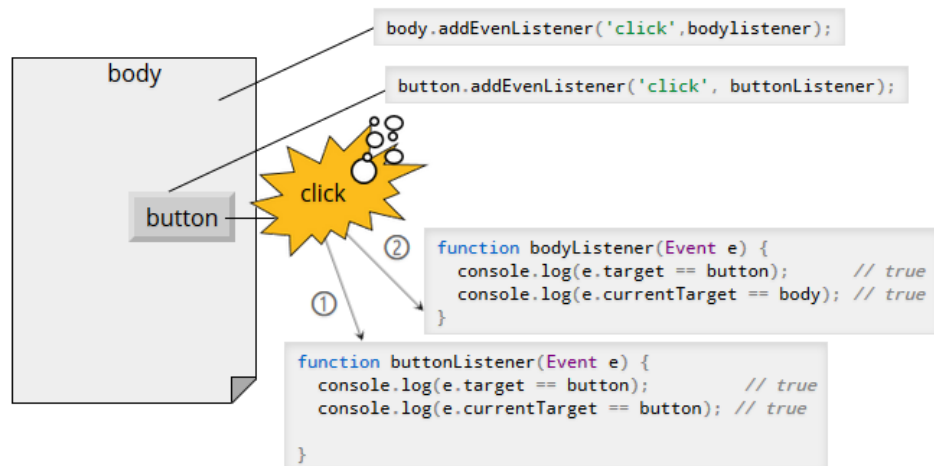
Si le gestionnaire du parent appelle *Event.stopPropagation()*, alors le gestionnaire de la cible n'est **pas** appelé.



Bouillonnement

Certains événements sont dits "bouillonnants". Sauf en cas de capture, un événement de cette catégorie déclenche d'abord les gestionnaires de la cible (*target*), puis remonte l'arbre vers le parent, puis le parent du parent, jusqu'à arriver au nœud document.

Le gestionnaire d'un parent ayant capturé l'événement est toujours appelé avant la cible, mais n'est dans ce cas pas appelé durant la phase de bouillonnement.



Si le gestionnaire de la cible appelle *Event.stopPropagation()*, alors le gestionnaire du parent n'est pas appelé.

Annulation du comportement par défaut

Suivant le contexte, certains événements peuvent déclencher un comportement par défaut chez le navigateur (*suivi de lien, soumission de formulaire, ...*). Tous ces événements (*click, mousedown, mouseup, mouseover, mouseout, submit, activate*), sont annulables (*cancelable*).

Avant de déclencher l'action par défaut, le navigateur appelle les gestionnaires enregistrés pour cet événement qui ont alors l'opportunité d'empêcher l'occurrence de l'action par défaut, en appelant la méthode *preventDefault()* de l'objet Event :

```
if ( evt.type == 'click' && evt.target.nodeName == 'A' ) {
    evt.preventDefault();
}
```

Arrêt immédiat de toute propagation

Contrairement à ce qu'indique DOM2, il semble (*specs ?*) que les pratiques actuelles fassent que lorsque plusieurs gestionnaires sont attachés au même élément pour un même événement, ils sont appelés **dans l'ordre dans lequel ils ont été enregistrés**. [\[event order\]](#)

La méthode *stopPropagation()* déjà évoquée empêche l'événement d'atteindre d'autres éléments que l'élément courant, mais les gestionnaires suivants enregistrés sur le même élément sont tout de même appelés.

La méthode *stopImmediatePropagation()* empêche tout traitement ultérieur de l'événement, même par des gestionnaires enregistrés sur la même cible que celui qui effectue cet appel.

2.4.3 Divers types d'événements

L'interface Event

Tous les événements du DOM implémentent l'interface **Event**. Autrement dit, ils partagent tous les attributs suivants :

```
DOMString    type;           // nom de l'événement
EventTarget  target;        // élt à l'origine de l'événement
EventTarget  currentTarget; // élt porteur du gestionnaire
unsigned short eventPhase;   // phase actuelle de l'événement
boolean      bubbles;        // évt bouillonnant ?
boolean      cancelable;     // évt annulable ?
DOMTimeStamp timeStamp;     // instant de l'évt (ms ?)
```

... ainsi que les méthodes *stopPropagation()*, *stopImmediatePropagation()* et *preventDefault()*.

L'interface UIEvent


Le document de travail **UI Events** spécifie l'interface **UIEvent** et d'autres qui en héritent et sont implémentés par l'ensemble des événements décrits par la suite.

Hérite de **Event**.

```
Window view; // fenêtre dans laquelle a eu lieu l'événement
long detail; // valeurs diverses suivant le type de l'événement
```

Exemple :

```
div.addEventListener('click', function(e) {
  console.log(e.view == window); // true
  console.log(e.detail);          // 1 = click, 2 = double click, ...
});
```

 L'attribut **view** peut être utile lorsque l'on utilise des **Iframes**...

Certains événements utilisent directement cette interface :

event	target	occurrence
load	W,D,E	la cible est complètement chargée (DOM exploitable)
unload	W,D,E	la cible est en passe d'être supprimée
abort	W,E	le chargement de la cible a été interrompu par l'utilisateur
error	W,E	la cible ne correspond pas au format attendu
select	E	l'utilisateur a surligné du texte

 Cibles : W = Window, D = Document, E = Element.

L'interface FocusEvent

Hérite de **UIEvent** et de **Event**.

```
EventTarget relatedTarget; // élément lié à l'événement (cf. infra)
```

Les événements qui implémentent cette interface sont liés à la gestion du focus, le mécanisme permettant de naviguer au sein d'une interface web entre champs de formulaires et hyperliens à l'aide des touches de tabulation.

☞ Un champ de formulaire ayant le focus reçoit les caractères tapés au clavier, un hyperlien ayant le focus peut être déclenché avec la touche *retour*.

event	target	occurrence
blur	W,E	la cible vient de perdre le focus
focus	W,E	la cible vient de gagner le focus
focusin	W,E	la cible est en passe de gagner le focus
focusout	W,E	la cible est en passe de perdre le focus

L'attribut *relatedTarget* indique l'élément perdant le focus (*pour focus ou focusin*) et l'élément gagnant le focus (*pour blur et focusout*).

Lors du passage du focus d'un élément à l'autre, l'ordre des événements est *focusout*, *focusin*, *blur*, *focus*.

N.B. *focusin* et *focusout* sont bouillonnants, *blur* et *focus* non.

