Comprendre HTML5

Benjamin Canou, OCaml PRO

16 & 23 février 2016

Développement d'Applications Réticulaires

HTML5 ?

HTML5 ?

Derrière cet acronyme se cache :

- une proposition du WHATWG pour faire avancer le W3C plus vite,
- maintenant adoptée par le W3C.

En fait, on y associe toutes les technologies Web client modernes :

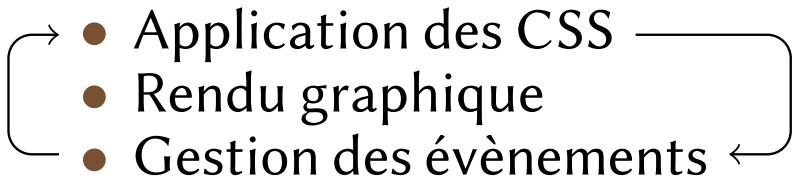
- la version 5 d'HTML,
- les versions récentes d'ECMAScript (ES6 en cours),
- les nouvelles APIs JavaScript du navigateur,
- les nouveaux attributs CSS 3.

Buts de ces premiers cours :

- Rappels des bases de la programmation client.
  N'hésitez pas à poser vos questions ou demander une démo !
- Étudier les principales nouveautés.

Rappels

Modèle d'exécution : la boucle d'évènements



- Pas de mise à jour de l'affichage pendant la gestion d'évènements.
- Impossible de toucher à la queue d'évènements de l'instant courant.
- Exécution bloquée durant la gestion d'évènements.
- (Don't) try `javascript:setTimeout(function(){while(true){}})` .

Gestion d'évènements :

- Prendre en charge un évènement = lui affecter une fermeture JavaScript
- Mécanisme particulier d'inhibition/propagation d'évènements (le bullage).
- Pour déclencher un évènement : setTimeout (pour le tour prochain).

Lorsque le navigateur rencontre un élément `script` :

- il stoppe l'analyse syntaxique,
- lance le téléchargement du script,
- met à jour le DOM avec la partie déjà analysée,
- interprète le script dès son arrivée,
- reprend l'analyse syntaxique.

L'affichage est bloqué pendant le téléchargement.

- Le navigateur peut télécharger en parallèle plusieurs scripts consécutifs.
- Il les exécute dans l'ordre une fois chargés.
- Les CSS sont traitées de la même façon.

À retenir :

- Les calculs longs bloquent le chargement de la page.
- Seul le DOM de la partie avant le script est disponible.

La boucle peut même démarrer avant le chargement complet.

- Il ne suffit pas d'utiliser `setTimeout` pour être sûr que le DOM est prêt.
- Astuce classique : utiliser les événements `onload` ;
- ou insérer les scripts à la fin du document.

Nouveautés HTML5 :

- `async` , le script est exécuté dès qu'il est téléchargé
- `defer` , le script est exécuté une fois le document complet

Lorsqu'on modifie le document depuis JavaScript :

- la structure DOM est modifié instantanément,
- si la modification est illégale,
  - soit une exception DOM est levée,
  - soit une modification alternative est appliquée
    (ex : insérer un nœud dans un `pre` , affecter un littéral de style),
- et c'est tout !

Au début du prochain tour de boucle,

- l'application des règles CSS est recalculée,
- la mise en page est recalculée,
- l'affichage est mis à jour,
- les nouvelles tailles sont reflétées dans le DOM.

Comment observer un changement de style ?

- on coupe le code en deux parties (avant / après la modification),
- on insère une pause avec `setTimeout` ou `requestAnimationFrame`.

Aie aie aie, ça clignotte !

Meilleure solution : éviter ce type de design autant que possible.

Les CSS modernes permettent

- d'éviter beaucoup de code de calcul d'interface,
- de rendre fluide les transitions lorsque ce code est inévitable.

HTML

Une longue histoire :

- À l'origine, SGML et HTML.
- Découplage structure / grammaire avec XML et DTD.
- Variante XML d'HTML : XHTML.
- Plusieurs versions d'XHTML, et variantes strict et transitional.
- Support variable des navigateurs.

La version 5 est un effort d'uniformisation, centrée sur les implantations.

