JavaScript: DOM et Gestion des Événements

Quentin Bouillaguet <quentin.bouillaguet@u-psud.fr>

2019-2020

Document Object Model (DOM)

DOM pour les documents HTML

Gestion des événements

Événements globaux DOM

Programmer une action

Document Object Model (DOM)

Le *Document Object Model* (DOM)

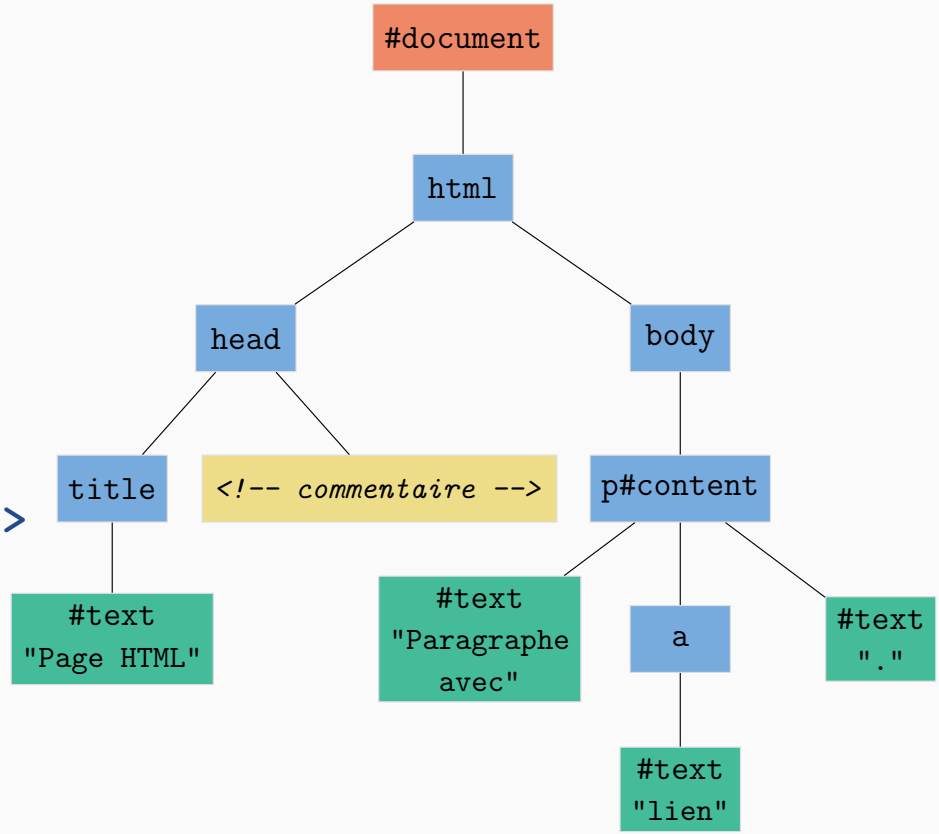
- **Interface de programmation** (API) pour les documents décrits dans un *langage de balisage* (HTML, XML, SVG, ...).
  - représentation structurée du document de manière **arborescente**
  - un **noeud** de cet arbre peut correspondre à un *élément* (délimité par sa balise de début et de fin), du *texte*, ...
  - **manipulation** de la **structure** du document par l'intermédiaire d'**objets** (méthodes et propriétés) représentant les noeuds
- Décrit par la spécification DOM du W3C/WHATWG
  - "*Living Standard*": standard en constante évolution
  - implémenté par les navigateurs modernes
  - manipulable en JavaScript

Exemple d'arborescence d'un document HTML

Document HTML

```
<html>
  <head>
    <title>Page HTML</title>
    <!-- commentaire -->
  </head>
  <body>
    <p id="content">Paragraphe
    avec <a href="#">lien</a>.</p>
  </body>
</html>
```

Représentation DOM



Nœuds (`Node`) – Principes

Différents types de nœuds:

