Web Components

Laurent Jouanneau

Novembre 2013 - FranceJS

mozilla

Innophi

mozilla

XULFR.ORG

CasperJS

http://ljouanneau.com
@ljouanneau

http://innophi.com

openweb.eu.org jelix.org slimerjs.org casperjs.org

Qu'est ce qu'un composant web ?

Définition

- Morceau de HTML/JS/CSS qui a une fonction bien précise
  - datepicker, bouton, contrôle video, menu, grille...
- Peut être réutiliser facilement d'une page à une autre
- un moyen technique pour définir ses propres éléments HTML
  - = contenu html + design + API + comportement

Les composants web d'hier et d'aujourd'hui

- Bibliothèque JS proposant des composants tout fait. ex : jQuery UI
  - pour les « instancier » : API ou parfois moyen déclaratif
- Bibliothèques pour faire des templates

Quelques soucis...

- Chargements des scripts et CSS
- Lib souvent lourdes
- Instanciés **après** le document
- « Pollution » du document avec les éléments des composants
- Imbrication pas aisée. Voire impossible..
- Collisions possibles au niveau CSS (cascade), DOM (id), JS (attention au scope)

Les composants web de demain

Le w3c vous a entendu

- Nouvelles spécifications pour supprimer tout nos soucis

Template
+ Shadow DOM
+ Custom Elements
& Décorators
+ html import

= WEB COMPONENTS

Les plus

- Créés en même temps que le document
- Pas de pollution :
  - contenu (template) dans un DOM caché (shadow DOM)
  - encapsulation du code JS et CSS
  - permet une séparation contenu / presentation dans un document HTML
- Utilisation déclarative, moins de code JS
- Réutilisation simplifiée :
  - Custom Elements + import

De XBL aux Web Components

Un peu d'histoire...

La génèse des technos Web Components

- XBL 1 : première implèm. en 2001 dans Gecko
  - pouvoir réaliser des composants pour l'UI de Mozilla Suite / Netscape 6
- Fichier déclaratif : template, shadow DOM, API, comportements...
- Toujours utilisé dans Firefox en XUL mais aussi HTML (contrôles <video>, <audio>...)
- Pas de spécification officielle & stable.
  - https://developer.mozilla.org/en-US/docs/XBL

XBL2 : premier espoir pour le web

- XBL2 : spécification de XBL1++ au W3C
- Candidate Recommandation \o/
  - http://www.w3.org/TR/2007/CR-xbl-20070316/
- Problème : pas d'implémentation native
  - trop lourd, trop gros, trop compliqué
- => Découpage de la spéc en 2011
  - http://lists.w3.org/Archives/Public/public-webapps/2011JulSep/1568.html
- → naissance de Web Components

Pour bientôt ?

- Spécifications indépendantes
- Attention, spécifications en cours d'écriture == « brouillons »
- Ce qui suit est susceptible de changer dans les mois qui suivent.
- Peu utilisable pour le moment (ou presque), mais polyfills
- http://www.w3.org/TR/components-intro/
- Achtuce : http://caniuse.com

Web Components :
Les templates

Contenu déclaratif

- Objectif :
  - déclarer du contenu directement en HTML
  - pouvoir le dupliquer à loisir
- Balise <template>

```
<template id="super-button">
  <div>
    <img src="foo.png">
    <span class="label">Un label</span>
    <span class="dropdown">▼</span>
  </div>
  <ul class="menu">
     <li>item</li>
  </ul>
</template>
```

Utilisation d'un template

- contenu non présent dans le DOM
- propriété « content »
- cloneNode() sur « content » + insertion dans le DOM

```
<template id="super-button">
  <div>
    <img src="foo.png">
    <span class="label">Un label</span>
    <span class="dropdown">▼</span>
  </div>
  <ul class="menu">
     <li>item</li>
  </ul>
</template>
```

```
  <div id="plop">

  </div>
```

```
var tpl = document.getElementById('super-button');
var div = document.getElementById('plop');

var btn = tpl.content.cloneNode(true);
div.appendChild(btn)
```

```
tpl.content.firstElementChild.localName == 'div'
tpl.content.nodeType == tpl.DOCUMENT_FRAGMENT_NODE
```

Pour quand ?