Finalement, HTML5 c'est :

- Deux variantes : une rétro-compatible SGML et une XML.
- Grammaire moins exigeante.
- Spécification de l'imbrication "clarifiée" :
  - Avant : inline et block comme en CSS.
  - HTML5 : metadata, flow, sectioning, heading, phrasing, embedded, interactive.
- Spécification complète de l'interprétation des documents mal formés.
- Quelques éléments modifiés / ajoutés / supprimés.
- Attributs personnalisés : `data-xxxxxx` .

Les éléments de structure ajoutés :

- Buts : indexation, accessibilité.
- `main`, `aside`, `header`, `footer`, `nav` : éléments principaux de la page.
- `section` (inclut un `h1` … `h6`), `article`, `figure` (inclut un `figcaption`).
- Identification de contenu : `mark`, `output`, `ruby`, `bdi`, etc.
- Éléments d'interface : `progress`, `meter`, `time`
- Nouveaux types de champs de formulaire : `<input type='email'>`, etc.
- Contenu non-HTML inclus : `math`, `svg`, `audio`, `video`, `canvas`

Un petit exemple

```
1 : <!DOCTYPE html>
2 : <html>
3 :   <head>
4 :     <meta charset='UTF-8'>
5 :     <title>Bienvenue sur presse-le-bouton.com</title>
6 :     <style>
7 :       body { text-align : center ; }
8 :     </style>
9 :   </head>
10 :  <body>
11 :    <button>PRESSE MOI</button>
12 :  </body>
13 : </html>
```

Écrit en variante ”polyglotte” : compatible SGML et XML.

CSS

Syntaxe de base : `selecteur { attribut: valeur; }`.

- Sélecteur : expression ciblant un élément de la page.
- Attribut : parmi une liste bien définie.
- Valeur : syntaxe dépendante de l'attribut.

Exemple : `body { color: red; }`.

Commentaires : `/* ... */`

Styles inclus dans une page :

- Par élément : `<body style="color: red"> ... </body>`.
- Feulle incluse : `<style> /* <!-- */ ... /* --> */ </style>`.

On peut factoriser un sélecteur :

- `body { color: red; background: pink; }`

qui est strictement équivalent à :

- ```
  body { color: red; }
  body { background: pink; }
  ```

On peut aussi factoriser les attributs :

- `div,span { color: red; }`

qui est strictement équivalent à :

- ```
  div { color: red; }
  span { color: red; }
  ```

On peut factoriser certains attributs liés avec des macro attributs :

- `body { border: 3px solid yellow; }`

qui est trictement équivalent à :

- `body { border-color: yellow; }`
  `body { border-width: 3px; }`
  `body { border-style: solid; }`

mais attention, en réalité il s'agit de :

- `body { border-top-color: yellow; }`
  `body { border-right-color: yellow; }`
  `body { border-bottom-color: yellow; }`
  `body { border-left-color: yellow; }`
  `body { border-top-width: 3px; }`
  `...`

Certains macro-attributs `-vendor-xxxx` définissent une variante propriétaire.

- `-moz-xxxx` pour Gecko, Servo (Firefox)
- `-webkit-xxxx` pour Webkit (Chrome, Safari)
- `-ms-xxxx` pour les navigateurs Microsoft

En général, il s'agit d'introduire un brouillon de spécification.
Une fois la spécification implantée, les attributs préfixé et définitif sont des alias.

On se retrouve souvent avec un motif comme suit :

```
1 : -webkit-opacity : 0.5;
2 : -moz-opacity : 0.5;
3 : opacity : 0.5;
```

Des préprocesseurs existent pour faire ce travail automatiquement (SASS, LESS).

Sélecteurs de base :

- `xxx` cible les éléments `<xxx>...</xxx>`.
- `#xxx` cible les éléments avec l'attribut `id="xxx"`.
- `.xxx` cible les éléments avec l'attribut `class="... xxx ..."`.

Attention, on ne sélection que les éléments, pas les nœuds texte.
Pour personnaliser une partie de texte, on peut la mettre dans un `<span>`.

Pour sélectionner un élément répondant à plusieurs critères,
coller simplement plusieurs sélecteurs sans espace.

Par exemple,

- le sélecteur `div#saucisse.com`
- sélectionne l'élément `<div id="saucisse" class="com"></div>`.

