Javascript : manipuler l'arbre DOM

Technologies du Web 1

Jean-Christophe Routier
Licence 1 SESI
Université Lille 1

UFR IEEA
Formations en
Informatique de
Lille 1

Sélection d'éléments

Pour manipuler les éléments de la page il faut au préalable les sélectionner.

La sélection d'éléments peut se faire

- par son attribut `id`
- par son attribut `class`
- par sa balise
- par un sélecteur CSS

\+ sélection par attribut `name` sur certains éléments

Sélection par l'identité

getElementById

la méthode **getElementById** de l'objet **document** sélectionne l'unique élément du document dont l'**id** est fourni en paramètre, ou **null** si aucun

- le résultat est un objet élément (de type HTMLElement)

```
var element = document.getElementById("joconde");
```

Sélection par la classe ou la balise

getElementsByClassName

la méthode `getElementsByClassName` sélectionne les éléments dont la classe est fournie en paramètre

getElementsByTagName

la méthode `getElementsByTagName` sélectionne les éléments dont la balise est fournie en paramètre

Ces 2 méthodes :

- peuvent s'appliquer au document ou à un élément du document, dans le second cas seuls les éléments descendants sont sélectionnés
- ont pour résultat la liste des éléments sélectionnés
  $\longrightarrow$ se manipule comme un tableau non mutable
  cette liste est dynamique

```
                                // tous les <div> du document
var divList = document.getElementsByTagName("div");
divList.length;                 // -> le nombre d'éléments sélectionnés
                                // toutes les <img> descendants de 'sec1'
var sec1 = document.getElementById("section1");
var sec1DivList = sec1.getElementsByTagName("img");
sec1DivList[0];                 // -> le premier élément sélectionné
```

```
                                // tous les éléments de classe 'remarque'
var classList = document.getElementByClassName("remarque");
                                // éléments descendant de 'unique' de classe
                                    'gauche' et 'encadre'
var elt = document.getElementById("unique");
var eltList = elt.getElementsByClassName("gauche encadre");
                                // les mêmes éléments
var eltList2 = elt.getElementsByClassName("encadre gauche");
```

Sélection par sélecteurs CSS

querySelectorAll

la méthode `querySelectorAll` sélectionne les éléments retenus par le sélecteur CSS fourni en paramètre
`querySelector` est similaire mais ne fournit que le premier élément

- peut s'appliquer à `document` ou à un élément
  dans le cas d'une invocation sur un élément `e`, le sélecteur est appliqué à tout le document et seuls les éléments descendants de `e` sont retenus
- la liste résultat n'est pas dynamique
- certaines pseudo-classes (`:link`, `:visited`) et pseudo-éléments (`::first-letter`, `::first-line`) ne sont pas acceptés

```
    // tous les éléments <img> du document emboités dan un
    //     élément <div> de classe 'exercice'
var listElement = document.querySelectorAll("div.exercice img");

    // le premier de ces éléments
var premier = document.querySelector("div.exercice img");

    // tous les liens ciblant un ".pdf" descendants
    //     de l'élément d'id 'exo1'
var elmt = document.getElementById("exo1");
var listElem = elemt.querySelectorAll('a[href$=".pdf"]');
```

Propriétés des éléments

les objets éléments possèdent des propriétés manipulables :

- attributs
- contenu
- style css

une fois un élément sélectionné, on peut agir sur ces propriétés

Manipuler les attributs

- les atttributs html sont des propriétés
  - même nom, en minuscules, avec « conversion camelback »
  - l'attribut `class` devient `className`
  - la valeur peut être `string`, `number` ou `boolean` selon attribut
- on peut également utiliser `getAttribute` et `setAttribute`
  - dans ce cas la valeur est **toujours** une chaîne de caractères

```
    // récupération de l'élément d'id 'celebre'
var monImage = document.getElementById("celebre");
    // accès à son attribut 'width' : un number
var taille = monImage.width;
    // modification de son attribut 'width'
monImage.width = taille + 100;
    // modification d'autres attributs
monImage.src = "images/joconde.jpg";
monImage.alt = "le tableau de la Joconde";
monImage.className = "gauche";
```

cf. *exemple_propriete.html*⊙

Manipuler le contenu

`innerHTML`

la propriété `innerHTML` représente le contenu HTML d'un élément

- lorsque la valeur de cette propriété est modifiée, son contenu est interprété par le navigateur