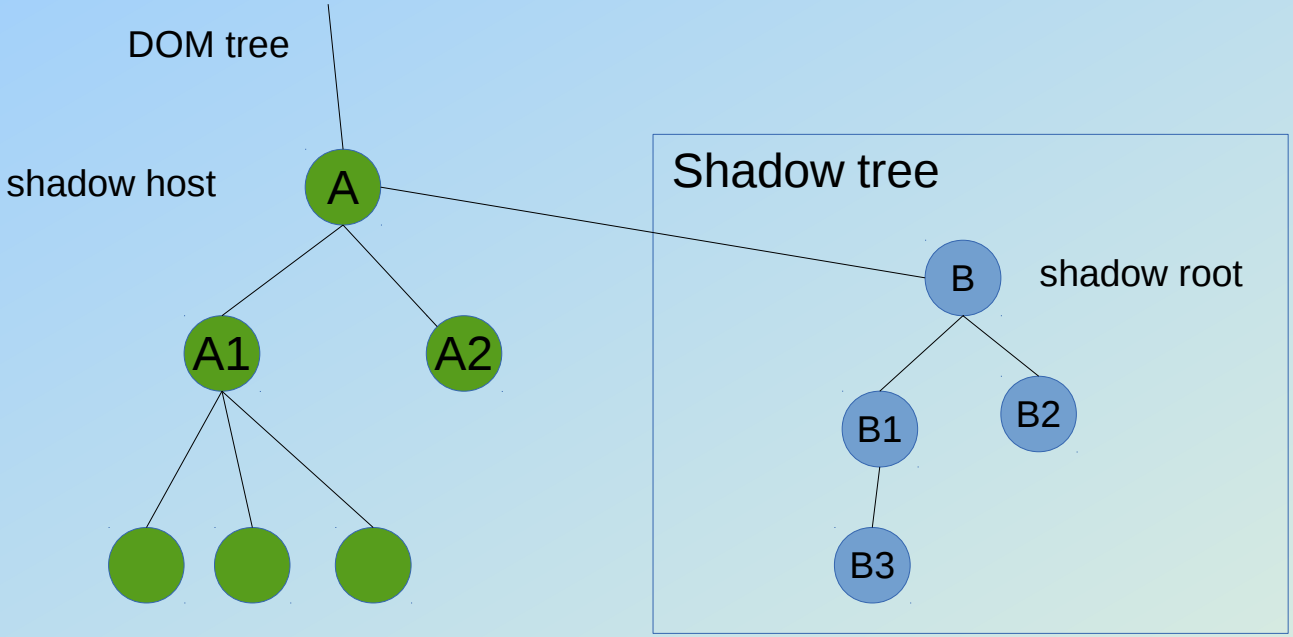
- Maintenant ! (ou presque)
  - implémenté dans Firefox 23+, Chrome 29+, Opera 17+ et...
  - c'est tout ;-)
- Mais spéc toujours en « draft »

Web Components :
Shadow Dom

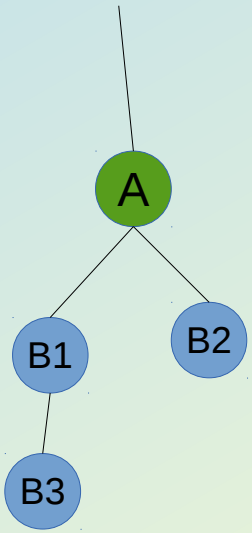
Steack haché où le DOM ?

- (désolé)
- Le Shadow DOM représente des éléments DOM qui sont visibles à l'écran, mais qui n'apparaissent pas dans le DOM « normal »
  - exemple : balise <video>
- Shadow Tree = fragment de document contenant des éléments cachés
- Shadow host : élément DOM normal qui contient un ou plusieurs Shadow Tree

Schématiquement...