L'interface MouseEvent

Hérite de *UIEvent* et de *Event*.

```

long screenX;           // abscisse dans le référentiel de l'écran
long screenY;           // ordonnée dans le référentiel de l'écran
long clientX;           // abscisse dans le référentiel de la cible
long clientY;           // ordonnée dans le référentiel de la cible
boolean ctrlKey;        // vrai si la touche Ctrl est activée
boolean shiftKey;       // vrai si la touche Shift est activée
boolean altKey;         // vrai si la touche Alt est activée
boolean metaKey;        // vrai si la touche Meta est activée
short button;           // numéro du bouton activé (0, 1 ou 2)
unsigned short buttons; // combinaison de boutons (bitmask)
EventTarget relatedTarget; // cible éventuelle liée à l'événement

```

Cette interface est utilisée par tous les événements liés à une action de l'utilisateur via le dispositif de pointage (*souris, trackball, trackpad, trackpoint...*) :

event	target	occurrence
click	E	le bouton principal a été enfoncé puis relâché ... au-dessus la cible
dblclick	E	... 2 fois de suite
mousedown	E	un bouton a été enfoncé avec le curseur au-dessus de la cible
mouseenter	E	le curseur commence à survoler la cible ou l'un de ses enfants
mouseleave	E	le curseur ne survole plus la cible ni aucun de ses enfants
mousemove	E	le curseur a été déplacé au-dessus de la cible
mouseout	E	le curseur ne survole plus la cible
mouseover	E	le curseur commence à survoler la cible
mouseup	E	un bouton a été relâché avec le curseur au-dessus de la cible

Les événements liés à l'action d'un bouton, ainsi que *mouseover*, *mouseout*, et *mousemove* sont bouillonnants, contrairement à *mouseenter* et *mouseleave*.

Pour *click* et *dblclick* l'attribut *detail* indique le nombre de clics en cours, pour *mousedown* et *mouseup* le nombre de clics incrémenté de 1.

Pour *mouseenter* et *mouseover* *relatedTarget* indique l'élément qui vient d'être quitté, pour *mouseleave* et *mouseout* celui qui est actuellement survolé.

Le changement de référentiel depuis celui utilisé par ces événements vers un système plus intuitif nécessite une petite gymnastique : `[position curseur]` `[getBoundingClientRect]`

```
<div id="testDiv">
  <div id="testCursor"></div>
</div>
```

```
#testDiv { position: relative; }
#testCursor { position: absolute; width: 0;
  height: 0; border: 2px solid red; }
```

```
testDiv.addEventListener('mousemove', function(e) {
  if ( e.target !== e.currentTarget ) return;
  var rect = e.target.getBoundingClientRect();
  testCursor.style.left = (e.clientX - rect.left) + 'px';
  testCursor.style.top = (e.clientY - rect.top) + 'px';
});
```

Lors de l'action sur un bouton, l'ordre des événements est *mousedown*, *mouseup*, *click* éventuellement suivi par *mousedown*, *mouseup*, *click*, *dblclick*. Lors du survol d'un élément par le curseur : *mouseover*, *mouseenter*, *mousemove*... *mouseout*, *mouseleave*.

☞ Les spécifications précisent également l'ordre exact des événements dans des cas de figure plus complexes, notamment lorsque le curseur survole des éléments imbriqués. [\[ordre des événements\]](#)

L'interface WheelEvent

Cette interface est destinée aux événements liés à des dispositifs rotatifs comme une molette ou un trackball.

Hérite de MouseEvent.

```
double deltaX;           // mesure du déplacement suivant l'axe X
double deltaY;           // mesure du déplacement suivant l'axe Y
double deltaZ;           // mesure du déplacement suivant l'axe Z
unsigned long deltaMode; // 0 : pixels, 1 : lignes, 2 : pages
```

Un seul événement est spécifié comme utilisant cette interface :

event	target	occurrence
wheel	E	Le dispositif a détecté une rotation

Les mesures de déplacement sont exprimées en pixels, en lignes ou en pages. L'attribut *deltaMode* indique quelle est l'unité utilisée.

Lorsque le dispositif rotatif est associé à un pointeur (cf. *molette de souris par exemple*), les attributs définis par MouseEvent sont renseignés comme il se doit.

L'interface InputEvent

Hérite de UIEvent.

```
DOMString data;           // caractère saisi
boolean isComposing;       // utilisé pour l'édition en ligne
```

Les spécifications prévoient deux événements conformes à cette interface :

event	target	occurrence
beforeinput	E (contenteditable)	Une modification est en passe d'être effectuée
input	E (contenteditable)	Une modification vient d'être effectuée

Ces événements sont déclenchés lors de la modification du contenu d'un élément HTML `<input>`, `<select>` ou `<textarea>`, ou lors de la modification de contenu d'un éditeur **contenteditable**.

☞ Chrome est à l'heure actuelle (03-2019) le seul navigateur à implémenter l'événement *beforeinput* qui n'est même pas mentionné sur **Can I Use**.

Par ailleurs, Firefox ne dispose pas de l'attribut *data*, Edge et IE ne déclenchent pas l'événement *input* sur `<select>` et Firefox est le seul à déclencher sur les éléments `<input>` du type `checkbox` ou `radio`. [\[notes de compatibilité\]](#)

Exemple de code pour tester l'implémentation de ces événements :

```
function handler(e) {
  pre.innerHTML += this+(e.data?` "${e.data}"`:' null')+`<br>`;
}
['beforeinput', 'input'].forEach( e =>
  inputDemo.addEventListener(e, handler.bind(e)));
```

L'interface KeyboardEvent

Cette interface est destinée aux événements correspondant à l'action d'une touche sur le clavier.

Hérite de UIEvent.

```
DOMString key;           // valeur (caractère ou nom) de la touche
DOMString code;          // nom de l'emplacement physique de la touche
unsigned long location;   // 0: standard, 1: gauche, 2: droite, 3: numpad
boolean ctrlKey;          // vrai si touche Ctrl active
boolean shiftKey;         // vrai si touche Shift active
boolean altKey;           // vrai si touche Alt active
boolean metaKey;          // vrai si touche Meta active
boolean repeat;           // indicateur de répétition
boolean isComposing;      // utilisé pour l'édition en ligne
```

Les événements utilisant cette interface sont :

event	target	occurrence
keydown	E	une touche a été enfoncée
keyup	E	une touche a été relâchée

☞ L'événement *keypress* (non décrit ici) avec les attributs *keyCode* et *charCode* a été déclaré obsolète et ne devrait pas être utilisé

Les chaînes de caractères décrivant le nom des touches, ainsi que leurs emplacements font l'objet d'une spécification dédiée. [\[codes clavier\]](#)

☞ IE et Edge n'implément pas l'attribut *code* et ont certains problèmes avec l'attribut *key*. [\[Can I use keyboard\]](#)

Démo live :

key	code	location	ctrlKey	shiftKey	altKey	repeat
a	KeyQ	0	false	false	false	false
Shift	ShiftLeft	1	false	true	false	false
A	KeyQ	0	false	true	false	false
Control	ControlLeft	1	true	false	false	false
a	KeyQ	0	true	false	false	false
Control	ControlLeft	1	true	false	false	false
Alt	AltRight	2	true	false	true	false
a	KeyQ	0	true	false	true	false
End	Numpad1	3	false	false	false	false
NumLock	NumLock	0	false	false	false	false
1	Numpad1	3	false	false	false	false

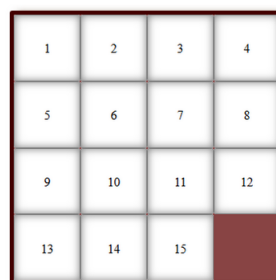
2.4.4 Exercices

- Créer une page interactive avec la liste des couleurs nommées. Pour chacune des couleurs la page devra :
 - afficher le nom de la couleur,
 - afficher une zone colorée avec la couleur en question,
 - afficher un popup lors du survol de la case colorée, avec le code HTML de la couleur.

