Programmation WEB

DOM et api javascript

Programmation licence

IUT de Fontainebleau

19 novembre 2014

Sommaire

Sommaire

Pourquoi ?

- Chaque navigateur implantait **ses propres méthodes** de manipulation du contenu html. D'où la nécessité :
  - Uniformiser et abstraire (indépendante de tout langage) la représentation d'un document html (et xml)
  - Spécifier une api de manipulation de cette représentation.
- Le Dom **Document Object Model**

Le DOM fournit :

- une représentation objet normalisée des documents html et xml, **sous forme arborescente**.
- une **api** qui permet **d'accéder** au document et de **manipuler** son contenu, sa structure et ses styles.
- permet ainsi d'interfacer un document avec un langage, comme javascript, mais aussi python, php, java, etc....

- Plusieurs niveaux de spécification ont vu le jour :
  - DOM 1 (1998) : manipulation d'un document html ou xml.
  - DOM 2 (2001) : dernière version finalisée : ajout de méthodes de parcours de l'arbre, gestion des evénements et des feuilles de styles, vues filtrées.
  - DOM 3 : en cours de spécification. interface de chargement et de sauvegarde de documents xml, XPath...
- Tous les navigateurs supportent le DOM 2.

Quelques références

`https://developer.mozilla.org/fr/Référence_du_DOM_Gecko`
`http://www.w3.org/DOM`

Sommaire

Structure du DOM

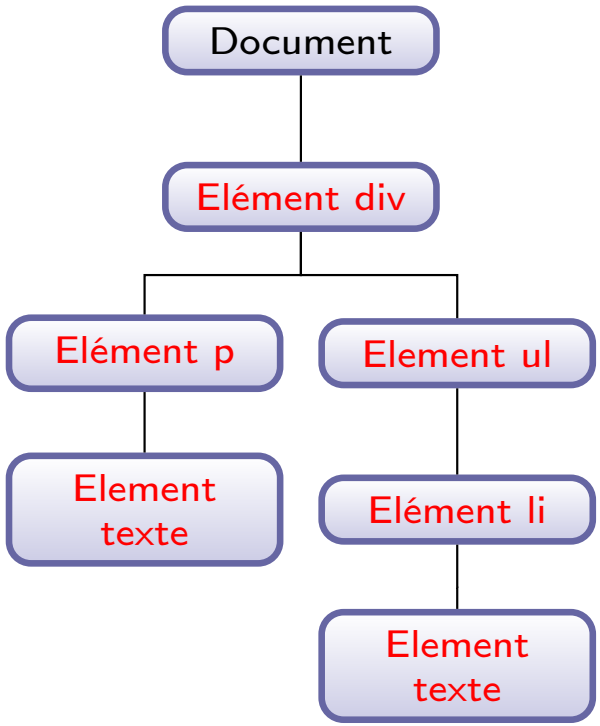
Structure arborescente **de noeuds** (node).

- Chaque noeud est un objet, avec des méthodes et des attributs.
- Cette interface est implanté pour plusieurs langages, comme javascript, php, java, python, perl, activeX.
- Les noms des interfaces, classes, méthodes et propriétés sont indépendantes du langages.
- Interface DOM pour d'autres documents (xml) du WEB :
  - MathML
  - SVG
  - X3D
  - ...

Un exemple

Représentation hiérarchique sous forme d'arbre.

```
<div id="exemple">
  <p>un paragraphe</p>
  <ul>
    <li>un element de liste</li>
  </ul>
</div>
```



Classe Node

Classe centrale représentant un noeud DOM.

```
interface Node {

  // NodeType
  const unsigned short      ELEMENT_NODE                   = 1;
  const unsigned short      ATTRIBUTE_NODE                 = 2;
  const unsigned short      TEXT_NODE                      = 3;
  const unsigned short      CDATA_SECTION_NODE             = 4;
  const unsigned short      ENTITY_REFERENCE_NODE          = 5;
  const unsigned short      ENTITY_NODE                    = 6;
  const unsigned short      PROCESSING_INSTRUCTION_NODE    = 7;
  const unsigned short      COMMENT_NODE                   = 8;
  const unsigned short      DOCUMENT_NODE                  = 9;
  const unsigned short      DOCUMENT_TYPE_NODE             = 10;
  const unsigned short      DOCUMENT_FRAGMENT_NODE         = 11;
  const unsigned short      NOTATION_NODE                  = 12;

  readonly attribute DOMString        nodeName;
  attribute DOMString         nodeValue;
  // raises(DOMException) on setting
  // raises(DOMException) on retrieval
  readonly attribute unsigned short   nodeType;
  readonly attribute Node             parentNode;
  readonly attribute NodeList         childNodes;
```