Rendu :

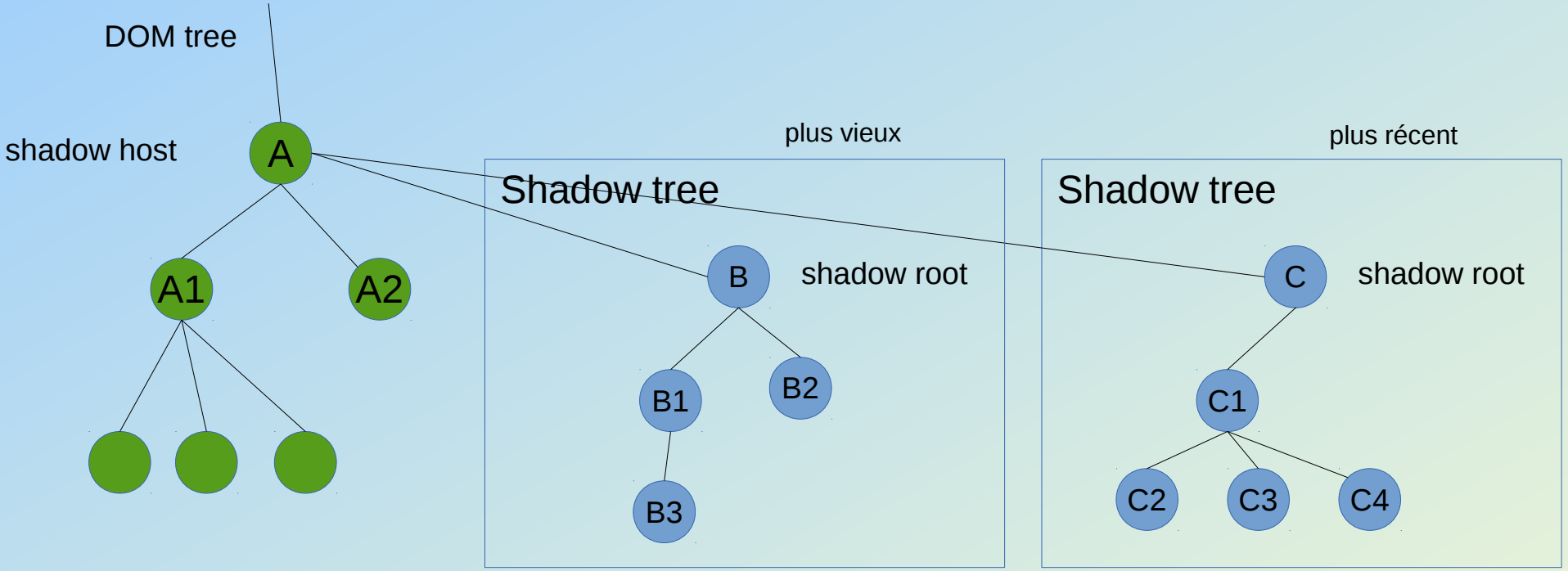


Mais :

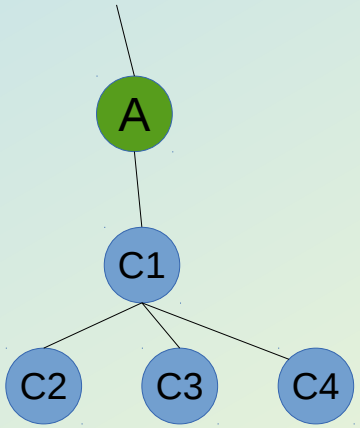
A.firstElementChild == A1

A.childNodes == [A1, A2]

Avec plusieurs shadow tree



Rendu :