La page devra par ailleurs proposer un mécanisme permettant de choisir entre le format `#RRGGBB` ou `rgb()` pour l'affichage de la couleur lors du survol par le curseur.

La liste des couleurs est disponible [en ligne](#).

- Créer une page faisant revivre le jeu de télécran. [\[télécran\]](#)
- Créer une page permettant de jouer au taquin. [\[taquin\]](#)



2.5 Ajax

2.5.1 Qu'est-ce qu'Ajax ?

Ajax **n'est pas** une technologie, mais un acronyme pour *Asynchronous Javascript And XML* conçu en 2005 pour désigner un certain type d'usage de technologies existantes comme HTML, CSS, Javascript, le DOM, et surtout l'objet XMLHttpRequest.

Le modèle Ajax combine ces technologies pour mettre à jour l'interface utilisateur à l'aide d'informations échangées avec le serveur de manière transparente pour l'utilisateur.

Dans l'exemple ci-dessous, le paragraphe de droite est dynamiquement mis à jour à l'aide d'Ajax, lorsque le curseur survole l'un des termes de la colonne de gauche (*nécessite une connexion internet*).

[démon live]

- HTML
- CSS
- Javascript
- DOM
- XMLHttpRequest

XMLHttpRequest est un objet ActiveX ou JavaScript qui permet d'obtenir des données au format XML, JSON, mais aussi HTML, ou encore texte simple à l'aide de requêtes HTTP.
On explique le succès récent de l'objet et la très grande utilisation qui en est faite actuellement (parfois au détriment de l'accessibilité d'un site) par la simple création du nom AJAX.
Source: Wikipedia

À noter : bien que le "X" de Ajax signifie XML, il est devenu plus courant d'utiliser le format JSON pour les informations échangées entre le navigateur et le serveur. [JSON] [RFC4627]

2.5.2 L'objet XMLHttpRequest

XMLHttpRequest est un objet Javascript qui permet d'effectuer très simplement des requêtes vers le serveur : [XMLHttpRequest] [Using XMLHttpRequest]

```
var request = new XMLHttpRequest();

request.onload = function() {
    dropzone.innerHTML = this.responseText; // par exemple
};

request.open("GET", "exemple1_HTML.html", true);
request.send();
```

En fonction de la ressource demandée au serveur, la réponse peut être analysée comme un texte (via `request.responseText`) ou comme un document XML (via `request.responseXML`).

D'autres informations peuvent être exploitées, comme le statut et les entêtes de la réponse (cf. `request.status` et `request.getResponseHeader()`).

Bon à savoir : Pour suivre plus précisément le déroulement et le résultat de la requête, XMLHttpRequest peut déclencher d'autres événements comme `error`, `progress` ou `abort`.

2.5.3 Historique

L'objet XMLHttpRequest a été développé en 1998 par Microsoft comme objet ActiveX pour Internet Explorer 5.0, puis repris successivement par Mozilla en 2002, Safari en 2004, et Opera en 2005 sous la forme d'un objet Javascript.

C'est à cette époque que l'on voit apparaître pour la première fois le terme *AJAX*. [article fondateur]

Cet objet a fait l'objet d'un draft auprès du W3C dès 2006. Un second draft nommé *XMLHttpRequest Level 2* a proposé une meilleure gestion des événements liés à la requête en cours (*ProgressEvent*) et des requêtes Cross-Origin. Ces deux propositions ont été fusionnées en 2011.

Actuellement, et depuis 2012, les spécifications de référence concernant XMLHttpRequest sont maintenues par le WHATWG. [XMLHttpRequest living standard]

2.5.4 Un potentiel problème de sécurité

Imaginons un utilisateur *lambda* ayant une fenêtre de navigateur ouverte sur son gestionnaire de messagerie (disons *gmail*).

L'utilisateur ouvre ensuite une seconde fenêtre vers le site *pirate.net*.

La page de garde de ce site contient un script qui effectue une requête sur gmail. Comme l'utilisateur est déjà identifié, le navigateur réutilise ses identifiants pour accéder à gmail qui renvoie une page HTML contenant la liste des messages de l'utilisateur.

Le script analyse ces données, les renvoie à *pirate.net*, et bien sûr n'affiche rien de ce qu'il vient de faire. Les pirates connaissent maintenant une partie des contacts de l'utilisateur, et savent de quoi sont faits ses messages.

🔧 *pirate.net* aurait pu tout aussi bien faire une requête sur gmail qui :

- efface la liste des contacts,
- envoie un mail à la place de l'utilisateur,
- change son mot de passe, ou efface ses messages...

Cet exemple s'applique évidemment de la même façon à tous les réseaux sociaux, les sites de banque, etc...

2.5.5 CORS - Cross-Origin Resource Sharing

Pour éviter ce genre d'attaques, XMLHttpRequest possède plusieurs limitations. En particulier :

- il est impossible d'accéder aux URLs en file:,
- un script émis par un serveur ne peut se connecter qu'à ce même serveur (*Same Origin Policy*), ou à un serveur autorisant explicitement les accès par d'autres scripts que les siens (*standard CORS*). [\[Wikipédia\]](#) [\[Recommandation\]](#)

Lorsqu'un script provenant d'un serveur *srv1* émet une requête AJAX vers un serveur *srv2*, le navigateur ajoute une en-tête à la requête :

```
Origin: http://srv1
```

A réception de cette requête, si le serveur *srv2* fait confiance à *srv1*, il inclut dans sa réponse une en-tête :

```
Access-Control-Allow-Origin: http://srv1
```

... et le navigateur autorise alors le script à accéder à la réponse.