On peut inverser un sélecteur avec `:not(selecteur)`.
Exemple : `:not(div)` sélectionne tous les éléments sauf les divs.

Deux sélecteurs séparés par une espace `sel1 sel2` signifient que :

- l'élément cible doit être sélectionné par `sel2`,
- un de ses ancêtre doit être sélectionné par `sel1`.

En réalité, on peut décrire un chemin complet dans l'arbre :

- par exemple `div#main h1 a { color: red; }`
- colore en rouge sélection les liens dans les titres du div à l'id `main`.

Opérateurs de chemin :

- `sel1 sel2` : un ancêtre de l'élément doit vérifier `sel1`
- `sel1>sel2` : le parent de l'élément doit vérifier `sel1`
- `sel1+sel2` : le frère juste à gauche de l'élément doit vérifier `sel1`
- `sel1~sel2` : un frère à gauche de l'élément doit vérifier `sel1`

où l'élément cible doit vérifier `sel2`, qui n'est pas un opérateur de chemin.

À noter :

- Les chemins ne peuvent que descendre ou avancer.
- On sélectionne toujours le dernier élément de chemin.

Il est possible de sélectionner selon les attributs de lélément :

- `[attr]` : l'élément possède un attribut `attr`
- `[attr=val]` : l'attribut `attr` vaut `val`
- `[attr^=val]` : l'attribut `attr` commence par `val`
- `[attr$=val]` : l'attribut `attr` finit par `val`
- `[attr*=val]` : l'attribut `attr` contient `val`
- `[attr~=val]` : l'attribut `attr` contient le mot `val`
- `[attr|=val]` : l'attribut `attr` est `val` ou commence par `val-`

Certains sélecteurs sont appelés pseudo classes.
Ils permettent de tester l'état de l'élément (et non sa structure).

Sélecteurs spécifiques aux liens :

- `:link`, un lien non visité
- `:visited`, un lien visité

Sélecteurs spécifiques aux états de l'interface :

- `:active`, un élément activé (bouton pressé, lien cliqué)
- `:hover`, un élément survolé
- `:focus`, l'élément ayant le focus clavier
- Formulaires : `:enabled`, `:disabled`, `:checked`, etc.

Le sélecteur `:target` sélectionne l'élément cible de l'URL du document.

Par exemple, voir mette en valeur la cible d'un lien une fois cliqué :

- On identifie la partie cible `<div id="sec4">...</div>`.
- On insère un lien vers cette ancre `Voir <a href="#sec4">section 4</a>.`
- On met en valeur la cible `div:target { background: yellow; }`.

Sélecteurs de position :

- `:first-child`, lélément est le premier fils de son parent
- `:nth-child(an+b)`, $\exists n$. l'élément est le $a \times n^{\text{ième}} + b$ fils de son parent
- `:only-child`, `:last-child`, `:nth-last-child(n)` `:only-of-type`, `:first-of-type`, `:last-of-type`, `:nth-of-type(n)`, `:nth-last-of-type(n)`

Attention : ne pas confondre avec les pseudo éléments :

- `::first-letter` la première lettre du contenu texte d'un élement
- `::first-line` la première ligne du contenu texte d'un élement
- `::before` un pseudo élément vide avant le texte d'un élément
- `::after` un pseudo élément vide après le texte d'un élément

Ces sélecteurs créent un élément virtuel dans l'arbre !
On peut modifier leur contenu avec l'attribut `content` .

Exemple : `button::before {content: "["} button::after {content: "]"}`
Ajoute des crochets autour du texte des boutons.

Pour chaque (élément, attribut), on regarde les sélecteurs qui s'appliquent.

Puis on jour au jeu suivant.

- 1 `#id` donne 100 point !
- 1 `.classe`, `:pseudoclasse` ou `[attribut]` donne 10 point !
- 1 `element` ou `::pseudoelement` donne 1 point !

La règle avec la meilleure somme gagne.

En cas d'égalité, la dernière règle s'applique.
L'attribut `style` gagne sur les sélecteurs.

Le mot clef `!important` :

- donne la plus haute priorité à une règle, même sur l'attribut `style` ;
- en cas de conflit, la dernière règle s'applique ;
- ne devrait être utilisé que très rarement.

Et si aucune règle ne s'applique ?

