DÉVELOPPEMENT JAVASCRIPT

jour n°3

JS

6. API DOM

JS

6. API DOM

- C'est quoi l'API DOM ?
- Les objets de l'API DOM
- Sélectionner un élément
- Modifier un élément
- Les événements
- Les Formulaires

DOM

=

Document Object Model

DOCUMENT OBJECT MODEL

- API JS fournie par les navigateurs
- Standard du W3C
- Permet de lire / modifier la page
- Chaque balise = un objet (branche/feuille)
- Le DOM = l'arbre des balises du document

07 April 2004

1. Document Object Model Core

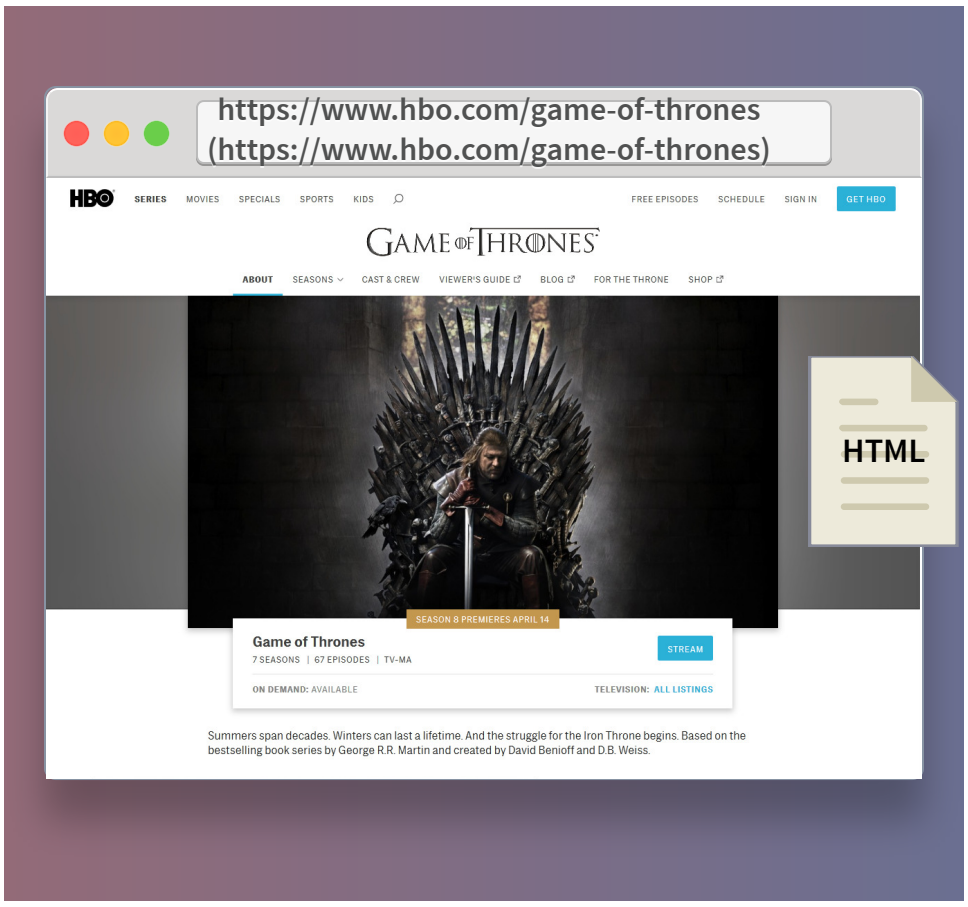
Editors:
Arnaud Le Hors, IBM
Philippe Le Hégaret, W3C
Gavin Nicol, Inso EPS (for DOM Level 1)
Lauren Wood, SoftQuad, Inc. (for DOM Level 1)
Mike Champion, Arbortext and Software AG (for DOM Level 1 from November 20, 1997)
Steve Byrne, JavaSoft (for DOM Level 1 until November 19, 1997)

Table of contents

Notes :

https://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/core.html (https://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/core.html) Spec de l'API DOM sur le site du W3C

LES OBJETS DE L'API DOM



window :
fenêtre/onglet du navigateur

window.document :
document html complet (head & body)

window.document.body :
balise <body>

ELEMENT

- Toutes les balises héritent de la classe `Element`
- propriété `children` : tableau des noeuds enfants
- propriété `attributes` : attributs HTML

3 FAÇONS DE SÉLECTIONNER UN ÉLÉMENT

1. propriété element.children
2. méthodes
   element.getElementsByXXXX()
3. méthode element.querySelector()

SÉLECTION : CHILDREN


```
<!DOCTYPE html>
<html>
<head>
  <title>Tywin last wishes</title>
</head>
<body>
  <ul>
    <li>Kill Tyrion</li>
    <li>Kill The Starks</li>
    <li>Kill The Tyrells</li>
    <li>Kill the dwarf !</li>
  </ul>
</body>
</html>
```

```
const element:Element =
  document
    .body
      .children[0]
        .children[0];
```

SÉLECTION : GETELEMENT...

getElementById(*id*) : 1 Element en fonction de son id

getElementsByTagName(*tagName*) : les Elements dont la balise correspond à "tagName"

getElementsByClassName(*className*) : les Elements ayant la classe CSS "className"

```
<h1 id="lastSeason" class="article">Saison 8</h1>
<article class="s08">14 avril 2019</article>
<article class="s08 e02">21 avril 2019</article>
```

```
const element = document.getElementById('lastSeason');
```

```
const element = document.getElementsByTagName('article');
```

```
const element = document.getElementsByClassName('s08'); // ??
```

Notes :

Ces trois méthodes sont employées pour sélectionner un noeud XML existant. Toutes à l'exception de getElementById retournent un objet NodeList (tableau d'Elements) car plusieurs éléments peuvent potentiellement correspondre à la recherche. Comme il ne peut y avoir qu'un seul élément avec un id donné, getElementById retourne directement un objet Element.