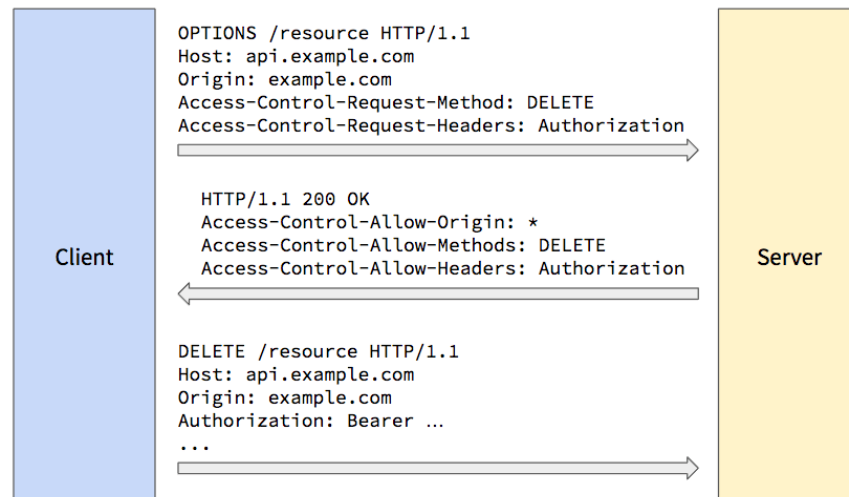
En l'absence de cette entête le navigateur refuse de transmettre la réponse au script. Du point de vue du script, c'est comme si la requête avait échoué.

Cette méthode protège un site non *CORS-aware* par défaut : si le site n'inclut pas l'entête *Access-Control-Allow-Origin* dans sa réponse, il ne sera pas possible de l'exploiter via AJAX.

🔧 Cette limitation est toutefois facile à contourner via un proxy qui ajoute systématiquement cette entête à toutes les réponses qu'il obtient en tant qu'intermédiaire. [\[crossorigin.me\]](#)

Dans la pratique, CORS est bien plus complexe que cela. [\[MDN CORS\]](#) [\[organigramme\]](#)

En particulier, les requêtes HTTP autres que GET, HEAD, POST avec un Content-Type classique donnent lieu à une pré-requête (*preflight request*) destinée à s'assurer que le serveur est compatible CORS :



N.B. Illustration dérobée sur le site <https://drstearns.github.io/tutorials/cors/>

2.5.6 Exemples de requêtes AJAX

Exploitation du statut et des entêtes de la réponse

```

var request = new XMLHttpRequest();

request.onload = function() {
  window['ajax-dropzone'].appendChild(
    document.createTextNode([
      this.status + ' ' + this.statusText,
      this.getAllResponseHeaders(),
      this.responseText
    ].join("\n"))
  );
};

request.open("GET", "http://httpbin.org/robots.txt", true);
request.send();
  
```

```

200 OK
content-type: text/plain

User-agent: *
Disallow: /deny
  
```

Suivi de la progression de la réponse

```
var r = new XMLHttpRequest();

r.addEventListener("load", () => display('Transfer completed'));
r.addEventListener("error", () => display('Error during transfer'));
r.addEventListener("abort", () => display('Canceled by user'));
r.addEventListener("progress", e => {
    if (e.lengthComputable) {
        pc = e.loaded / e.total * 100;
        display(pc);
    }
});
// possibilité de piloter une barre d'avancement...
function display(s) {
    dropzone.appendChild(document.createTextNode(s+"\n"));
};

r.open("GET", "http://httpbin.org/bytes/100000", true);
r.send();
```

```
14.366000000000001
43.566
49.406
100
Transfer completed
```

Requête POST

```
form.onsubmit = function(e) {
    // empêche le remplacement de la page par la réponse du service
    e.preventDefault();

    let request = new XMLHttpRequest();
    request.onload = function() { ... }; // on traite la réponse

    query_string = [form[1],form[2]].map(f =>
        encodeURIComponent(f.name)+'='+encodeURIComponent(f.value)
    ).join('&');

    request.open("POST", "http://httpbin.org/post", true);
    request.setRequestHeader(
        'Content-Type', 'application/x-www-form-urlencoded');
    request.send(query_string);
}
```

Formulaire Prénom: Nom:

```
200 OK
content-type: application/json

{"nom":"Deubaze","prenom":"Raymond"}
```


2.5.7 Exercice

Il a déjà été mentionné que pour des raisons de sécurité les requêtes AJAX ne fonctionnent pas vers des URLs en file:, c'est à dire pour charger depuis une page des fichiers situés sur le disque dur.

Comme CORS impose par ailleurs à une requête AJAX d'indiquer l'origine du script effectuant la requête (cf. *entête HTTP Origin*;) il est également impossible de faire fonctionner AJAX **depuis** une page en file: c'est-à-dire chargée directement par le navigateur depuis le disque de sa machine.

Toutefois, Firefox (c'est le seul) accepte d'effectuer des requêtes AJAX depuis des documents locaux, c'est-à-dire en file:, vers d'autres documents locaux, ce qui permet de développer et de tester de petites applications sans devoir recourir à un serveur. Cet exercice nécessite donc d'utiliser Firefox, ou d'avoir recours à un serveur.

- Développer une application d'un "livre dont vous êtes le héros" à partir des informations fournies **en ligne** sous forme d'un ensemble de fichiers au format json décrivant chacune des pièces à visiter. (Courtesy P.A. Champin)

Exemple de fichier json fourni :

```
{
  "links": [
    {
      "link": "#2",
      "txt": "Rendez vous au 2"
    }
  ],
  "txt": "1. Alors que vous pénétrez dans les ruines du donjon
apparemment abandonnées, la lourde grille se referme derrière
vous comme par magie. Il va falloir trouver une autre sortie !"
}
```

Fonctionnement demandé :

Lorsque l'utilisateur visite une pièce, l'application doit afficher le texte décrivant la pièce et proposer les liens vers les pièces voisines. L'activation d'un lien doit provoquer le chargement du fichier json correspondant, qui sera traité en javascript pour décrire la pièce suivante. Exemple de rendu :

Le livre dont vous êtes le héros

1. Alors que vous pénétrez dans les ruines du donjon, apparemment abandonnées, la lourde grille se referme derrière vous, comme par magie. Il va vous falloir trouver une autre sortie !

Rendez vous au 2

Une attention particulière sera portée à l'expérience utilisateur, et plus précisément au bon fonctionnement du bouton "back" du navigateur pour revenir en arrière dans le parcours de l'histoire.

Pour cela :

- Il est très fortement conseillé d'utiliser des identifiants de fragment (*parfois appelés "liens internes"*) comme par exemple `livre.html#1` pour naviguer d'une pièce à l'autre, car ce type de liens ne provoque pas de rechargement de la page par le navigateur.
- Les actions de l'utilisateur pourront être interceptées en s'abonnant à l'événement `hashchange` de `window`, et en utilisant l'API `window.location` pour déterminer la requête AJAX à effectuer afin de récupérer les informations à afficher.

✎ La solution qui consisterait à intercepter les clics sur les liens, pour savoir quand charger et afficher les éléments du livre peut sembler plus naturelle pour le programmeur, mais crée une expérience utilisateur bien plus pauvre que la solution préconisée. En effet, avec cette méthode :

- la navigation à l'intérieur du livre n'est pas stockée dans l'historique du navigateur,
- donc le bouton "retour" ne fonctionne pas,
- un rechargement de la page redémarre au premier élément,
- il n'est pas possible de mettre un signet sur l'endroit où l'on se trouve...