Classe Node

```
readonly attribute Node             firstChild;
readonly attribute Node             lastChild;
readonly attribute Node             previousSibling;
readonly attribute Node             nextSibling;
readonly attribute NamedNodeMap     attributes;
// Modified in DOM Level 2:
readonly attribute Document         ownerDocument;
Node                insertBefore(in Node newChild,
    in Node refChild)
raises(DOMException);
Node                replaceChild(in Node newChild,in Node oldChild)
  raises(DOMException);
Node                removeChild(in Node oldChild)
  raises(DOMException);
Node                appendChild(in Node newChild)
  raises(DOMException);
  boolean             hasChildNodes();
  Node                cloneNode(in boolean deep);
  // Modified in DOM Level 2:
  void                normalize();
  // Introduced in DOM Level 2:
  boolean             isSupported(in DOMString feature,in DOMString version);
```

```
// Introduced in DOM Level 2:
readonly attribute DOMString        namespaceURI;
// Introduced in DOM Level 2:
attribute DOMString        prefix;
// raises(DOMException) on setting

// Introduced in DOM Level 2:
readonly attribute DOMString        localName;
// Introduced in DOM Level 2:
boolean            hasAttributes();
```

- D'autres classes dérivent, en la spécialisant, de node.
- Certains noeuds peuvent avoir des fils, d'autres représentent des feuilles.

DOM Core

Les noeuds, et leurs enfants

`Document`	Element (un, au maximum), ProcessingInstruction, Comment, DocumentType(un, au maximum)
`DocumentFragment`	Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference
`DocumentType`	aucun enfant
`EntityReference`	Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference
`Element`	Element, Text, Comment, ProcessingInstruction, CDATASection, EntityReference
`Attr`	Text, EntityReference
`ProcessingInstruction`	aucun enfant
`Comment`	aucun enfant
`Text`	aucun enfant
`CDATASection`	aucun enfant
`Entity`	Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference
`Notation`	aucun enfant

DOM Core

- Pour chaque classe, il existe des méthodes et propriétés pour accéder aux données et les modifier.
- DOM HTML spécialise Le DOM CORE.

Sommaire

Accéder à un noeud

Directement ...

- par un nom d'élément :

```
NodeList getElementsByTagName(in DOMString tagname);
```

avec l'interface correspondante à une liste de noeud :

```
interface NodeList {
  Node                item(in unsigned long index);
  readonly attribute unsigned long    length;
};
```

- par un identifiant : (DOM HTML)

```
Element getElementById(in DOMString elementId);
```

ou ...

Accéder à un noeud

... en parcourant l'arbre avec les propriétés de la classe node

```
readonly attribute Node          parentNode;
readonly attribute NodeList      childNodes;
readonly attribute Node          firstChild;
readonly attribute Node          lastChild;
readonly attribute Node          previousSibling;
readonly attribute Node          nextSibling;
readonly attribute NamedNodeMap  attributes;
```

Firefox stocke l'ensemble des espaces du document html sous forme de noeuds texte. il faut les prendre en compte lors des traitements.

Créer un noeud

Dans la classe document

- un noeud élément :

```
Element                createElement(in DOMString tagName)
  raises(DOMException);
```

- un noeud texte :

```
Text                   createTextNode(in DOMString data);
```

- un noeud attribut :

```
Attr                   createAttribute(in DOMString name)
  raises(DOMException);
```

Créer un noeud

On peut également dupliquer un noeud avec la méthode `cloneNode` de la classe `node` :

```
var elem=document.getElementById("mon_div");
var sous_arbre=elem.cloneNode(true);
var div=elem.cloneNode(false);
```

Ajouter/Supprimer

- Méthode de la classe node :

```
Node insertBefore(
      in Node newChild,
      in Node refChild)
raises(DOMException);

Node replaceChild(
      in Node newChild,
      in Node oldChild)
raises(DOMException);

Node removeChild(in Node oldChild)
  raises(DOMException);

Node appendChild(in Node newChild)
  raises(DOMException);
```