- **document**: nœud racine de l'arbre (page Web, document `XML`, ...)
- **élément**: élément délimité par ses balises de début et de fin
- **texte**: bloc de texte sans balisage
- **commentaire**: bloc de texte commenté

L'interface `Node`

- Ensemble de méthodes et propriétés partagées par tous les nœuds

Propriétés

- `nodeType`: `ELEMENT_NODE`, `TEXT_NODE`, `DOCUMENT_NODE`, ...
- `nodeName`: `"#document"`, `"#text"`, `"img"` pour `<img/>`, ...
- `nodeValue`: renvoie/définit le contenu d'un nœud texte/commentaire

Exemple

```
var content = document.getElementById("content");
content.nodeType; // vaut Node.ELEMENT_NODE
content.nodeName; // vaut "p"
```

Nœuds (`Node`) – Parcours d'arbre

Propriétés (en lecture)

- `childNodes[]` – tableau des nœuds enfants
- `firstChild` – premier enfant du nœud
- `lastChild` – dernier enfant du nœud
- `parentNode` – nœud père
- `ownerDocument` – nœud document dont dépend le nœud

Exemple

```
var a = content.childNodes[1]; // noeud enfant <a>
var body = content.parentNode; // noeud père <body>
var document = content.ownerDocument;
a.firstChild.nodeValue; // vaut "lien"
a.lastChild.nodeValue; // vaut "lien"
```

Nœuds (`Node`) – Ajout/Suppression

Méthodes

- `appendChild(node)` – insère un nœud dans l'élément courant.
- `removeChild(node)` – retire un nœud des fils de l'élément courant (le nœud n'est pas détruit)
- `cloneNode(bool)`– clone le nœud courant (récursivement ou pas)

Exemple

- Création d'un élément `<div>` et ajout comme nœud enfant de l'élément `<body>` dans un document HTML:

```
var div = document.createElement('div');
body.appendChild(div);
```

Référence

- Voir la référence sur `Node`

Les nœuds Document (`Document`)

Un nœud de type *document* représente la racine de l'arborescence.

L'interface `Document`

- dérive de l'interface `Node`
- fournit des fonctionnalités globales au document: obtenir l'URL, créer de nouveaux éléments, rechercher des éléments, ...

Méthodes

- `createElement(nom)` – crée un élément avec le `nom` de balise.
- `getElementById(id)` – renvoie l'élément d'attribut `id` ayant la valeur spécifiée ou `null` si l'élément n'existe pas
- `getElementsbyTagName(nom)` – renvoie un tableau d'éléments du `nom` spécifié

Référence

- Voir la référence sur `Document`

Les nœuds Éléments (`Element`)

On nomme **éléments** des nœuds de type *élément*.

Un élément peut avoir un identifiant **unique** (ID) lui étant associé.

Document

- L'attribut `id` définit une valeur identifiant l'élément de manière **unique** dans le document:

```
<p id="content">...</p>
```

DOM

- L'identifiant peut être (re)définit de manière dynamique par la propriété `id` de l'objet élément.

```
var a = content.childNodes[1];
// définition de l'id de l'élément `a`
// "link" est un identifiant non utilisé dans le document
a.id = "link";
```

DOM pour les documents HTML

Introduction

- Une page `HTML` est formée d'une *entête* `<head>` et d'un *corps* `<body>`
- Éléments, attributs et valeurs d'attributs ont une **sémantique** (une signification) définie par la norme. Par exemple:
  - l'élement `<p>` réprésente un *paragraphe* et `<a>` un *hyperlien*
  - l'attribut `id` précise l'*identifiant* d'un élément et `href` la *cible* d'un lien (sous la forme d'une URL)
  - `<img>` décrit une *image*
  - ...
- **DOM HTML** étend le DOM avec les spécificités sémantiques du HTML

Le document HTML (`HTMLDocument`)