[exemple de solution]

2.6 Autres APIs

2.6.1 APIs client

Le développeur Web est très vite confronté à une multitude de ressources disponibles sous forme d'APIs (*Application Programming Interfaces*) dont certaines ont déjà été abordées : DOM Core, DOM HTML, DOM Events, XMLHttpRequest.

En voici d'autres, proposées dans le contexte d'un navigateur Web (*couverture évidemment largement non exhaustive*). [APIs]

Audio et vidéo

HTML5 propose deux balises dédiées au contenu multimédia : `<audio>` et `<video>` qui disposent d'une API dédiée pour le contrôle du lecteur. [media APIs]

Dans son utilisation la plus simple, la balise `<audio>` permet de jouer des plages sonores depuis un document HTML, de manière automatique ou via une interface de type *player*.

```
<audio src="boo_boopy_doo.wav" controls="true"></audio><span>Boo Boopy Doo...</span>
```



L'aspect visuel de l'élément `audio` dépend beaucoup du navigateur, et il est peu modifiable via CSS.

Il est par contre possible de le rendre invisible (*il suffit d'omettre l'attribut controls*) et de le contrôler via Javascript.



© 2018 GetDrawings.com, [CC NC-BY 4.0] <http://getdrawings.com/old-radio-drawing>

```
<audio id="FatsDomino" src="Blueberry_Hill.mp3"></audio>

<script>
old_radio.addEventListener('mouseover', function(){ FatsDomino.play(); });
old_radio.addEventListener('mouseout', function(){ FatsDomino.pause(); });
</script>
```

Comme le montre cet exemple, l'API exposée par l'élément `audio` permet de contrôler le démarrage du player, et la mise en pause.

Certains attributs permettent de contrôler plus finement le fonctionnement espéré. Ainsi l'attribut `paused` indique l'état du player :

```
old_radio.addEventListener('click', startStopAudio);
function startStopAudio() {
  let media = window.FatsDomino;
  if ( media.paused ) media.play(); // cf. attribut "paused"
  else media.pause();
}
```


L'arrêt complet passe par une pause suivie d'un rembobinage forcé :

```
function stopAudio() {
  let media = window.FatsDomino;
  media.pause();
  media.currentTime = 0; // cf. attribut "currentTime"
}
```

Durée totale de l'extrait de Fats Domino : 140.9 s

```
mediaDuration.textContent = parseInt(media.duration*10)/10;
```

Utilisé conjointement avec l'attribut `duration`, `currentTime` (cf. ci-dessus) permet d'envisager la création d'actionneurs du type "avance rapide" ou "retour rapide"...

 L'interface `HTMLMediaElement` expose d'autres attributs et méthodes dont notamment `autoplay`, `loop` ou `volume` dont le nom est suffisamment explicite et la découverte du fonctionnement laissée à la sagacité du lecteur. [\[HTMLMediaElement\]](#)

La balise `<video>` est comparable dans son fonctionnement et dans son utilisation à celle de l'élément `audio`, et partage la même API.

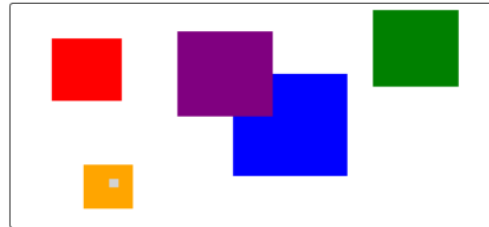


La fenêtre d'affichage de la vidéo est sujette à personnalisation via CSS, mais la présentation du tableau de contrôle est figée, et dépend du navigateur. [\[exemple en ligne\]](#)

Toutefois, comme il est possible de contrôler le *player* via Javascript, cela permet de concevoir n'importe quelle interface maison via des éléments HTML standards.

Canvas

La balise `<canvas>` de HTML5 crée une zone dans la page, dans laquelle il est possible de tracer des images à l'aide de Javascript.



Il existe pour cela au moins deux APIs différentes (*d'autres peuvent apparaître dans le futur*) accessibles via un objet de contexte, nommées `2d` et `webgl` :

```
var rnd = (m) => parseInt(Math.random()*m)
, canvas = document.querySelector("#canvas1")
, ctx = canvas.getContext('2d') // utilisation du contexte 2d
, colors = ['blue', 'red', 'orange', 'purple', 'green', 'lightgrey']
;
for ( i = 0; i < 6; i++ ) {
  var s = rnd(canvas.height/2);
  ctx.fillStyle = colors[i];
  ctx.fillRect(rnd(canvas.width-s), rnd(canvas.height-s), s, s);
}
```

Le contexte `2d` est le plus simple à utiliser. Comme son nom l'indique il permet de générer des graphiques 2D. [\[canvas API\]](#)

Le contexte `webgl` permet de gérer des scènes 2D ou 3D. Pour ceci, le navigateur fait directement appel au(x) GPU(s) disponibles sur la machine du client. [\[WebGL\]](#)

Il n'est pas dans le propos de ce cours de détailler ces APIs. Il existe pour cela de nombreux tutoriaux en ligne, auxquels le lecteur intéressé pourra se reporter... [\[tutorials\]](#) [\[tutorials\]](#)

Exercice

A l'aide de l'API canvas, tracer un assortiment de drapeaux nationaux de votre choix en n'oubliant pas de mentionner au voisinage de chaque drapeau le nom du pays correspondant.



🔧 La création de fonctions génériques pour le tracé de drapeaux à n rayures horizontales ou verticales ne serait pas un luxe...

2.6.2 Services

Les APIs vues jusqu'à présent sont toutes disponibles directement depuis un navigateur. Ce n'est pas toujours le cas. Il existe de nombreuses ressources en ligne mettant à disposition des APIs pour la manipulation de services comme : Google Maps, Twitter, Facebook, Paypal...

Les quelques exercices qui suivent vous permettront symboliquement d'en égratigner la surface.

Polices open source

- Développer une page permettant d'afficher un texte arbitraire (*type "lorem ipsum" ou autre*) à l'aide de polices téléchargeables disponibles en ligne.

La liste des polices devra être obtenue dynamiquement par interrogation du service en ligne via AJAX.

Attention police !

Philosopher

- ☒ regular
☐ italic
☐ 700
☐ 700italic

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis fringilla nisi ut ligula consequat consectetur. Donec sodales augue ligula, id accumsan turpis efficitur ac. In non ante sit amet odio auctor lacinia. Morbi fringilla, sem vel tempus gravida, elit mauris consequat felis, accumsan rhoncus magna nisi vitae nisl. Aenean interdum et elit tempus pharetra. Sed eu viverra lectus, et congue lectus. Suspendisse potenti. Proin finibus dui erat. Phasellus porttitor ligula vitae felis consequat, non vulputate metus consectetur. Vestibulum nec eros maximus, hendrerit turpis condimentum, lacinia massa. Nam metus ante, pulvinar ut ipsum non, fermentum pretium sem. Vivamus ut nulla vel metus malesuada efficitur. Vivamus eget mi arcu.