1. Arbre initial

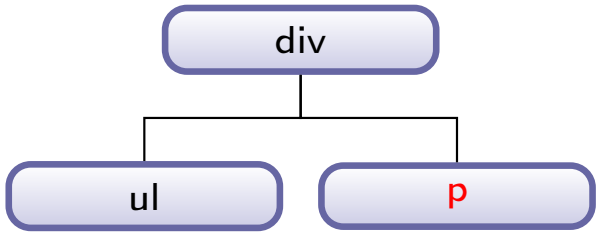
```
<div id="mon_div">
  <ul><li>toto</li></ul>
</div>
```

2. Execution du code

```
var zone=document.getElementById("mon_div");
var p=document.createElement("p");
var texte=document.createTextNode("blablabla ...");
p.appendChild(texte);
zone.appendChild(p);
```

3. Nouvel arbre

```
<div id="mon_div">
  <ul><li>toto</li></ul>
  <p>blablablabla ...</p>
  </div>
```



Attributs

- tout élément (node de type NODE_ELEMENT) peut contenir des attributs, c'est à dire une paire (clé,valeur) rajoutant des informations.

```
interface Element : Node {
  readonly attribute DOMString        tagName;
  DOMString           getAttribute(in DOMString name);
  void                setAttribute(in DOMString name,
      in DOMString value)
    raises(DOMException);
  void                removeAttribute(in DOMString name)
    raises(DOMException);
  Attr                getAttributeNode(in DOMString name);
  Attr                setAttributeNode(in Attr newAttr)
    raises(DOMException);
  Attr                removeAttributeNode(in Attr oldAttr)
    raises(DOMException);
```

- On peut savoir si un noeud en posséde avec la requête :

```
boolean             hasAttribute(in DOMString name);
```

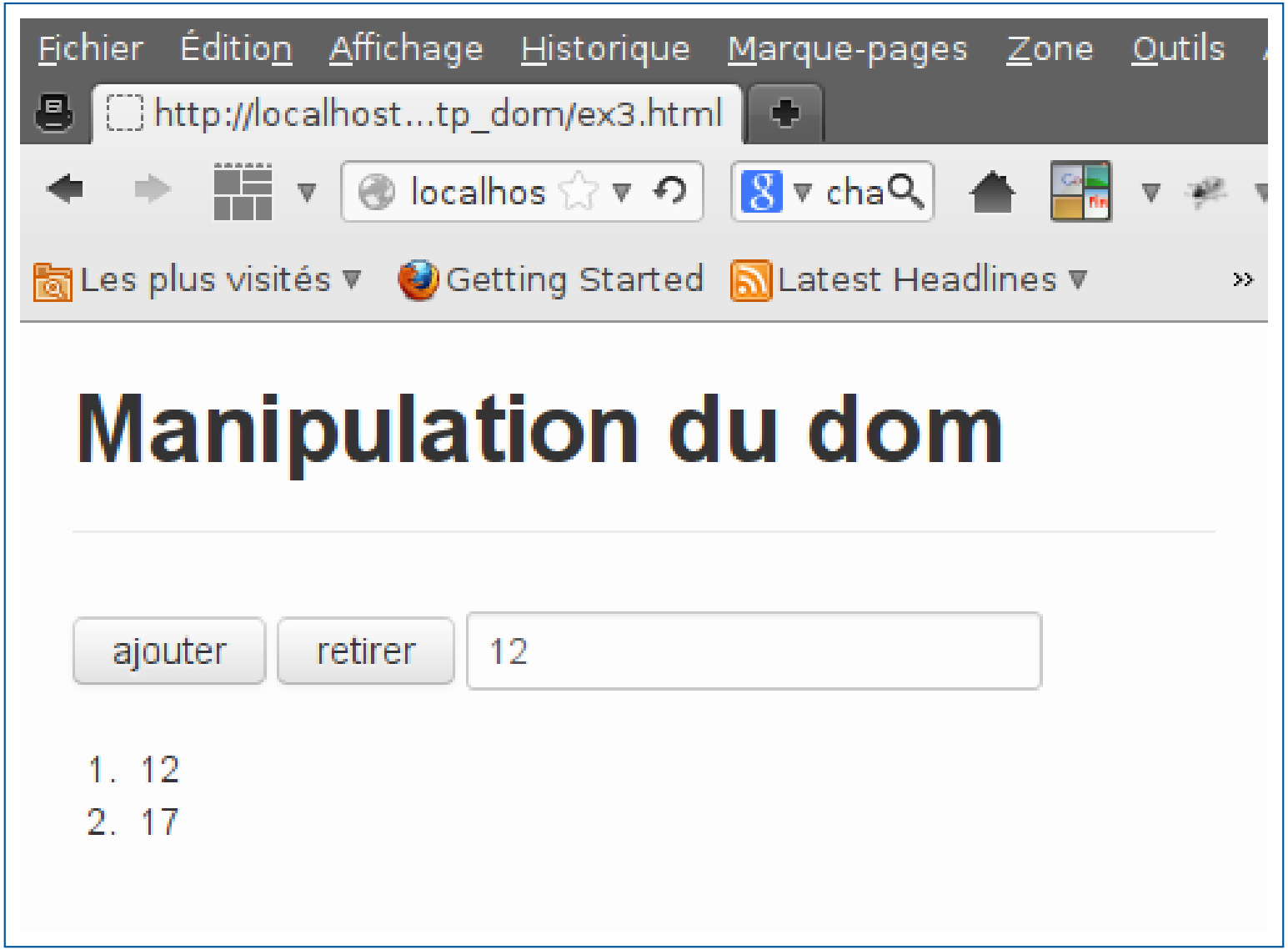
Exemple

```
<!DOCTYPE html>
<html>
  <head>
    <LINK href="/bootstrap/css/bootstrap.css" rel="stylesheet" type="text/css">
  </head>
  <body>
    <div class="container-fluid">
      <div class="page-header">
        <h1>Manipulation du dom</h1>
      </div>
      <form class="form-inline">
        <input class="btn" type=button value="ajouter" onclick="ajouter('ma_liste');">
        <input class="btn" type=button value="retirer"onclick="retirer('ma_liste');">
        <input type=text id="survol" value="survol">
      </form>
      <ol  id="ma_liste"></ol>

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

Exemple : suite