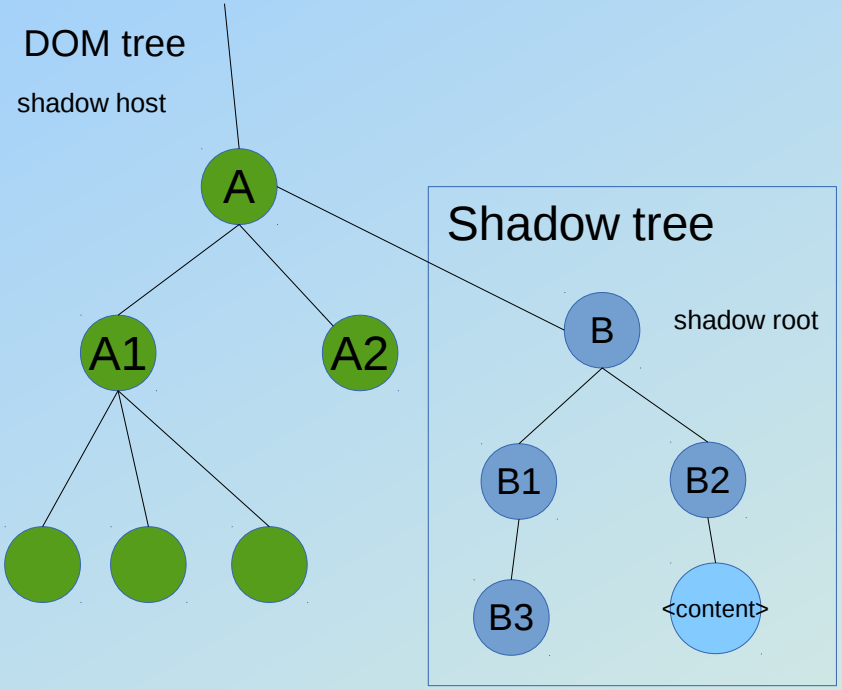
Mais toujours :

A.firstElementChild == A1

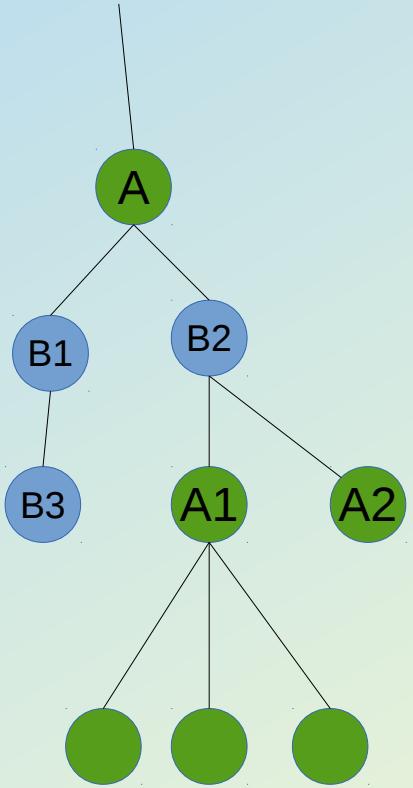
A.childNodes == [A1, A2]

Insérer le contenu existant

- balise <content> == « insertion point » pour les éléments du DOM



Rendu :