L'interface `HTMLDocument`

- dérive de l'interface `Document`

Propriétés

- `body`/`head` – raccourci vers l'élément `<body>`/`<head>` du document
- `title` – titre du document
- `cookie` – cookies associés au document

Variable globale `document`

- par commodité, la variable `document` représente la racine du document HTML **courant** en JavaScript:

```
var n = document.createElement('div');
var body = document.body.appendChild(n);
```

Référence

- Voir la référence sur `HTMLDocument`

Les éléments HTML (`HTMLElement`)

- Différents types de nœuds éléments de sémantiques différentes:
  - métadonnées: titre (`<title>`), `<link>`, `<meta>`, ...
  - divisions du contenu: titre (`<h1>`), sous-titre (`<h2>`), `<section>`, ...
  - blocs de texte: paragraphe (`<p>`), liste (`<ol>` et `<ul>`), `<div>`, ...
  - textes en ligne: hyperlien (`<a>`), saut de ligne (`<br>`), ...
  - médias: image (`<img>`), audio (`<audio>`), vidéo (`<video>`), ...
  - ...

Propriétés DOM

- Modification du style d'un élément avec `style`
- Taille d'un élément et sa position relative par rapport à la zone d'affichage (`offsetTop`, `offsetLeft`, ...)
- Modification du texte interne à cet élément (`innerHTML`)
- Différentes propriétés disponibles selon le type d'élément !

Exemple de création d'un élément image (<img>)

```
// Creation de l'objet image (encore invisible):
var img = document.createElement("img");
// NB on peut aussi écrire directement
var img = new Image();

// Definition des attributs et style:
img.src = "figure.png"; // spécifique à <img>
img.style.position = "absolute";
img.style.left = "10px";

// Ajout dans la page (visible maintenant):
document.body.appendChild(img);
```

Manipulation du style d'un élément

- Le style d'un élément déclaré **en ligne** (*inline*) avec l'attribut `style` est **prioritaire** sur les déclarations des feuilles de styles (CSS).

```
<img id="image" style="position: absolute; left: 15px;" />
```

- Le DOM permet d'**accéder** au style **inline** d'un élément et le **modifier** via la propriété `style`. Les propriétés de `style` correspondent aux règles CSS (où les valeurs sont des chaînes de caractères).

```
var img = document.getElementById("image");
img.style.position = "absolute";
img.style.left = "15px";
```

- **Attention**: L'accès au style **calculé** d'un élément (après application des règles définies dans le CSS) se fait en utilisant la méthode `window.getComputedStyle(element)`

Manipulation du style d'un élément – Positionnement

Propriétés de taille

- `position`
  - `absolute` – placement par rapport à la position (0, 0) du parent
  - `relative` – placement par rapport aux éléments frères
  - Placement vertical avec `top` et `bottom`, horizontal avec `left` et `right`

```
element.style.position = "absolute";
element.style.left = "40px";
element.style.top = "10px";
```

- Voir la documentation CSS sur la position

Unités et valeurs de distance

- Nombre suivi **directement** d'une unité (sans espace avant)
  - `em` (taille de la police de l'élément), `px` (pixel), ...
- **Attention**: ces valeurs ne se manipulent pas comme des nombres

```
element.style.left += 10; // vaut "40px10" et non pas "50px"
```

Manipulation du style d'un élément – Visibilité

Faire apparaitre/disparaitre un élément

- Propriété `visibility`

```
element.style.visibility = "visible";
element.style.visibility = "hidden";
```

Placer un élément devant un autre

- Propriété `zIndex` – valeur plus élevée placée au dessus

```
element.style.zIndex = 1;
element.style.zIndex = 2;
```

- **Attention**: le nom est différent en HTML/CSS (`z-index`)

Manipulation du style d'un élément – Autres propriétés

- Voir la référence des propriétés CSS pour une liste des propriétés accessibles par `style`.
- **Attention**: les noms peuvent toutefois différer des noms standards CSS (*camelCase* en JavaScript et *kebab-case* en HTML/CSS).
  - exemple: `zIndex` en DOM, `z-index` en CSS
- La propriété `style` a le même niveau de priorité que le style *inline* d'un élément et est donc la plus **prioritaire**.