```
function affiche(ev)
{
  var li=ev.target;
  document.getElementById("survol").value=li.childNodes.item(0).nodeValue;
}
function ajouter(id)
{
  var li=document.createElement("li");
  var texte=document.getElementById("survol").value;
  var t=document.createTextNode(texte);
  li.appendChild(t);
  document.getElementById(id).appendChild(li);
  li.addEventListener("mouseover",affiche,false);
}
function retirer(id)
{
  var n=document.getElementById("survol").value;
  var ol=document.getElementById(id);
  ol.removeChild(ol.childNodes.item(n-1));
}
```


Fichier Édition Affichage Historique Marque-pages Zone Outils
http://localhost...tp_dom/ex3.html
Les plus visités
Getting Started
Latest Headlines
Manipulation du dom
ajouter
retirer
12
1. 12
2. 17

Sous-arbre

On peut manipuler des sous-arbres avec l'objet DocumentFragment.

```
var fragment=document.createDocumentFragment();
```

On utilise un fragment comme un noeud.

```
var zone=document.getElementById("ma_zone");
var fragment=document.createDocumentFragment();
var jours=["lundi","mardi","mercredi","jeudi","vendredi",
  "samedi","dimanche"];
for(i=0;i<jours.length;i++) {
  var p=document.createElement("p");
  var texte=document.createTextNode(jours[i]);
  p.appendChild(texte);
  fragment.appendChild(p);
}
zone.appendChild(fragment);
```

```
zone.appendChild(fragment);
```

Très utile lorsque l'on aurait été obligé de modifier le dom après chaque insertion de noeud, comme c'est le cas dans l'exemple précédent.
Permet de gagner en performance.
Il est important de savoir comment une librairie qui abstrait le dom gére sa modification.

XML

Javascript permet :

- de créer des documents xml, vierges ou à partir d'un fichier xml local.
- de le parser avec l'api DOM.
- de le transformer avec xslt.

le noeud document doit posséder selon DOM2 un attribut implementation de type DOMImplementation dont voici l'interface :

```
interface DOMImplementation {
  boolean            hasFeature(in DOMString feature,
      in DOMString version);
  // Introduced in DOM Level 2:
  DocumentType       createDocumentType(in DOMString qualifiedName,
      in DOMString publicId,
      in DOMString systemId)
    raises(DOMException);
  // Introduced in DOM Level 2:
  Document           createDocument(in DOMString namespaceURI,
      in DOMString qualifiedName,
      in DocumentType doctype)
    raises(DOMException);
};
```