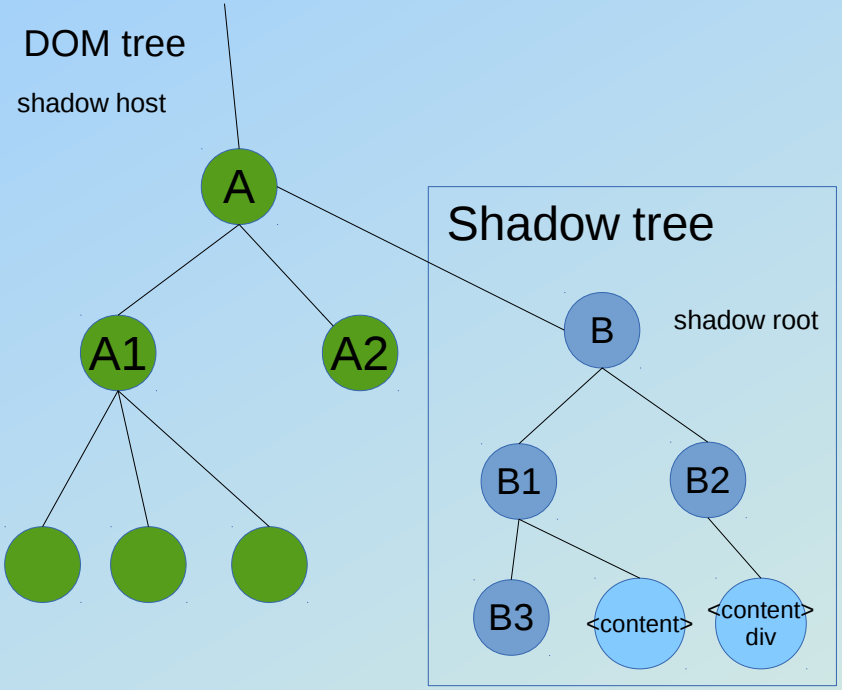


Mais toujours :

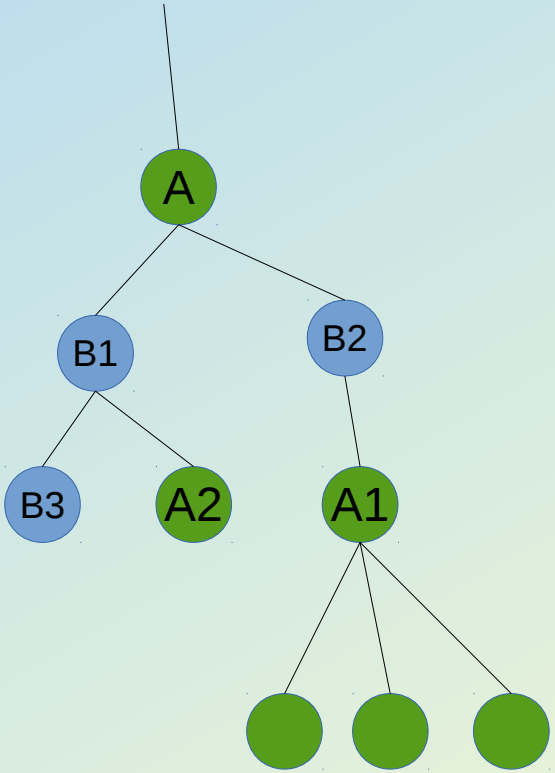
A.firstElementChild == A1

A.childNodes == [A1, A2]

Insérer et dispatcher le contenu existant



Rendu :



- balise <content> peut indiquer un filtre d'élèments
  - <content select="div">
- Plusieurs <content> peuvent être présents
- A1 =