à suivre...

Références et liens

[Dynamic HTML], p.3

"Dynamic HTML", Wikipédia, 30 avril 2018.

En ligne, disponible sur https://en.wikipedia.org/wiki/Dynamic_HTML

[consulté le 29 août 2018]

[DOM specs], p.3

PH. LE HÉGARET, "Document Object Model (DOM) Technical Reports", W3C Architecture Domain, 2 mai 2012.

En ligne, disponible sur <https://www.w3.org/DOM/DOMTR>

[consulté le 29 août 2018]

[DOM Level 1], p.3

L.WOOD *et al.*, "Document Object Model (DOM) Level 1 Specification", W3C Recommendation, 1 Oct. 1998.

En ligne, disponible sur <https://www.w3.org/TR/REC-DOM-Level-1/>

[consulté le 29 août 2018]

[DOM Core], p.3

M. CHAMPION, S. BYRNE, G. NICOL, L.WOOD (ÉD.), "Document Object Model (Core) Level 1", W3C Recommendation, 1 Oct. 1998.

En ligne, disponible sur <https://www.w3.org/TR/REC-DOM-Level-1/level-one-core.html>

[consulté le 29 août 2018]

[DOM HTML], p.3

M. CHAMPION, V. APPARAO, S. ISAACS, C. WILSON, I. JACOBS (ÉD.), "Document Object Model (HTML) Level 1", W3C Recommendation, 1 Oct. 1998.

En ligne, disponible sur <https://www.w3.org/TR/REC-DOM-Level-1/level-one-html.html>

[consulté le 29 août 2018]

[DOM Level 2], p.3

A.LE HORS, PH.LE HÉGARET, L.WOOD *et al.*, "Document Object Model (DOM) Level 2 Core Specification", W3C Recommendation, 13 Nov. 2000.

En ligne, disponible sur <https://www.w3.org/TR/DOM-Level-2-Core/>

[consulté le 29 août 2018]

[DOM Level 3], p.3

A. LE HORS, P. LE HÉGARET, L. WOOD *et al.*, "Document Object Model (DOM) Level 3 Core Specification", W3C Recommendation, 7 Avr. 2004.

En ligne, disponible sur <https://www.w3.org/TR/DOM-Level-3-Core/>

[consulté le 29 août 2018]

[DOM Level 3 Events], p.3

G. KACMARCIK, T. LEITHEAD (ÉD.), "UI Events", W3C Working Draft, 4 août 2016.

En ligne, disponible sur <https://www.w3.org/TR/DOM-Level-3-Events/>

[consulté le 29 août 2018]

[DOM Level 4], p.3

A.VAN KESTEREN, A.GREGOR, MS2GER, A.RUSSEL, R.BERJON (ÉD.), "W3C DOM4", W3C Public Working Draft, 19 Nov. 2015.

En ligne, disponible sur <https://www.w3.org/TR/dom/>

[consulté le 29 août 2018]

[mozilla], p.3

E. SHEPHERD *et al.*, "Document Object Model (DOM)", Mozilla Developer Network, 16 août 2018.

En ligne, disponible sur <https://developer.mozilla.org/en-US/docs/DOM/Levels>

[consulté le 29 août 2018]

[DOM Living Standard], p.3

"DOM - Living standard", whatwg.org, 8 août 2018.

En ligne, disponible sur <https://dom.spec.whatwg.org/>

[consulté le 29 août 2018]

[sérialisation], p.4

"Sérialisation", Wikipédia, 9 mai 2018.

En ligne, disponible sur <http://fr.wikipedia.org/wiki/Sérialisation>

[consulté le 29 août 2018]

[démon], p.4

D. MULLER, "Exemple de document HTML5", 4 fév. 2014, 1 p.

En ligne, disponible sur <http://dmolinarius.github.io/demofiles/elc-d3/cours3/html5.html>

[consulté le 29 août 2018]

[WHATWG Node], p.5

"DOM Living standard - Interface Node", whatwg.org, 8 août 2018.

En ligne, disponible sur <https://dom.spec.whatwg.org/#interface-node>

[consulté le 29 août 2018]

[DOM3 Node], p.5

A.LE HORS, PH.LE HÉGARET et al., "Document Object Model Core - Interface Node", W3C Recommendation, 7 Avr. 2004.

En ligne, disponible sur <https://www.w3.org/TR/DOM-Level-3-Core/core.html#ID-1950641247>

[consulté le 29 août 2018]

[démon], p.5

D. MULLER, "Interface Node - Propriétés d'introspection", 29 août 2018, 1 p.

En ligne, disponible sur <http://dmolinarius.github.io/demofiles/elc-d3/cours3/introspection.html>

[consulté le 29 août 2018]

[démon], p.5

D. MULLER, "Interface Node - Parcours de l'arbre", 29 août 2018, 1 p.

En ligne, disponible sur <http://dmolinarius.github.io/demofiles/elc-d3/cours3/nodetree.html>

[consulté le 29 août 2018]

[démon], p.6

D. MULLER, "Interface Node - Document et élément racine", 30 août 2018, 1 p.

En ligne, disponible sur http://dmolinarius.github.io/demofiles/elc-d3/cours3/document_root.html

[consulté le 30 août 2018]

[démon], p.6

D. MULLER, "Interface Document - Méthodes de requêtage", 29 août 2018, 1 p.

En ligne, disponible sur <http://dmolinarius.github.io/demofiles/elc-d3/cours3/requetage.html>

[consulté le 29 août 2018]

[HTML5], p.6

"HTML Living Standard", WHATWG (Apple, Google, Mozilla, Microsoft), 6 sept. 2018.

En ligne, disponible sur <https://html.spec.whatwg.org/>

[consulté le 10 sept. 2018]

[Selectors-API], p.6

A.VAN KESTEREN, L.HUNTS, "Selectors API Level 1", W3C Recommendation, Feb. 21, 2013.

En ligne, disponible sur <https://www.w3.org/TR/selectors-api/>

[consulté le 10 sept. 2018]

[d  mo], p.6

D. MULLER, "*Interface Document - querySelector et querySelectorAll*", 29 ao  t 2018, 1 p.

En ligne, disponible sur <http://dmolinarius.github.io/demofiles/elc-d3/cours3/querySelector.html>

[consult   le 29 ao  t 2018]

[d  mo], p.8

D. MULLER, "*Interface Document - Modification dynamique de l'arbre*", 29 ao  t 2018, 1 p.

En ligne, disponible sur <http://dmolinarius.github.io/demofiles/elc-d3/cours3/appendChild.html>

[consult   le 29 ao  t 2018]

[d  mo], p.8

D. MULLER, "*DocumentFragment*", 30 ao  t 2018, 1 p.

En ligne, disponible sur <http://dmolinarius.github.io/demofiles/elc-d3/cours3/documentFragment.html>

[consult   le 30 ao  t 2018]

[DOM2 HTML], p.9

J. STENBACK, PH. LE H  GARET, A. LE HORS (  D.), "*Document Object Model (DOM) Level 2 HTML Specification*", W3C Recommendation, 9 Jan. 2003.