La fenêtre courante (`Window`)

L'objet `window`

- Représente la fenêtre ou l'onglet courant du navigateur
- Nos "**variables globales**" sont en fait des **propriétés** de cet objet

Propriétés

- `document` – nœud document racine contenu dans la fenêtre
- `window/self` – auto référence

Méthodes

- `alert()/prompt()` – affiche ou lit une chaîne de caractères dans une boîte de dialogue (voir Cours "JavaScript: Introduction")
- `setInterval()/setTimeout()` — programme l'exécution de code
- `getComputedStyle(element)` — style calculé d'un élément

Référence

- Voir la référence sur `Window`

Taille et position relative d'un élément

Position relative au parent

- Décalage horizontal: `offsetLeft` et `offsetRight`
- Décalage vertical: `offsetTop` et `offsetBottom`

Position relative par rapport à la fenêtre

- Rectangle formé par l'élément: `getBoudingClientRect()`
  - décalage horizontal: `left` et `right`
  - décalage vertical: `top` et `bottom`

Gestion des événements

Événements

Définition

- Structure logicielle contenant des informations au sujet d'une action ou occurrence **asynchrone** d'intérêt
- Des **gestionnaires**/**écouteurs** (*handlers*/*listeners*) d'événements peuvent s'**abonner** à certains types d'événements
- Ces **gestionnaires**/**écouteurs** sont des **fonctions**, appelées à **chaque fois** que ces événements se **produisent**

Catégories d'événements

- **Entrées** utilisateur
  - clic ou déplacement de la souris, pression d'une touche, ...
- **Actions** dans le document
  - création/destruction d'éléments, fin de chargement de la page, ...
- **Programmés** par l'application
  - parfois utilisé pour la communication entre composants (*timer*, ...)

Événements JavaScript

- Certains objets JavaScript peuvent avoir des gestionnaires/écouteurs d'événements
- Les nœuds du DOM aussi !
  - Cela permet de manipuler des pages web dynamiquement en réponse à différentes interactions.

Intégration avec HTML

- Intégration avec HTML par les attributs du type ou par JavaScript

```
<button onclick="thank()">...</button>
```

- Préférable toutefois de ne pas mélanger JavaScript et HTML (et donc d'utiliser les gestionnaires et écouteurs d'événements)

Propriétés communes à tous les objets d'événements

- `type`: type de l'événement (`keypress`, `load`, ...)
- `target`: objet référençant la cible de l'événement

Exemple: retarder l'exécution et clic de la souris

Document HTML:

```
<html>
  <head> ... </head>
  <body>
    <button id="ici">Cliquer ici</button>
  </body>
</html>
```

Fichier JavaScript:

```
function thank() { ... }

window.onload = function () {
  let button = document.getElementById("ici");
  button.addEventListener('click', thank, false);
}
```

Gestionnaires (*handlers*) d'événements

Ajout

- Propriétés '`onXXX`' de l'objet correspondant aux événements
- Similaire aux attributs HTML d'événements

```
window.onload = function () { ... }
button.onclick = thank
```

Suppression

- Affectation d'une valeur `null` à la propriété

```
button.onclick = null;
```

Remarque

- On ne peut avoir qu'**un** *handler* par événement ciblant l'objet

Écouteurs (*listeners*) d'événements

Ajout

- `addEventListener(type, listener, capture)` ajoute un `listener` pour le `type` d'événement donné.
- Le booléen `capture` indique si l'écoute se fait pendant la phase de capture de l'événement (voir plus bas)

```
window.addEventListener('load', function () { ... }, false);
button.addEventListener('click', thank, false);
```

Suppression

- `removeEventListener(type, listener, capture)` enlève le *listener* enregistré avec ces mêmes paramètres.