`textContent`

la propriété `textContent` représente le contenu textuel d'un élément

- lorsque cette propriété est lue, elle ne contient pas les balises HTML

```
var elemt = document.getElementById("exemple");
var htmlText = element.innerHTML;
alert(htmlText);   // -> "<p> Ceci est <strong>
                   //  mon</strong> contenu.</p>"

var txt = element.textContent;
alert(txt);        // -> "Ceci est mon contenu"
```

```
...
<div id="exemple">
   <p>Ceci est
       <strong>mon</strong>
      contenu.
   </p>
</div>
...
```

```
  // modifie le contenu HTML de l'élément et donc son 'affichage'.
  //   La balise <em> est interprétée.
elemt.innerHTML = "un <em>autre</em> contenu";

  // modifie le contenu texte de l'élément et donc son 'affichage.'
  // Le texte <strong> N'est PAS interprété.
elemt.textContent = "un contenu <strong>texte</strong>";
```

cf. *exemple_text-inner.html*⊙

Agir sur les propriétés CSS

- la propriété `style` d'un élément permet d'agir sur les propriétés CSS de cet élément
  mais elle **ne permet pas** d'accéder aux valeurs des propriétés définies dans une feuille de style, seulement aux propriétés définies dans le document HTML ou via `style`
- on utilise directement le nom de la propriété CSS après « conversion camelback » si nécessaire
  $\longrightarrow$ `font-size` $\rightsquigarrow$ `fontSize`, `border-right-style` $\rightsquigarrow$ `borderRightStyle`, etc.
- les valeurs sont toujours des chaînes de caractères
- les unités doivent être précisées

```
      // sélection de l'élément voulu
var elemt = document.getElementById("exemple");

      // modification de certaine propriétés CSS
      //    'l'affichage' est immédiatement impacté
elemt.style.fontWeight = "bold";
elemt.style.fontSize = "12px";          // ne pas oublier l'unité
elemt.style.marginRight = "10px";
elemt.style.marginTop = "2%";
elemt.style.backgroundColor = "rgba(128,0,0,0.5)";

var r = element.style.marginRight;   // /!\ 'string' avec les unités
var nouveauR = r + 100;              // -> "10px100" !!!
var valNouvR = parseInt(r)+100;      // -> 110
element.style.marginRight = valNouvR+"px";//ne pas oublier l'unité !
```

cf. *exemple_modification-style.html*⊙

Style « calculé »

`getComputedStyle`

la méthode `getComputedStyle` de l'objet `window` permet d'obtenir le valeurs des propriétés CSS appliquées par le navigateur

- les propriétés CSS ont le même nom que précédemment
  pas de « raccourci » autorisé : `margin` interdit, utiliser `marginLeft`, ...
- elles sont en **lecture seule**
- elles s'expriment en unité **absolue** (*px*)

```
    // sélection de l'élément voulu
var elemt = document.getElementById("exemple");

    // récupération du style calculé
var computed = window.getComputedStyle(elemt);
var marge = computed.marginLeft;            // avec les unités, en px
var couleur = computed.backgroundColor;     // rgb(...,...,...)

    // appliquer un facteur d'échelle de 10% à la largeur :
var largeur = parseInt(computed.width);  // récupère valeur calculée
var nvLargeur = Math.floor(largeur*1.10);// ajoute 10%
elemt.style.width = nvLargeur + "px";    // modifie style appliqué
```

Evénements

Certaines actions sur des éléments produisent un **événement**.
Il existe différents types d'événements, ils caractérisent l'action réalisée et dépendent de l'élément **cible** (sur lequel porte l'action).

- actions de l'utilisateur via le clavier ou la souris
  $\longrightarrow$ `click`, `keypress`, `keyup`, `mouseover`, etc.
- changement d'état
  $\longrightarrow$ `change`, `focus`
- chargement d'un élément
  $\longrightarrow$ `load`
- etc.

Programmation événementielle

programmation événementielle

La **programmation événementielle** consiste à lier une fonction à l'occurrence d'un événement sur un élément.
On parle d'**abonnement** de la fonction à l'élément pour l'événement.
La fonction est déclenchée lorsque l'événement se produit sur cet élément **cible** – ***target***.

fonction *listener*

La fonction attachée à un événement est appelée fonction « gestionnaire d'événement » – *event handler* – ou « d'écoute » – ***event listener***.