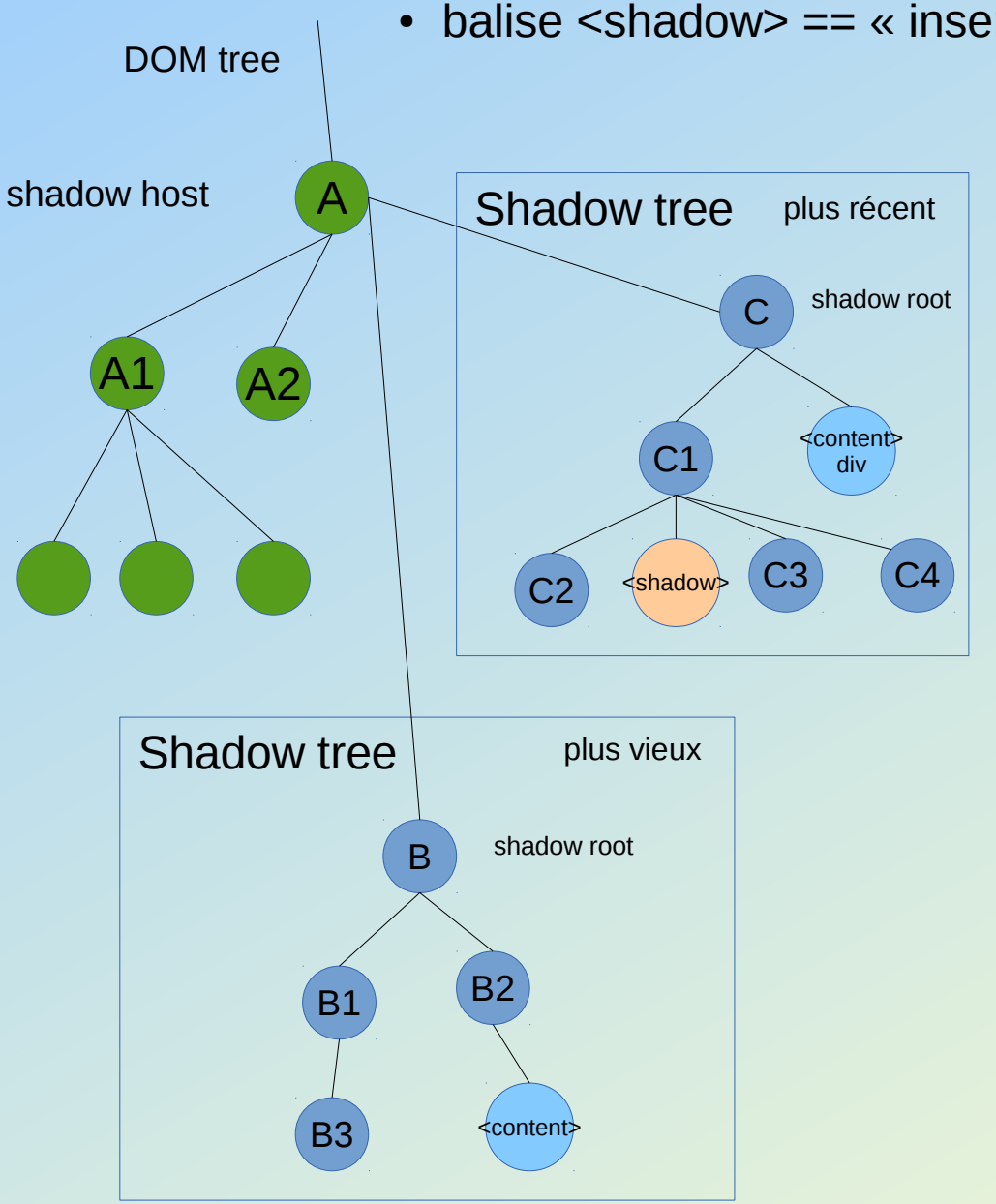
Mais toujours :