- Une valeur par défaut (initiale), définie dans une table, est appliquée.
- Si cet attribut est héritable, l'attribut de l'élément parent est préféré.

On peut définir l'héritage manuellement :

- Affecter `inherit` à un attribut force l'héritage.
- Affecter `initial` casse l'héritage et donne la valeur par défaut.
- Affecter `unset` à un attribut héritable équivaut à `inherit`.
- Affecter `unset` à un attribut non héritable équivaut à `initial`.

Sémantique des attributs CSS

Le box-model

`content` (de taille `width` / `height`)

`padding` (affecté par `background`)

`border`

`outline` (par dessus la `margin`)

`margin` (parfois recouvertes par celles d'autres éléments)

Marges :

- `margin: top right bottom left` , valeurs négatives autorisées !
- `padding: top right bottom left`

Fond :

- `background-image: image, ...`
- `background-color: couleur, ...`
- `background-position: x y`
- `background-size: width height`
- `background-repeat: no-repeat|repeat|repeat-x|repeat-y`

Texte :

- `font-family: serif|sans-serif|monospace` ou un nom specifique
- `font-style: normal|italic|oblique`
- `font-weight: normal|bold|bolder|lighter`
- `font-size: size`
- `line-height: height`

Fonte personnalisée :

```
1 : @font-face {
2 :     font-family : mafonte ;
3 :     src : url(ma_fonte.woff);
4 : }
```

Plusieurs fournisseurs de fontes sur le Web (ex : google fonts).

Bordures (idem pour outline) :

- `border-left-width: width`
- `border-left-style: solid|dotted|dashed|none`
- `border-left-color: color`
- Raccourci `border: width color style`
- Raccourci `border-left: width color style`
- Raccourci `border-width|style|color: top right bottom left`
- `border-top-left-radius: hradius (/ vradius)`
- Raccourci `border-radius: topleft topright bottomright bottomleft`

Les attributs comsmétiques

Les couleurs :

- couleurs nommées
- `#RGB` , `#RRGGBB`
- `rgb(0..255, 0..255, 0..255)` , `rgba(0..255, 0..255, 0..255, 0..1)`

Les images :

- `url('url')` , y compris `url('...')`
- `linear-gradient(0..360deg|to top..., color pos, color pos, ...)`
- `radial-gradient(...)` attention, kitsch assuré !

Les longueurs :

- `px` : pixels virtuels (à 200%, un `px` prend 4 vrais pixels)
- `ex` : hauteur d'un x
- `em` : largeur d'un m

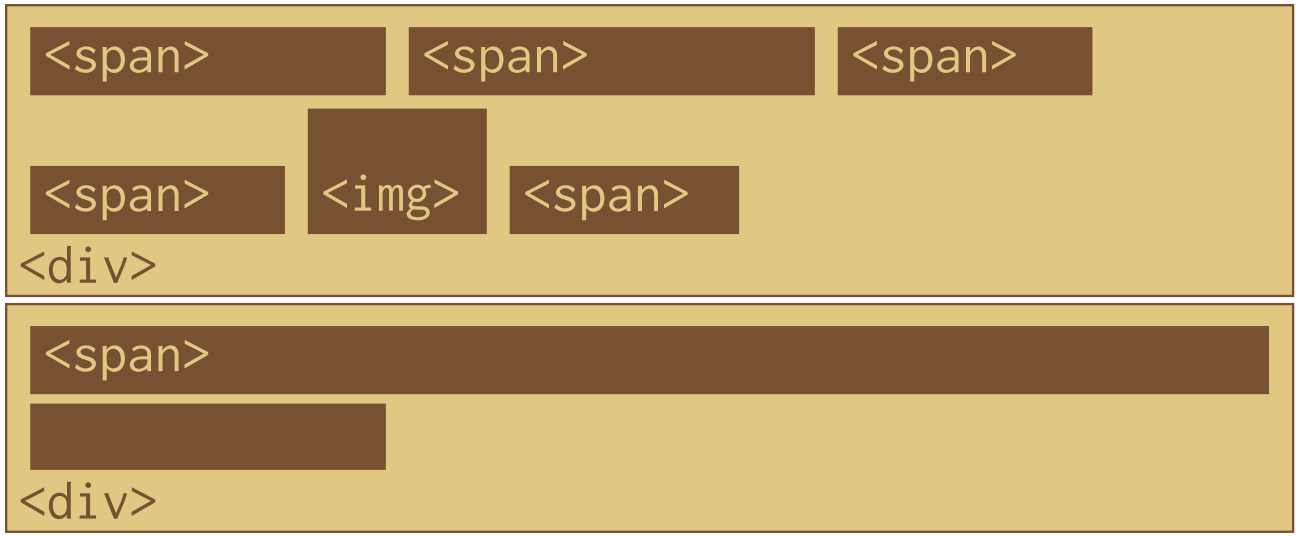
Les effets :