```
xmlDoc=document.implementation.createDocument("","",null);
xmlDoc.async=false;
xmlDoc.load("cia.xml");
var continent=xmlDoc.getElementsByTagName('continent');
var liste=document.createElement("ul");
for(i=0;i<continent.length;i++)
{
  li=document.createElement("li");
  texte=document.createTextNode(
      continent.item(i).getAttributeNode('name').nodeValue
      );
  li.appendChild(texte);
  liste.appendChild(li);
}
document.getElementById("continent").appendChild(liste);
```

Sommaire

Dom HTML

Le DOM HTML est une extension du DOM CORE. Le but :

- Spécialiser et ajouter des fonctionnalités spécifiques aux documents et éléments HTML.
- Assurer la compatibilité avec le DOM 0.
- Ajouter des mécanismes, des traitements utiles, communs et pratiques dans le cas spécifique d'HTML.

DOM HTML

- `HTMLDocument` dérive de `Document` du DOM CORE.
- `HTMLElement` dérive de `Element` du DOM CORE.
- Nouvelle Interface pour chaque type d'éléments html qui spécialise `HTMLElement`.

HTMLDocument

```
interface HTMLDocument : Document {
  attribute DOMString       title;
  readonly attribute DOMString       referrer;
  readonly attribute DOMString       domain;
  readonly attribute DOMString       URL;
  attribute HTMLElement     body;
  readonly attribute HTMLCollection  images;
  readonly attribute HTMLCollection  applets;
  readonly attribute HTMLCollection  links;
  readonly attribute HTMLCollection  forms;
  readonly attribute HTMLCollection  anchors;
  attribute DOMString       cookie;
  // raises(DOMException) on setting

  void                  open();
  void                  close();
  void                  write(in DOMString text);
  void                  writeln(in DOMString text);
  NodeList              getElementsByName(in DOMString elementName);
};
```

HTMLElement

```
interface HTMLElement : Element {
  attribute DOMString       id;
  attribute DOMString       title;
  attribute DOMString       lang;
  attribute DOMString       dir;
  attribute DOMString       className;
};
```

La plupart des éléments html spécialisent cette classe. Voici deux exemples

HTMLTableElement

```
interface HTMLFormElement : HTMLElement {
  readonly attribute HTMLCollection  elements;
  readonly attribute long            length;
  attribute DOMString       name;
  attribute DOMString       acceptCharset;
  attribute DOMString       action;
  attribute DOMString       enctype;
  attribute DOMString       method;
  attribute DOMString       target;
  void                 submit();
  void                 reset();
};
```

HTMLTableElement

```
interface HTMLTableElement : HTMLElement {
  // Modified in DOM Level 2:
  attribute HTMLTableCaptionElement caption;
  // raises(DOMException) on setting

  // Modified in DOM Level 2:
  attribute HTMLTableSectionElement tHead;
  // raises(DOMException) on setting

  // Modified in DOM Level 2:
  attribute HTMLTableSectionElement tFoot;
  // raises(DOMException) on setting

  readonly attribute HTMLCollection  rows;
  readonly attribute HTMLCollection  tBodies;
  attribute DOMString       align;
  attribute DOMString       bgColor;
  attribute DOMString       border;
  attribute DOMString       cellPadding;
  attribute DOMString       cellSpacing;
  attribute DOMString       frame;
  attribute DOMString       rules;
  attribute DOMString       summary;
  attribute DOMString       width;
};
```

HTMLTableElement (suite)