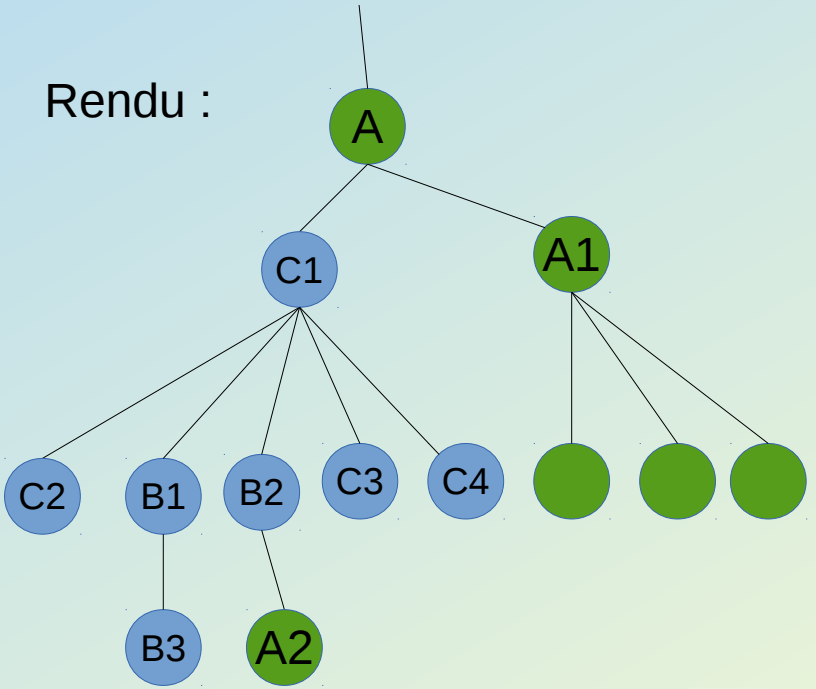
A.firstElementChild == A1

A.childNodes == [A1, A2]

Insérer les shadow tree plus anciens

- balise <shadow> == « insertion point » pour le shadow tree précédent





Mais toujours :

A.firstElementChild == A1

A.childNodes == [A1, A2]

Concrètement

contenu de la page

```
<div id="contenu">
  <h1>My title</h1>
  <p class="intro">Lorem ipsum</p>
  <p>bla bla bla</p>
  <p class="intro">dolor sit</p>
</div>
```

script d'insertion d'un shadowTree

```
var c = document.getElementById('contenu');
var t = document.getElementById('tpl');

var sr = c.createShadowRoot() ;
sr.appendChild(t.content.cloneNode(true)) ;
```

template

```
<template id="tpl">
  <header>
    <content select="h1"/>
  </header>
  <content select=".intro"/>
  <article>
    <content />
  </article>
  <footer>
     <content select=".foot">
       <p>Licence CC</p>
     </content>
   </footer>
</template>
```

==> DOM équivalent **pour le rendu**

```
<div id="contenu">
  <header>
    <h1>My title</h1>
  </header>
  <p class="intro">Lorem ipsum</p>
  <p class="intro">dolor sit</p>
  <article>
    <p>bla bla bla</p>
  </article>
  <footer>
       <p>Licence CC</p>
   </footer>
</div>
```

Pour quoi faire ?

- Un shadow Tree ne « pollue » pas le DOM « normal »
  - → « invisible » pour les scripts de la page
  - → ne perturbe pas les scripts de la page
    - → même si le Shadow Tree est modifié
    - → même si un Shadow Tree est ajouté/enlevé
  - → les scripts de la page n'ont à se soucier que de manipuler le contenu réèl de la page

Feuilles de styles avec les shadow tree

- pas d'application des styles de la page, sauf par héritage (valeur « inherit »)
- Contrôle des styles sur le shadowRoot:
  - propriété applyAuthorStyles
  - propriété resetStyleInheritance
- dans la feuille de style du shadowTree
  - ::distributed(*) : styles sur les points d'insertions
  - @host {..} : styles sur le shadow host

Événements dans les shadow tree

- Certains événements sont stoppés en sortie du ST :
  - événements de mutation
  - abort, error, select, load, change, scroll...
- Les autres sont « reciblés » : « target » == shadow host.

C'est pour quand ?

- Maintenant
  - implémenté Chrome 25+, Opera 15+
  - en cours d'implémentation dans Firefox
- Spéc toujours en « draft »

Web Components : Décorateurs

Kesako ?

- Un moyen de « décorer » un élément existant avec des élements d'un template → création d'un shadow tree
- Attachement du décorateur via un style css
- Exemple : décoration d'une balise <details>. On veut ajouter un « bouton » pour ouvrir et fermer

```
<details open>
    <summary>Timepieces</summary>
    <ul>
      <li>Sundial
      <li>Cuckoo clock
      <li>Wristwatch
    </ul>
</details>
```

Exemple

le décorateur

```
<decorator id="details-open">
    <template>
        <a id="summary">
          &blacktriangledown;
          <content select="summary"></content>
        </a>
        <content></content>
    </template>
</decorator>
```

le style pour attacher le décorateur

```
details[open] {
    decorator: url(#details-open);
}
```

Equivalent rendu :

```
<details open>
    <a id="summary">
        &blacktriangledown;
        <summary>Timepieces</summary>
    </a>
    <ul>
      <li>Sundial
      <li>Cuckoo clock
      <li>Wristwatch
    </ul>
</details>
```

(exemple issu de http://w3c.github.io/webcomponents/explainer/#decorator-section)

C'est pour quand ?