- `transform: scale(0..1)|scaleX(0..1)|scaleY(0..)|rotateX(ndeg)`
- `filter: blur(radius) | contrast(pct)| ...` (effets SVG)
- `background-blend-mode`
  `blend-mode: normal|multiply|screen|overlay|...`
- `text-shadow: x y radius color`
- `box-shadow: x y spread radius color`

L'attribut display

Deux modes historiques :

- `display: block`, blocs principaux alignés verticalement,
  exemple d'élément ayant l'attribut `display` à `block` par défaut : `<div>`.
- `display: inline`, contenu des blocs, aligné horizontalement avec césure
  exemple d'élément ayant l'attribut `display` à `inline` par défaut : `<span>`



L'interprétation de certains attributs dépend du mode de `display` :

- `text-align` (`left` , `justify` , etc.) n'a de sens qu'en mode `block` .
- `vertical-align` (`top` , `baseline` , etc.) n'a de sens qu'en mode `inline` (mais on peut le définir sur le bloc parent et utiliser l'héritage).

Modes spéciaux :

- `inline-block` : permet d'inclure un bloc dans une ligne.
- `none` : l'elément n'apparait pas du tout (différent de `visibility: hidden` ou `opacity: 0`).
- `run-in` : choisit selon le contexte (par exemple `block` dans un `div` mais `inline` dans un `p`).
- Simulation de tableaux et listes : `list-item` , `table` , `table-cell` , `table-row` , etc.
- `inline-table` : permet d'intégrer un tableau au milieu d'une ligne.

display : flex

Modèle reprenant la mise en page d'interfaces graphiques classiques.

- Équivalent des boîtes et des glues des toolkits classiques.
- Un bloc dont le contenu est une liste de blocs répartis de façon flexible.

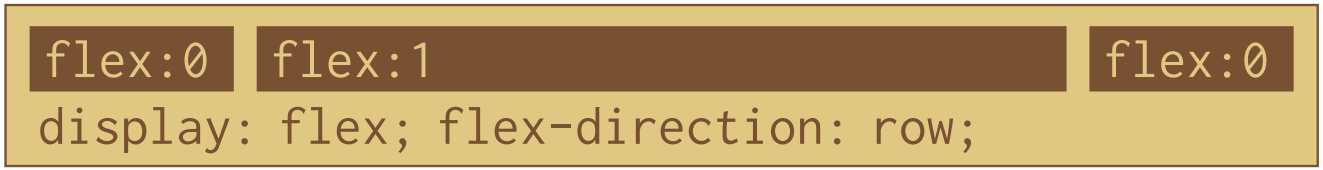
Sur le parent (la boîte) :

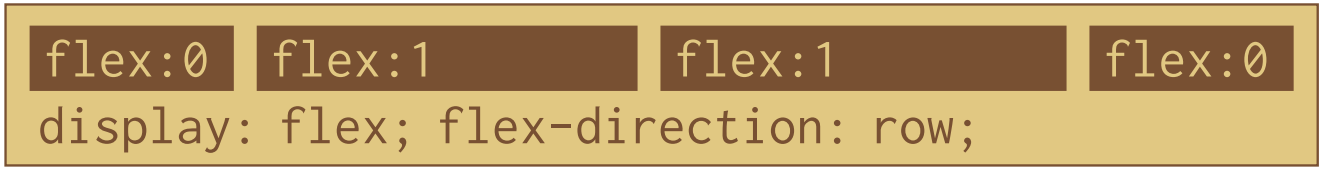
- `display: flex` ou `display: inline-flex`
- (opt) `flex-direction: row` ou `column`
- (opt) `align-items: strech|center|baseline|flex-start|flex-end`