```
interface HTMLTableElement : HTMLElement {

HTMLElement         createTHead();
  void                deleteTHead();
  HTMLElement         createTFoot();
  void                deleteTFoot();
  HTMLElement         createCaption();
  void                deleteCaption();
  // Modified in DOM Level 2:
  HTMLElement         insertRow(in long index)
    raises(DOMException);
  // Modified in DOM Level 2:
  void                deleteRow(in long index)
    raises(DOMException);
};
```

Spécifications complètes

http://www.w3.org/TR/DOM-Level-2-HTML/

Sommaire

But ?

Des événements peuvent être associé à des balises html en utilisant certains de leur attribut, dont le nom est préfixé par on (onload, onclick, etc...)

 mélange `(x)html` et `javascript` 

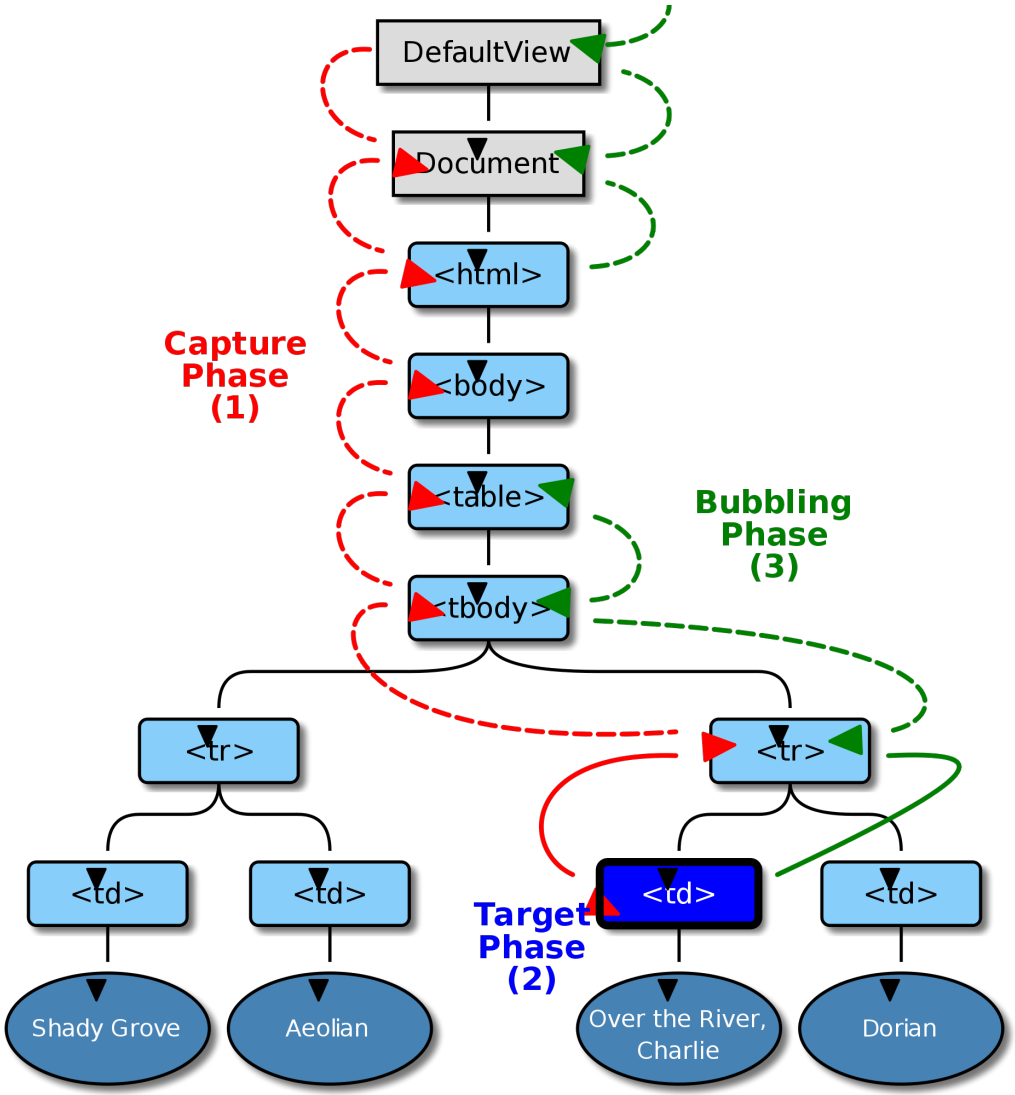
DOM 2 fournit un support à la gestion des événements.

Support DOM des evts

- Modéle générique qui permet :
  - l'enregistrement de handlers d'evénements.
  - de décrire le mécanisme de propagation des evénements dans une structure d'arbre.
  - de décrire une information contextuel pour chaque événement.
- Fournir un sous-ensemble commun pour ce qui est déjà utilisé dans DOM 0. (augmenter la portabilité)
- Pour savoir si une applicationle supporte, utiliser `hasFeature(feature, version)` de l'interface `DOMImplementation` avec hasFeature("Events","2.0")
- Consulter les pages

`http://www.w3.org/TR/DOM-Level-3-Events/`

Le flot


DefaultView
Document
<html>
<body>
<table>
<tbody>
<tr>
<tr>
<td>
<td>
<td>
<td>
Shady Grove
Aeolian
Over the River, Charlie
Dorian
Capture Phase (1)
Target Phase (2)
Bubbling Phase (3)

Le flot

- Basiquement, chaque événement posséde une cible (`EventTarget`) à qui il est délivré.
- **Tous** les handlers correspondant sont alors éxécutés (ordre ?).

Il existe deux modes de propagations :

Modes

`Ascendant`	nommé `bubbling`, bas niveau vers haut niveau.(I.E au début)
`Descendant`	nommé `capturing`, haut niveau vers bas niveau. (Netscape au début)

Le flot

- `capturing`
  Lors de la délivrance d'un événement à sa cible, l'événement peut être délivré à un noeud parent (en partant de la racine) si le noeud a enregistré une fonction reflexe en autorisant la capture avec le paramètre `useCapture` dans la fonction d'enregistrement (`addEventListener`).
  La propagation peut-être interrompu avec la méthode `stopProgagation` de la classe `Event`.
- `bubbling`
  Certains événements, après la phase de descente, sont alors délivrés aux parents de la cible (qui ont enregsitré une fonction reflexe), en remontant vers la racine. (ceux qui avaient capturé en sont exclus !)
  La propagation peut être interrompue avec la méthode `stopProgagation` de la classe `Event`.

- `Cancelation` : L'implémentation Dom (l'application) possède un traitement par défaut pour certains événements (click et hyperlien par exemple)
  Quand l'événement survient, les fonctions reflexes sont exécutées.
  Le traitement par défaut peut être alors inhibé par la méthode `preventDefault` de la classe `Event`.

- Gestion des gestionnaires d'événements.