Méthode d'abonnement

addEventListener

La méthode `addEventListener` permet d'abonner à l'objet sur lequel elle est invoquée une fonction pour l'événement précisé.

objet.`addEventListener(`*eventType*`,` *listenerFunction*`)`

- *objet* : l'objet ciblé : `window`, ou un élément de la page
- *eventType* : une chaîne de caractères désignant le type d'événement `"click"`, `"load"`, `"change"`, `"mouseover"`, `"keypress"` etc.
- *listenerFunction* : la fonction *listener* qui sera appelée lorsque l'événement se produit

```
var action1 = function() {
    window.alert("on a cliqué sur le pied de page");
}

      // sélection d'un élément
var pied =  document.getElementById("piedDePage");
      // abonnement sur cet élément de la fonction 'action1'
      // pour un événement "click" => 'action()' déclenchée si clic
pied.addEventListener("click",action1);
```

cf. *exemple_event1.html*⊙

Un peu de méthodologie

1. placer les fonctions javascript dans un fichier à part de l'html
2. définir une fonction chargée de mettre en place les abonnements :
   1. récupérer l'élément ciblé
   2. abonner la fonction listener pour l'événement voulu
3. déclencher cette fonction
   - doit être fait lorsque la page est complètement chargée pour être sûr que tous les éléments existent
     ↪ utiliser l'événement `load` sur `window`

```
<html>
  <head> ...
     <script src="monScript.js"></script>
  </head>
  <body> ...
     <img id="laJoconde" ... /> ...
     <div id="piedDePage"> ... </div>
     ...
```

cf. *exemple_event2.html*⊙
cf. *temperature.html*⊙

avec monscript.js :

```
                        // fonction de mise en place des abonnements
var setupEvents = function() {
                        // abonnement pour élément d'id 'piedDePage'
  var pied = document.getElementById("piedDePage");
  pied.addEventListener("click",action1);
                        // abonnement pour élément d'id 'laJoconde'
  var joconde = document.getElementById("laJoconde");
  joconde.addEventListener("mouseover",action2);
}

              // on provoque l'exécution de "setupEvents" à la fin du
              //                  chargement du document
window.addEventListener("load", setupEvents);

              // définition des fonctions listener
var action1 = function() { ... }
var action2 = function() { ... }
```

Sur un élément donné on peut avoir

- plusieurs abonnements pour différents événements
- plusieurs abonnements pour le même événement

removeEventListener

la méthode `removeEventListener` permet de désabonner de l'objet sur lequel elle est invoquée une fonction pour un événement

objet.`removeEventListener(`*eventType*`,` *listenerFunction*`)`

cf. `exemple_event3.html`

Complément sur fonctions d'écoute

- dans une fonction *listener*, la variable `this` est définie et désigne l'objet qui a déclenché l'événement
  typiquement l'élément du document
- un objet `event` est créé pour chaque événement. Cet objet est passé en paramètre lors du déclenchement de la fonction listener associée.
  Le type d'objet `event` varie selon l'événement.
  Un objet `event` possède des propriétés qui informent sur l'événement.

L'objet `event`

Quelques propriétés (selon les types d'événements) :

- `clientX`, `clientY`/ `screenX`, `screenY`/ `pageX`, `pageY` coordonnées de l'événement par rapport au "navigateur"/l'écran/la page
- `altKey`, `ctrKey`, `shiftKey` l'une des touches Alt, Ctrl ou Shift était pressée lors de l'événement ?
- `keyCode` information sur la touche appuyée
  à combiner avec `String.fromCharCode()`
- `target` la cible de l'événement (== `this`)
- etc.

cf. *exemple_event4.html*⊙ – *exemple_target.html*⊙

Versions précédentes

- DOM niveau 0 :
  - les gestionnaires d'événements s'appellent `on`*event* : `onclick`, `onload`, `onmmouseover`, etc.
  - un gestionnaire d'événement peut être placé *en ligne* en tant qu'attribut d'un élément, la valeur associée est le code exécuté

```
<img src="..." onclick="var i = 2; action();"/>
```

  - on peut aussi créer l'abonnement dans le code javascript :

```
element.onclick = maFonction;
```

- défauts :
  - code javascript dans le code HTML
  - un seul abonnement possible par type d'événement

Conclusion

utiliser le modèle `addEventListener`

- `attachEvent(...)` dans IE < 8 ($+$ autres différences)

Le type Node

type Node

Les nœuds de l'arbre DOM sélectionnés par `getElementById`, `getElementsByClassName`, ..., sont de type `Node`.

Un objet `Node` propose les propriétés :

- `nodeName` : le nom du nœud (balise en majuscules)
- `nodeType` : type du nœud défini par des constantes nommées prédéfinies, `Node.ELEMENT_NODE` (= 1), `Node.TEXT_NODE` (= 3)
- `nodeValue` : `null` si le nœud est un élément, le contenu pour une nœud texte
- `parentNode` le nœud parent du nœud courant
- `childNodes` la liste (`NodeList`) des fils du nœud courant
- `fitstChild`, `lastChild` : le premier et dernier des nœuds fils
- `previouSibling`, `nextSibling` : le nœud frère précédent/suivant
- etc.

cf. `exemple_dom3.html`

Création

- `document.createElement(`*`balise`*`)` : crée un nouveau nœud avec la *balise* donnée
- `document.createTextNode(`*`texte`*`)` : crée un nouveau nœud texte avec le contenu *texte* fourni (non interprété)