Sur les enfants (les éléments) :

- `flex: 0` , ne prendre que l'espace nécessaire
- `flex: n` , prendre l'espace disponible avec un poids de `n`
- (opt) `flex-shrink: n` définir le poids lorsqu'il n'y a pas assez de place
- (opt) `flex-grow: n` définir le poids lorsqu'il y a trop de place
- (opt) `flex-basis: n` définir la taille nominale
- (opt) `order: n` pour réordonner les éléments

Quelques exemples :





L'attribut position

Trois cas :

- `position: static` par défaut
- `position: relative`
- `position: absolute`

Dans les deux cas `position: relative|absolute`, cet attribut :

- permet de positionner les bords l'élément avec `top`, `left`, `right` et `bottom` ;
- de le redimensionner avec `width` et `height` ;
- de contraindre ses descendants dans le rectangle englobant l'élément ;
- accessoirement, ouvre une portée locale de `z-index`.

Il est impossible de dissocier ces trois comportements.

Cet attribut influence les attributs ainsi :

- `width:` largeur de l'élément en pourcentage du rectangle englobant
- `left:` distance du bord gauche au bord gauche du rectangle englobant
- etc.

En `position: absolute`,

- le rectangle englobant pour l'élément lui-même est celui de son premier parent en `position: relative|absolute` (ou la fenêtre) ;
- celui de ses descendants est le rectangle une fois le positionnement appliqué.

En `position: relative`,

- le rectangle englobant pour l'élément lui-même est celui originellement calculé par le navigateur ;
- celui de ses descendants est le rectangle une fois le positionnement appliqué.

Transitions

Façon facile d'animer les changements de style.

On anime avec les attributs suivants.

- `transition-property': opacity|width|...`
- `transition-duration: 44s`
- `transition-timing-function: linear|ease-in|ease-out`
- `transition-delay: 13s`

Attention,

- tous les attributs ne sont pas animables ;
- toutes les valeurs ne sont pas animables (ex. seules les tailles absolues).

Animations

Façon plus complète mais plus complexe.

On anime un élément avec les attributs suivants.

- `animation-name: monAnimation`
- `animation-duration: 42s`
- `animation-timing-function: linear|ease-in|ease-out`
- `animation-delay: 13s`
- `animation-iteration-count: n`
- `animation-direction: normal|reverse|alternate`

Exemple de définition

```
1 : @keyframes monAnimation {
2 :   0% { width : 50px }
3 :   0% { width : 150px }
4 : }
```

Raccourci : `animation: monAnimation 1s alternate ease-in-out`

Media queries

Permet d'adapter l'affichage au support :

```
1 : <style>
2 : @media (min-width : 400px) {
3 :   #main { display : flex ; flex-direction : row ; }
4 :   #main .menu { flex : 0 0 400px ; order : 2 ; }
5 :   #main .content { flex : 1 ; order : 1 ; }
6 : }
7 : </style>
8 : <div id="main">
9 :   <div class="menu">...</div>
10 :  <div class="content">...</div>
11 : </div>
```

CSS pour SVG

```
1 : <!DOCTYPE html>
2 : <html>
3 :   <head>
4 :     <meta charset='UTF-8'>
5 :     <title>Bienvenue sur presse-le-bouton.com</title>
6 :     <style>
7 :       body { text-align : center ; }
8 :       #bigbutton { fill :red ; transition : fill 1s ; }
9 :       button :hover #bigbutton { fill : pink ; }
10 :    </style>
11 :  </head>
12 :  <body>
13 :    <button>
14 :      <svg width="60" height="60">
15 :        <circle id="bigbutton" cx="30" cy="30" r="28"/>
16 :      </svg>
17 :    </button>
18 :  </body>
19 : </html>
```

Quelques API Modernes

API ajoutant des fonctionnalités

Principalement du multimédia :

- Canvas, dessin 2D
- WebGL, dessin 3D
- Web Audio, API de synthétiseur numérique
- GamePad, gestion de périphériques d'entŕee
- WebRTC, visioconférence
- Géolocalisation

API améliorant le langage

Standard ou en voie de standardisation :

- Workers
- Local Storage
- Weak References
- Typed Arrays
- Promises

Et d'autres en état d'expérimentation.