```
// Introduced in DOM Level 2:
interface EventTarget {
  void                addEventListener(in DOMString type,
      in EventListener listener,
      in boolean useCapture);
  void                removeEventListener(in DOMString type,
      in EventListener listener,
      in boolean useCapture);
  boolean             dispatchEvent(in Event evt)
    raises(EventException);
};
```

- Avec l'interface EventListener

```
// Introduced in DOM Level 2:
interface EventListener {
  void                handleEvent(in Event evt);
};
```

Et l'interface Event

```
// Introduced in DOM Level 2:
interface Event {

  // PhaseType
  const unsigned short      CAPTURING_PHASE                 = 1;
  const unsigned short      AT_TARGET                       = 2;
  const unsigned short      BUBBLING_PHASE                  = 3;

  readonly attribute DOMString        type;
  readonly attribute EventTarget      target;
  readonly attribute EventTarget      currentTarget;
  readonly attribute unsigned short   eventPhase;
  readonly attribute boolean          bubbles;
  readonly attribute boolean          cancelable;
  readonly attribute DOMTimeStamp     timeStamp;
  void                stopPropagation();
  void                preventDefault();
  void                initEvent(in DOMString eventTypeArg,
      in boolean canBubbleArg,
      in boolean cancelableArg);
};
```

```
function ReponseClick(e)
{
/* traitement*/
}
var n=document.getElementById("mon_noeud");
n.addEventListener("click",ReponseClick,false);
/* pas de capture */

n.removeEventListener("click",ReponseClick,false);
};
```

DocumentEvent Interface

```
// Introduced in DOM Level 2:
interface DocumentEvent {
  Event                createEvent(in DOMString eventType)
    raises(DOMException);
};
```

Cette interface permet de simuler des évéments (click, etc...) par programmation.

Types d'evts

Il existe plusieurs catégories logiques d'événements :

Catégorie, type et description		
Evénement	Event	Evenement générique
Interface Utilisateur	UIEvent	DOMActivate,DOMFocusIn, DOMFocusOut et événements claviers
Evt souris	MouseEvent	click, mousedown, mouseup, mouseover, mouseout
Evt élément html	HTMLEvent	abort, blur, change, focus, error, load, unload, reset, scroll, select, submit, ...
Evt de mutation	MutationEvent	modification du DOM

Le modèle DOM offre la possibilité d'ajouter des catégories

User Interface events

User Interface event module est composé des événements listés dans HTML 4 et dans DOM 0.