- Aucune implémentation …
- … car même pas un brouillon de spécification

Web Components :
Custom Elements

Définissez vos propres balises

- Grâce à l’élément <element name="foo-bar">
- attribut name = nom de l'élément (avec tiret)
- Définition du contenu grâce à <template> (→ shadow tree)
- Définition du comportement et de son api grâce à <script>
- héritage possible
- callbacks possible sur création/insertion ..

Différence avec les décorateurs

- Toujours « vivant » même retiré du DOM
- déclaration d'une API

Simple exemple

```
<element name="dialog-popup">
    <template>
        <div id="popup-header">
            <content select="h1:first-child">
            </content>
            <button>Close</button>
        </div>
        <section>
            <content></content>
        </section>
        <div id="popup-footer">
            <button>Ok</button>
            <button>Cancel</button>
        </div>
    </template>
    <script>
        //....
    </script>
</element>
```

=>

```
<dialog-popup id="mypopup">
    <h1>My title</h1>
    <p>Lorem Ipsum</p>
    <p>Do you agree?</p>
</dialog-popup>
```

Une API pour notre élément

Custom element

```
<element name="dialog-popup">
    <template>
        ...
    </template>
    <script>
    ({
    open : function() {
        this.style.display = 'block';
    },
    close: function(ok) {
        this.style.display = 'none';
        if (this.onclose) {
            this.onclose(ok)
        }
    },
    onclose: null,
    })
    </script>
</element>
```

Quelque part dans la page

```
<dialog-popup id="mypopup">
    <h1>My title</h1>
    <p>Lorem Ipsum</p>
    <p>Do you agree?</p>
</dialog-popup>
```

```
var d = document.getElementById('mypopup')
d.onclose = function(ok) {
    if (ok)
        alert('OK')
    else
        alert('cancel')
}
d.open()
d.close(true)
```

Héritage

D'un autre custom element

```
<element extends="dialog-popup" name="super-dialog">
    <template>
        ...
    </template>
    <script>
        //....
    </script>
</element>
```

D'un élément natif

```
<element extends="button" name="super-button">
    <template>
        ...
    </template>
    <script>
        //....
    </script>
</element>
```

```
<button is="super-button">
    bla
</element>
```

Callback

- être informé de l'instanciation : created
  - → « constructeur »
- de l'insertion dans le DOM : enteredView
- du détachement : leftView
- changement d'attributs : attributeChanged

Custom element

```
<element name="dialog-popup">
    <template>
        ...
    </template>
    <script>
    ({
    created : function() {
    },
    enteredView: function() {
    },
    leftView: function() {
    },
    attributeChanged: function() {
    },
    })
    </script>
</element>
```

Attention : spécification pas clair au niveau de la déclaration des callbacks

C'est pour quand ?

- Aucune implémentation
- Spécification toujours en travaux

Web Components : import

Importer un web component

- balise <link rel="import" href="dialog.html">
- dialog.html = page html classique
- Seules les balises <element> et <decorator> sont interprétées

C'est pour quand ?

- Pas d'implémentation
- Spécification en brouillon

Des polyfills en attendant, et autres ressources

Polymer, X-tags & co

- Bibliothèques qui servent + ou – de polyfill
  - Polymer (google) https://github.com/Polymer/
  - x-tags (Mozilla) http://www.x-tags.org/
- http://www.html5rocks.com/en/tutorials/webcomponents/
- http://w3c.github.io/webcomponents/explainer

The end

Des questions ?