Query Selectors

Recherche déléments dans le DOM

- même syntaxe que les sélecteurs CSS ;
- fonctionnement similaire à jQuery.

Deux fonctions :

- `document.querySelector("selector")` renvoie un élément
- `document.querySelectorAll("selector")` renvoie tous les éléments

Exemple :

```
1 : for (var elt of document.querySelectorAll("span")) {
2 :     elt.style.border = '13px red dotted';
3 : }
```

Des threads dans le navigateur !

- On lance un script dams un espace séparé.
- Communication par passage de messages uniquement.
- Échanges asynchrones uniquement.

Petit exemple : un additionneur

- Un fichier `add.html` pour l'interface
- Un fichier `add_worker.js` pour l'additionneur
- Un fichier `add_main.js` pour l'appeler

Fichier add.html :

```
1 : <!doctype html>
2 : <html>
3 :   <head>
4 :     <title>add</title>
5 :     <script src="add_main.js" defer></script>
6 :     <meta charset="utf-8" />
7 :   </head>
8 :   <body>
9 :     <input type="number" id="x" value="0"> +
10 :    <input type="number" id="y" value="0"> =
11 :    <input type="number" id="r" value="0" readonly>
12 :  </body>
13 : </html>
```

Fichier add_main.js :

```
1 : var adder = new Worker("add_worker.js");
2 : var x = document.querySelector("#x");
3 : var y = document.querySelector("#y");
4 : adder.onmessage = function (e) {
5 :     document.querySelector("#r").value = e.data[0];
6 : }
7 : x.onchange = y.onchange = function () {
8 :     adder.postMessage ([ x.value, y.value ]);
9 : }
```

Fichier add_worker.js :

```
1 : onmessage = function (e) {
2 :     postMessage ([ parseInt(e.data[0])
3 :                  + parseInt(e.data[1]) ])
4 : }
```

Options avancées :

- Transfert d'objets au lieu de copie
- Terminaison interne (`close`) ou externe (`terminate`)

Des variantes :

- `SharedWorker` communication entre fenêtres
- `ServiceWorker` interception des XHR
- `AudioWorkerNode` génération de son en tâche de fond

Stockage (clé,valeur) dans le navigateur pour chaque domaine.

- `localStorage.length` , nombre de (clés,valeurs)
- `localStorage.key(n)` , nom de la clef `n`
- `localStorage.getItem("key")`
- `localStorage.setItem("key", "value")`
- `localStorage.removeItem("key")`
- `localStorage.clear()`

Et c'est tout !

On améliore notre additionneur :

```
1 : var adder = new Worker("add_worker.js");
2 : var x = document.querySelector("#x");
3 : var y = document.querySelector("#y");
4 : adder.onmessage = function (e) {
5 :     document.querySelector("#r").value = e.data[0];
6 : }
7 : x.onchange = y.onchange = function () {
8 :     localStorage.setItem("x", x.value)
9 :     localStorage.setItem("y", y.value)
10 :    adder.postMessage ([ x.value, y.value ]);
11 : }
12 : x.value = localStorage.getItem("x") || "0";
13 : y.value = localStorage.getItem("y") || "0";
14 : adder.postMessage ([ x.value, y.value ]);
```

Typed arrays

Api de gestion de grands tableaux :

- Tableaux d'entiers de taille variable.
- Vues d'un même tableau sous différents types.

Utilisé :

- pour représenter le contenu des images (canvas) ;
- pour manipuler des fichiers binaires ;
- pour écrire du code numérique avec `asm.js` ;
- par Qemu-JavaScript pour simuler mémoire ;
- par Emscripten pour simuler la gestion mémoire de C.

Weak Sets / Weak Maps

`var set = new WeakSet()` :

- ensemble d'objets faiblement référencés ;
- un objet disparait lorsqu'il next plus pointé que par `set` ;
- `for (var elt of set)` , itère sur les objets encore vivants ;
- `set.add(obj)` ;
- `set.delete(obj)` ;
- `set.has(obj)` .

`WeakMap` identique mais avec une valeur attachée à l'objet.

Exemple : utilisation dans un toolkit d'interface :

- on ajoute les composants à un WeakSet à la création ;
- les références sont cassées quand les composants sont supprimés ;
- on a un moyen d'itérer sur les éléments encore affichés.