```
// Introduced in DOM Level 2:
interface UIEvent : Event {
  readonly attribute views::AbstractView  view;
  readonly attribute long                 detail;
  void                initUIEvent(in DOMString typeArg,
      in boolean canBubbleArg,
      in boolean cancelableArg,
      in views::AbstractView viewArg,
      in long detailArg);
};
```

Ce type est dérivé lui-même en ...

MouseEvent

```
// Introduced in DOM Level 2:
interface MouseEvent : UIEvent {
  readonly attribute long             screenX;
  readonly attribute long             screenY;
  readonly attribute long             clientX;
  readonly attribute long             clientY;
  readonly attribute boolean          ctrlKey;
  readonly attribute boolean          shiftKey;
  readonly attribute boolean          altKey;
  readonly attribute boolean          metaKey;
  readonly attribute unsigned short   button;
  readonly attribute EventTarget      relatedTarget;
};
```

MouseEvent

```
(interface suite)
  void                initMouseEvent(in DOMString typeArg,
      in boolean canBubbleArg,
      in boolean cancelableArg,
      in views::AbstractView viewArg,
      in long detailArg,
      in long screenXArg,
      in long screenYArg,
      in long clientXArg,
      in long clientYArg,
      in boolean ctrlKeyArg,
      in boolean altKeyArg,
      in boolean shiftKeyArg,
      in boolean metaKeyArg,
      in unsigned short buttonArg,
      in EventTarget relatedTargetArg);
};
```

Evénements souris

Nom et description	
`click`	clic de souris dans l'élément
`mousedown, mouseup`	bouton enfoncé, relaché
`mouseover, mouseout`	le pointeur entre ou sort de l'élément
`mousemove`	déplacement du pointeur

- tous ces événements sont transmis aux ascendants (bubbling).
- seul `mousemove` n'a pas de traitement par défaut (cancelable).
- Il existe également une interface `WheelEvent` qui dérive de `MouseEvent`. Le nom de l'événement est `wheel`

Evénements clavier

- Il n'existe pas dans DOM 2 !
- Introduction dans DOM 3 (depuis 2009)
- L'interface correspondante est `KeyboardEvent`

```
// Introduced in DOM Level 3:
interface KeyboardEvent : UIEvent {

  // KeyLocationCode
  const unsigned long                DOM_KEY_LOCATION_STANDARD      = 0x00;
  const unsigned long                DOM_KEY_LOCATION_LEFT          = 0x01;
  const unsigned long                DOM_KEY_LOCATION_RIGHT         = 0x02;
  const unsigned long                DOM_KEY_LOCATION_NUMPAD        = 0x03;
  const unsigned long                DOM_KEY_LOCATION_MOBILE        = 0x04;
  const unsigned long                DOM_KEY_LOCATION_JOYSTICK      = 0x05;

  readonly attribute DOMString       char;
  readonly attribute DOMString       key;
  readonly attribute unsigned long   keyCode;
  readonly attribute unsigned long   location;
  readonly attribute boolean         ctrlKey;
  readonly attribute boolean         shiftKey;
  readonly attribute boolean         altKey;
  readonly attribute boolean         metaKey;
  readonly attribute boolean         repeat;
  boolean                            getModifierState(in DOMString keyArg);
  void                               initKeyboardEvent(in DOMString typeArg,
      in boolean canBubbleArg,
      in boolean cancelableArg,
      in views::AbstractView viewArg,
      in DOMString keyArg,
      in unsigned long locationArg,
      in DOMString modifiersListArg
```

Evénements clavier

Nom et description	
keydown keypress keyup	touches qui produisent un caractère

Mutation Events

Permettent la notification d'un changement de structure de l'arbre DOM.

```
// Introduced in DOM Level 2:
interface MutationEvent : Event {

  // attrChangeType
  const unsigned short      MODIFICATION                    = 1;
  const unsigned short      ADDITION                        = 2;
  const unsigned short      REMOVAL                         = 3;

  readonly attribute Node             relatedNode;
  readonly attribute DOMString        prevValue;
  readonly attribute DOMString        newValue;
  readonly attribute DOMString        attrName;
  readonly attribute unsigned short   attrChange;
  void               initMutationEvent(in DOMString typeArg,
      in boolean canBubbleArg,
      in boolean cancelableArg,
      in Node relatedNodeArg,
      in DOMString prevValueArg,
      in DOMString newValueArg,
      in DOMString attrNameArg,
      in unsigned short attrChangeArg);
};
```

Html Events

Evénements relatifs à l'interface graphique.

Nom et description	
`focus, blur`	un élément gagne ou perd le focus
`load`	chargement complet d'une page
`resize`	fenêtre du navigateur redimensionnée
`scroll`	page scrollée
`unload`	page déchargée
`abort`	chargement d'une page stoppée avant le chargement complet d'une image
`error`	erreur de chargement ou dans un script

Html Events

Evénements relatifs aux éléments de formulaire

Nom et description	
`change`	changement dans un élément de formulaire
`reset`	réinialisation du formulaire
`select`	selection de texte
`submit`	soumission d'un formulaire