NB : existe aussi

- *`node`*`.cloneNode` qui a pour résultat un nouveau nœud copie de `node`

Insertion

- *noeudParent*.**insertBefore**(*noeudInséré*,*noeudRéférence*) : insère *noeudInséré* avant *noeudRéférence* comme fils de *noeudParent*
- *parent*.**appendChild**(*noeudAjouté*) : le nœud *noeudAjouté* est ajouté à la fin des fils de *parent*

NB : si le nœud *inséré* ou *ajouté* existe dans le document, il est alors **déplacé** (donc supprimé de la position existante et inséré/ajouté à la position demandée).

cf. exemple_dom3.html⊙

Suppression et remplacement

- *parent*.**removeChild**(*noeud*) : *noeud* est supprimé des fils de *parent*
- *parent*.**replaceChild**(*remplaçant*,*remplacé*) : *remplaçant* prend la place de *remplacé* comme fils de *parent*

cf. exemple_dom3.html⊙

Fonction listener et paramètres

- le second paramètre de `addEventListener` est la *fonction d'écoute*, il s'agit d'une **valeur de type fonction** et **pas d'un appel de fonction** !

Problème

Comment faire pour que la *listener function* soit paramétrée ?

Par exemple :

- on dispose d'un document avec des images
- lors du clic sur une image on veut déclencher une action
- les actions sont similaires pour toutes les images mais dépendent du « numéro » de l'image

cf. *exemple_fonction_param.html*

Une solution ?

```
var setupListeners = function() {
    var elements = document.getElementsByTagName("img");

    elements[0].addEventListener("click",listener1);
    elements[1].addEventListener("click",listener2);
    elements[2].addEventListener("click",listener3);
    elements[3].addEventListener("click",listener4);
}
window.addEventListener("load", setupListeners);
// les fonctions "event listener" : ICI une par image...
var listener1 =  function() {
     var sourceDisplay = document.getElementById("source");
     sourceDisplay.innerHTML = "image 1";
};
var listener2 =  function() {
     var sourceDisplay = document.getElementById("source");
     sourceDisplay.innerHTML = "image 2";
};
var listener3 =  function() {  ... "image 3"
var listener4 =  function() {  ... "image 4"
```

cf. *exemple_fonction_param_mauvais.html*

Problème

et si on a plus de 4 images ?

- les `listener` diffèrent peu...
- on aurait pu résoudre le problème avec cette fonction

```
1 var listener =  function(numero) {
2     var sourceDisplay = document.getElementById("source");
3     sourceDisplay.innerHTML = "image "+numero;
4 };
```

- mais il y a un paramètre... on peut alors écrire cela

```
1 var numero = NNN  ;     // NNN = valeur à changer à chaque fois
2 var listener =  function() {
3     var sourceDisplay = document.getElementById("source");
4     sourceDisplay.innerHTML = "image "+numero;
5 };
```

mais il faut changer `NNN` à chaque fois...

Fonction qui calcule une fonction

- une fonction peut calculer une variable et la fournir en résultat

```
var exemple =  function() {
     var result = 10;
     return result*10+ 5;
};
```

- c'est aussi possible avec les variables dont **le résultat** est une fonction

```
var creeFonction =  function() {
     var func = function(i) {
         return 2*i;
     } ;
     return func;
}

var f = creeFonction(2);  // le résultat est une fonction
f;          // function() { return 2*i; }
f(12);      // -> 24
```

Fonction qui calcule une fonction (2)

- la fonction « créatrice » peut être paramétrée

```
1  var creeFonctionMutiple =  function(facteur) {
2       var func = function(i) {
3           return facteur*i;
4       } ;
5       return func;
6  }
7
8  fois2 = creeFonctionMultiple(2);
9  fois2(7);                                  // -> 14
10
11 fois5 = creeFonctionMultiple(5);
12 fois5(7);                                  // -> 35
```

Une fonction qui calcule les *listeners*

```
var createListener = function(numero) {
    var listener =  function() {
        var sourceDisplay = document.getElementById("source");
        sourceDisplay.innerHTML = "image "+numero;
    };
    return listener;
}
```

La solution

```
var setupListeners = function() {
    var elements = document.getElementsByTagName("img");
    for(var i=0; i < elements.length; i++) {
        elements[i].addEventListener("click",createListener(i+1));
    }
}

window.addEventListener("load", setupListeners);

var createListener = function(numero) {
    var listener =  function() {
        var sourceDisplay = document.getElementById("source");
        sourceDisplay.innerHTML = "image "+numero;
    };
    return listener;
}
```

Si plus de 4 images ? **Pas de problème !**

cf. *exemple_fonction_param.html*