En ligne, disponible sur <https://www.w3.org/TR/DOM-Level-2-HTML/>

[consult   le 30 ao  t 2018]

[d  mo], p.10

D. MULLER, "*Interface HTMLDocument*", 30 ao  t 2018, 1 p.

En ligne, disponible sur <http://dmolinarius.github.io/demofiles/elc-d3/cours3/resource-metadata.html>

[consult   le 30 ao  t 2018]

[DOM2 Style], p.11

CH. WILSON, PH. LE H  GARET, V. APPARAO (  D.), "*Document Object Model (DOM) Level 2 Style Specification*", W3C Recommendation, 13 Nov. 2000.

En ligne, disponible sur <https://www.w3.org/TR/DOM-Level-2-Style/>

[consult   le 04 mars 2019]

[getComputedStyle()], p.11

M. KABAB *et al.*, "*Window.getComputedStyle()*", Mozilla Developer Network, 24 jan. 2019.

En ligne, disponible sur <https://developer.mozilla.org/en-US/docs/Web/API/Window/getComputedStyle>

[consult   le 04 mars 2019]

[CSSOM], p.11

D. GLAZMAN, E. COBOS   LVAREZ, *et al.*, "*CSS Object Model (CSSOM)*", W3C Editor's Draft, 21 jan. 2019.

En ligne, disponible sur <https://drafts.csswg.org/cssom/#dom-window-getcomputedstyle>

[consult   le 3 mars. 2019]

[GlobalEventHandlers], p.12

"*HTML Living Standard - Global Event Handlers*", WHATWG (Apple, Google, Mozilla, Microsoft), 6 sept. 2018.

En ligne, disponible sur <https://html.spec.whatwg.org/#globoleventhandlers>

[consult   le 7 sept. 2018]

[DocumentAndElementEventHandlers], p.12

"*HTML Living Standard - Document and Element Event Handlers*", WHATWG (Apple, Google, Mozilla, Microsoft), 6 sept. 2018.

En ligne, disponible sur <https://html.spec.whatwg.org/#documentandelementeventhandlers>

[consult   le 7 sept. 2018]

[DOM Level 2 Events], p.12

T. PIXLEY (ÉD.), *"Document Object Model (DOM) Level 2 Events Specification"*, W3C Recommendation, 13 nov. 2000.

En ligne, disponible sur <https://www.w3.org/TR/DOM-Level-2-Events/>
[consulté le 30 août 2018]

[event order], p.15

S. CHEDYGOV, *"Are event handlers in JavaScript called in order ?"*, Stack Overflow, 9 oct. 2014.

En ligne, disponible sur <https://stackoverflow.com/questions/2706109/are-event-handlers-in-javascript-called-in-order>
[consulté le 8 sept. 2018]

[Event], p.16

A. VAN KESTEREN (ÉD.), *"W3C DOM4"*, W3C Recommendation, 19 nov. 2015, § 3.2 Interface Event.

En ligne, disponible sur <https://www.w3.org/TR/dom/#event>

[UI Events], p.16

G. KACMARCIK, T. LEITHEAD (ÉD.), *"UI Events"*, W3C Working Draft, 4 août 2016.

En ligne, disponible sur <https://www.w3.org/TR/DOM-Level-3-Events/>
[consulté le 29 août 2018]

[position curseur], p.18

NOBODY (OOPS!), *"CSSOM View Module"*, W3C Editor's Draft, 1 sept. 2018.

En ligne, disponible sur <https://drafts.csswg.org/cssom-view/#dom-element-getboundingclientrect>
[consulté le 9 sept. 2018]

[getBoundingClientRect], p.18

B. SHELOCK, *"Find mouse position relative to element"*, Stack Overflow, 13 juil. 2010.

En ligne, disponible sur <https://stackoverflow.com/questions/3234256/find-mouse-position-relative-to-element>
[consulté le 9 sept. 2018]

[ordre des événements], p.18

G. KACMARCIK, T. LEITHEAD (ÉD.), *"UI Events"*, W3C Working Draft, 4 août 2016, § 4.3.3 Mouse Event Order.

En ligne, disponible sur <https://www.w3.org/TR/DOM-Level-3-Events/#events-mouseevent-event-order>
[consulté le 10 sept. 2018]

[contenteditable], p.19

"HTML Living Standard", WHATWG (Apple, Google, Mozilla, Microsoft), 6 sept. 2018, § 6.6.1

Making document regions editable: The contenteditable content attribute.

En ligne, disponible sur <https://html.spec.whatwg.org/multipage/interaction.html#contenteditable>
[consulté le 10 sept. 2018]

[Can I Use], p.19

A. DEVERIA, *"Can I use ... ?"*, caniuse.com.

En ligne, disponible sur <http://caniuse.com>
[consulté le 10 sept. 2018]

[notes de compatibilité], p.19

M. BRINKHUIS *et al.*, *"input"*, Mozilla Developer Network, 15 juil. 2018, § Browser compatibility.

En ligne, disponible sur https://developer.mozilla.org/en-US/docs/Web/Events/input#Browser_compatibility
[consulté le 10 sept. 2018]

[codes clavier], p.19

G. KACMARCIK, T. LEITHEAD (ÉD.), *"UI Events KeyboardEvent code Values"*, W3C Candidate Recommendation, 1er juin 2017.

En ligne, disponible sur <https://www.w3.org/TR/uievents-code/>
[consulté le 10 sept. 2018]

[Can I use keyboard], p.19

@FYRD (ALEXIS DEVERIA), *"Can I use keyboard ?"*, caniuse.com, 29 août. 2018.

En ligne, disponible sur <https://caniuse.com/#search=keyboard>
[consulté le 10 sept. 2018]

[images nommées], p.20

D. MULLER, *"Images nommées : exemple de solution"*, 6 mars 2019.

En ligne, disponible sur <http://dmolinarius.github.io/demofiles/elc-d3/cours3/colors.html>
[consulté le 6 mars 2019]

[en ligne], p.20

D. MULLER, *"liste des couleurs nommées"*, 5 mars 2019.

En ligne, disponible sur http://dmolinarius.github.io/demofiles/elc-d3/cours3/named_colors.json
[consulté le 5 mars 2019]

[télécran], p.20

"Télécran", Wikipédia, 13 fév. 2019.

En ligne, disponible sur <https://fr.wikipedia.org/wiki/T%C3%A9l%C3%A9cran>
[consulté le 5 mars 2019]

[télécran], p.20

D. MULLER, *"télécran : exemple de solution"*, 19 sept. 2018.

En ligne, disponible sur <http://dmolinarius.github.io/demofiles/elc-d3/cours3/telecran.html>
[consulté le 6 mars 2019]

[taquin], p.20

"Taquin", Wikipédia, 17 oct. 2018.