Qu'est-ce qu'une promesse ?

- C'est un objet JavaScript (de prototype `Promise`),
- qui représente la terminaison d'une tâche,
- après laquelle on peut brancher une continuation,
- une tâche peut se terminer avec une valeur, passée à la continuation.

Une promesse peut être :

- en attente (la tâche est encore en cours),
- acquittée : tenue ou rompue.

Création d'une promesse :

- par certaines nouvelles API,
- pour le code utilisateur avec le constructeur `Promise`.

```
1 : var p = new Promise (function (resolve, reject) {
2 :   resolve (result);
3 :     // acquitte la promesse comme tenue
4 :   reject (Error (message));
5 :     // acquitte la promesse comme rompue
6 : });
7 : // initialement, p est en attente
```

Utilisation d'une promesse :

- la promesse a une méthode `then` ;
- premier argument : fonction de rappel de résultat ;
- second argument : fonction de rappel d'échec.

```
1 : p.then(function (result) {
2 :   // result est la valeur passée à resolve
3 : }, function (err) {
4 :   // result est la valeur passée à reject
5 : })
```

Exemple : version promesse de `getElementById` :

```
1 : <script language="JavaScript">
2 :  function elementById(id) {
3 :     var p = new Promise(function (resolve, reject) {
4 :      window.addEventListener("load", function () {
5 :        var elt = document.getElementById(id);
6 :        if(elt) resolve(elt);
7 :        else reject(Error("not found"));
8 :      })
9 :    });
10 :    return p;
11 :  }
12 :  elementById("bob").then(function (elt) {
13 :    elt.innerHTML = "saucisse";
14 :  });
15 : </script>
16 : <body><div id="bob"></div></body>
```

Combinaison sequentielle :

- la méthode `then` renvoie une promesse ;
- les fonctions de rappel peuvent renvoyer directement cette promesse ;
- sinon, c'est une promesse tenue, avec la valeur du `return` ;
- ou une promesse rompue, avec la valeur du `throw` ;
- si une des fonctions des `undefined`, la promesse parente.

Méthodes utilitaires :

- `Promise.resolve(result)`,
- `Promise.reject(Error(message))`,
- `.catch(f)` signifie `.then(undefined,f)`.

Exemple : transformation de promesse rompue en promesses tenue

```
1 : (Math.random () > 0.5 ?
2 :    Promise.resolve('result') :
3 :    Promise.reject(Error('fail')))
4 : .then(
5 :   function(result) { return 'result : ' + result},
6 :   function(err) { return Promise.resolve('failure'); })
7 : .then(
8 :   function(v) { console.log (v); })
```

Exemple : boucle interne en promesse récursive

```
1 :     function countdown (elt, nb) {
2 :       return new Promise (function (resolve, reject) {
3 :         window.setTimeout (function () {
4 :           if (nb == 0) {
5 :             resolve ();
6 :           } else {
7 :             elt.innerHTML = nb;
8 :             resolve (countdown (elt, nb - 1));
9 :           }}, 1000);
10 :      });
11 :    }
12 :    elementById ("bob").then (function (elt) {
13 :      countdown (elt, 5).then (function () {
14 :        elt.innerHTML = "saucisse";
15 :      });
16 :    });
```

Combinaison de promesses :

- `Promise.all`
  - prend un tableau de promesses,
  - renvoie la promesse tenue du tableau des résultats,
  - ou la promesse rompue d'un des échecs.
- `Promise.race`
  - prend un tableau de promesses,
  - renvoie la promesse tenue d'un des résultats,
  - ou la promesse rompue d'un des échecs.

Exemple :

```
1 : Promise.all (
2 :   [[ "bob", 2 ], [ "bab", 5 ], [ "bib", 10 ]].map (
3 :     function ([id, nb]) {
4 :       return elementById (id).then (function (elt) {
5 :         return countdown (elt, nb).then (function () {
6 :           return elt;
7 :         });
8 :       });
9 :     })).then (function (elts) {
10 :       for (var elt of elts) {
11 :         elt.innerHTML = "STOP";
12 :       }
13 :     });
```

Comprendre HTML5

Questions ?

Développement d'Applications Réticulaires