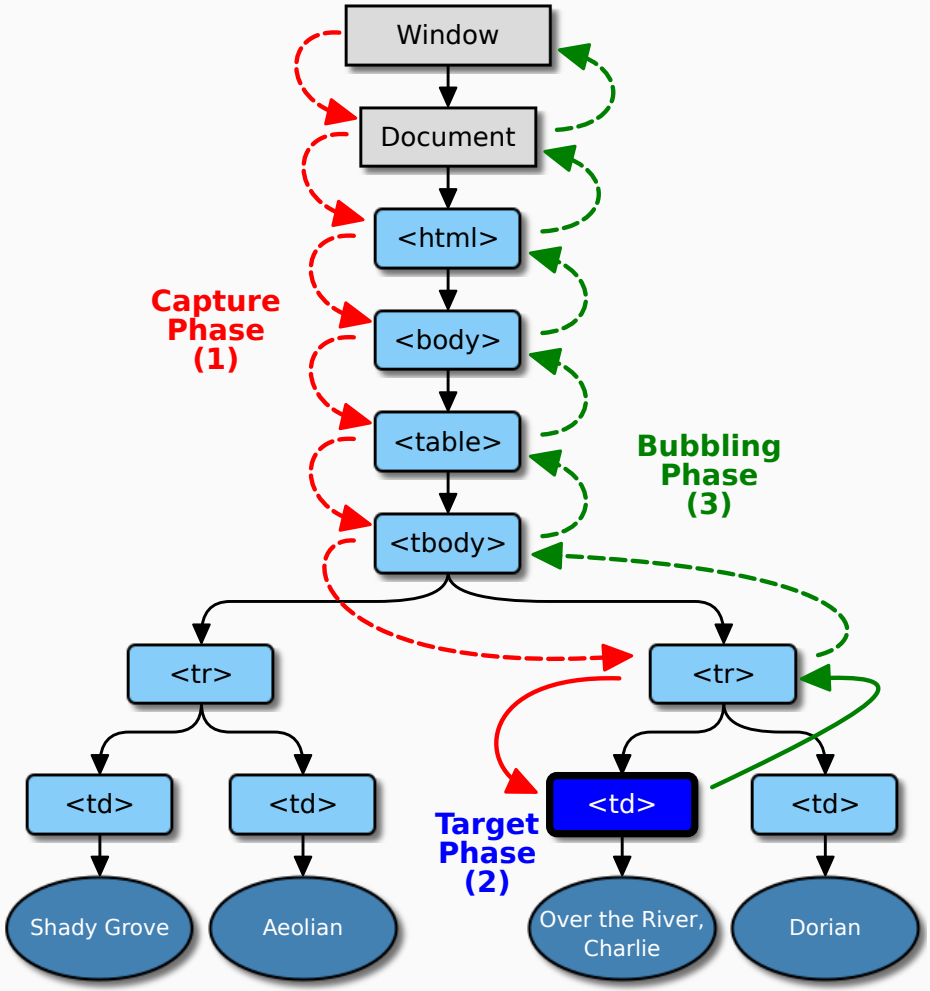
```
// Suppression du listener ajouté au dessus:
button.removeEventListener('click', thank, false);
```

Remarque

- **Plusieurs** *listeners* peuvent être associés par événement ciblant l'objet

Propagations des événéments

- Basé sur la hiérarchie du DOM
- Propagation en 2 phases:
  - Phase de **capture**:
    D'ancêtres en ancêtres de la fenêtre jusqu'à l'élément cible
  - Phase de **bouillonnement** (*bubbling*):
    En sens inverse du père de la cible jusqu'à la fenêtre



Événements globaux DOM

Aperçu

- Gestionnaires d'événéments usuels de nombreuses interfaces:
  - Éléments HTML (`HTMLElement`)
  - Document (`Document`)
  - Fenêtre (`Window`)
  - ...
- Chacune de ces interfaces peut implémenter plus de gestionnaires/d'écouteurs d'évènements.