En ligne, disponible sur <https://fr.wikipedia.org/wiki/Taquin>
[consulté le 5 mars 2019]

[taquin sans image], p.20

D. MULLER, *"Taquin : exemple sans image"*, 10 juil. 2018.

En ligne, disponible sur <http://dmolinarius.github.io/demofiles/elc-d3/cours3/taquin1.html>
[consulté le 6 mars 2019]

[taquin complet], p.20

D. MULLER, *"Taquin : exemple complet"*, 19 oct. 2018.

En ligne, disponible sur <http://dmolinarius.github.io/demofiles/elc-d3/cours3/taquin2.html>
[consulté le 6 mars 2019]

[démo Ajax], p.21

D. MULLER, *"Page HTML avec contenu dynamique via Ajax"*, 29 jan. 2014, 1 p.

En ligne, disponible sur <http://dmolinarius.github.io/demofiles/elc-d3/cours3/example1.html>
[consulté le 14 fév. 2016]

[JSON], p.21

"Introducing JSON", Juil. 2006.

En ligne, disponible sur <http://www.json.org>
[consulté le 14 fév. 2016]

[RFC4627], p.21

D. CROCKFORD, "*RFC 4627 - The application/json Media Type for JavaScript Object Notation (JSON)*", Juil. 2006.

En ligne, disponible sur <http://www.ietf.org/rfc/rfc4627.txt>

[consulté le 14 fév. 2016]

[XMLHttpRequest], p.21

SMUTT *et al.*, "*XMLHttpRequest*", Mozilla Developer Network, 4 Fév. 2016.

En ligne, disponible sur <https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest>

[consulté le 1 mars 2019]

[Using XMLHttpRequest], p.21

H. SIVONEN *et al.*, "*Using XMLHttpRequest*", Mozilla Developer Network, 18 Oct. 2018.

En ligne, disponible sur https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest/Using_XMLHttpRequest

[consulté le 1 mars 2019]

[article fondateur], p.21

JESSE JAMES GARRET, "*Ajax: A New Approach to Web Applications*", Fév. 2005.

En ligne, disponible sur <https://web.archive.org/web/20080702075113/http://www.adaptivepath.com/ideas/essays/archives/000385.php>

[consulté le 1 mars 2019]

[XMLHttpRequest living standard], p.21

A. VAN KESTEREN, "*XMLHttpRequest Living Standard*", WHATWG, 18 Feb. 2019.

En ligne, disponible sur <https://xhr.spec.whatwg.org/>

[consulté le 01 mars 2019]

[Wikipédia], p.22

"*Cross-origin resource sharing*", Wikipédia, 3 Jan. 2019.

En ligne, disponible sur https://en.wikipedia.org/wiki/Cross-origin_resource_sharing

[consulté le 1 mars 2019]

[Recommandation], p.22

A. VAN KESTEREN, "*Cross-origin Resource Sharing*", W3C Candidate Recommendation, 16 Jan. 2014.

En ligne, disponible sur <https://www.w3.org/TR/cors/>

[consulté le 1 mars 2019]

[crossorigin.me], p.22

C. HUDSON, "*crossorigin.me*", 24 Oct. 2017.

En ligne, disponible sur <https://corsproxy.github.io/>

[consulté le 1 mars 2019]

[MDN CORS], p.22

M. FUJIMOTO *et al.*, "*Cross-Origin Resource Sharing (CORS)*", Mozilla Developer Network, 18 fév. 2019.

En ligne, disponible sur <https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS>

[consulté le 1 mars 2019]

[organigramme], p.22

BLUESMOON, "*Flowchart showing Simple and Preflight XHR*", Wikimedia Commons, 14 août 2015.

En ligne, disponible sur https://upload.wikimedia.org/wikipedia/commons/c/ca/Flowchart_showing_Simple_and_Preflight_XHR.svg

[consulté le 1 mars 2019]

[<https://drstearns.github.io/tutorials/cors/>], p.23

DAVE STEARNS, "*Cross-Origin Resource Sharing - Enable JavaScript from different origins to call your APIs*", 14 mars 2018.

En ligne, disponible sur <https://drstearns.github.io/tutorials/cors/>
[consulté le 1 mars 2019]

[en ligne], p.25

P.A. CHAMPIN, "*Un livre dont vous êtes le héros*", 20 déc. 2018.

En ligne, disponible sur <http://dmolinarius.github.io/demofiles/elc-d3/cours3/livre.tar.gz>
[consulté le 5 mars 2019]

[exemple de livre dont vous êtes le héros], p.26

D. MULLER, "*Le livre dont vous êtes le héros : exemple de solution*", 10 juil. 2018.

En ligne, disponible sur <http://dmolinarius.github.io/demofiles/elc-d3/cours3/livre.html>
[consulté le 6 mars 2019]

[APIs], p.26

XUXINTAO *et al.*, "*Client-side web APIs*", Mozilla Developer Network, 19 juin 2018.

En ligne, disponible sur https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Client-side_web_APIs
[consulté le 5 mars 2019]

[media APIs], p.26

K. JONES *et al.*, "*Video and audio APIs*", Mozilla Developer Network, 8 oct. 2018.

En ligne, disponible sur https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Client-side_web_APIs/Video_and_audio_APIs
[consulté le 5 mars 2019]

[HTMLMediaElement], p.27

"*HTML - Living Standard*", WHATWG, 4 mars 2019, § 4.8.12 Media Elements.

En ligne, disponible sur <https://html.spec.whatwg.org/multipage/media.html#media-elements>
[consulté le 5 mars 2019]

[exemple en ligne], p.27

DANIEL MULLER, "*Exemple de document avec vidéo*", Exemple de cours, 17 juil. 2018.

En ligne, disponible sur <http://dmolinarius.github.io/demofiles/elc-d3/cours3/exemple-video.html>
[consulté le 5 mars 2019]

[canvas API], p.28

B. MINARD *et al.*, "*Canvas API*", Mozilla Developer Network, 3 mars 2019.

En ligne, disponible sur https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API
[consulté le 5 mars 2019]

[WebGL], p.28

"*WebGL Overview*", Khronos Group, 2019.

En ligne, disponible sur <https://www.khronos.org/webgl/>
[consulté le 5 mars 2019]

[tutorials], p.28

B. MINARD *et al.*, "*Canvas API*", Mozilla Developer Network, 3 mars 2019, § Guides and tutorials.

En ligne, disponible sur https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API#Guides_and_tutorials
[consulté le 5 mars 2019]

[tutorials], p.28

"*HTML5 Canvas Tutorials*", Html5CanvasTutorials, 2018.

En ligne, disponible sur <https://www.html5canvastutorials.com/>
[consulté le 5 mars 2019]