Événements de chargement

Types d'événements

- `(on)load` – après chargement de l'élément, la page, d'une image, ...
- `(on)unload` – en quittant la page
- `(on)abort` – interruption du chargement
- `(on)error` – erreur de chargement (image ou objet)

Exemple

```
// Execution du code au chargement de la page
window.onload = function () {
  let img = document.getElementById("img");
  img.src = "figure.png";
  // Si la ressource ne se charge pas (inexistante, ...):
  img.onerror = function () { ... };
}
```

Événements souris

Types d'événements

- Bouton de souris
  - `(on)mousedown`: pression du bouton de la souris
  - `(on)mouseup`: relâchement du bouton de la souris
  - `(on)click`: pression et relâchement du bouton souris
  - `(on)dblclick`: double-clic de souris (interfère avec `onclick`)
- Déplacement de la souris
  - `(on)mouseover`: entrée du curseur dans l'élément
  - `(on)mouseout`: sortie du curseur de l'élément
  - `(on)mousemove`: déplacement de la souris (coûteux!)

Événements souris

Propriétés de l'événement

- `clientX/clientY`: position horizontale/verticale par rapport au coin haut-gauche de la fenêtre

Exemple

- Position par rapport à la fenêtre:

```
function follow(e) {
    alert("Clic en " + e.clientX + " " + e.clientY);
}
```

- Position par rapport à l'élément ciblé:

```
function position(e) {
  var rect = e.target.getBoudingClientRect();
  var x = e.clientX - rect.left;
  var y = e.clientY - rect.top;
  alert("Clic en " + x + " " + y);
}
```

Événements clavier – Principes

Types d'événements

- `(on)keydown`: enfoncement de la touche
- `(on)keypress`: envoi du caractère (à l'enfoncement puis à chaque répétition)
- `(on)keyup`: relâchement de la touche

Propriétés de l'événement

- `keyCode`: code de la touche (pour `keydown/up`)
- `charCode`: code du caractère (pour `keypress`)
- `altKey`, `ctrlKey`, `shiftKey`: modificateurs associés

Événements clavier – État des touches

- **Problème**: pas de mécanisme pour connaître l'état des touches !
- **Solution**: suivre et enregistrer les changements d'états !

Exemple

- Savoir à tout moment si une touche est enfoncée:

```
var touche_gauche = false;
//
function appuie(e) {
    if (e.keyCode == 37) touche_gauche = true;
}
window.addEventListener('keydown', appuie, false);
//
function relache(e) {
    if (e.keyCode == 37) touche_gauche = false;
}
window.addEventListener('keyup', relache, false);
```

Programmer une action

Programmer une action

Mécanismes en concurrence avec la boucle d'événements (asynchrone)

Enregistrement

- `setTimeout(fonction, délai)` – enregistre une `fonction` à appeler **une fois** sur la **file des événements** après un `délai` ≥ 0
- `setInterval(fonction, intervalle)` – semblable mais appels **répétés** à la fonction toute les `intervalle` au minimum.
- Le `délai` est précisé en **millisecondes**

Interruption

- L'enregistrement d'une action à programmer retourne une valeur numérique unique identifiant `timer` la programmation
- `clearTimeout(timer)/clearInterval(timer)`

Pieges

- La fonction connaîtra les variables globales au moment de l'*exécution*

Boucle d'événements, pile d'exécution et setTimeout()

- La boucle d'événements exécute en priorité les fonctions sur la **pile d'exécution** (FILO)
- Si **la pile est vide**, alors la première fonction de rappel sur la **queue d'événements** (FIFO) est mise sur la pile d'exécution
- La fonction `setTimeout()` ajoutera la fonction en argument sur la queue des événements après le délai spécifié.
  - Même avec une valeur de délai nul (0), ce mécanisme aura pour effet de retarder l'exécution de la fonction