SÉLECTION : QUERYSELECTOR

```
let element = document.querySelector( 'body' );
```

Retourne le 1er élément qui correspond au sélecteur CSS

```
let elementsArray = document.querySelectorAll( 'ul li' );
```

Retourne un tableau de tous les éléments qui correspondent au sélecteur CSS

CSS : SÉLECTEURS DE BASE

```css
div {...} /* type de balise */
.dwarf {...} /* classe css */
#king {...} /* id */
```

```html
<div>Characters :</p>
<div class="dwarf">Tyrion</div>
<div id="king">Tommen</div>
```

CSS : SÉLECTEURS ENFANTS

```css
section p {...} /* les <p> contenus dans <section> */
section > p {...} /* les <p> contenus immédiatement dans <section> */
```

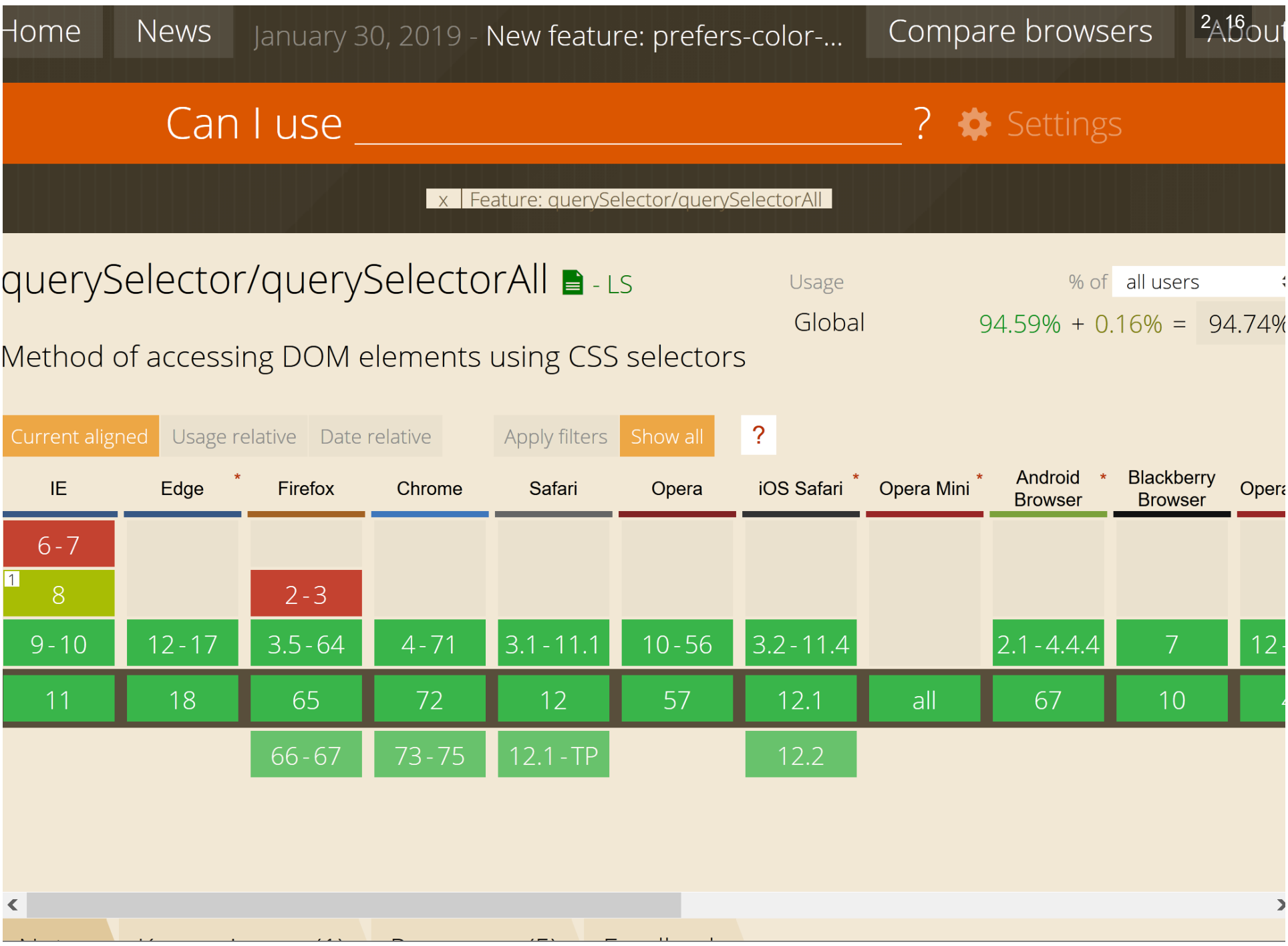
```html
<section>
    <p>Characters :</p>
    <div class="dwarf">
        Tyrion
        <p>best character ever !</p>
    </div>
    <div id="king">Tommen</div>
</section>
```

GETELEMENTXXX VS QUERYSELECTOR

```
document.getElementById('container')
    .getElementsByTagName('ul')[0]
    .getElementsByTagName('li');
```

VS

```
document.querySelectorAll('#container ul:first-child li');
```



Notes :

http://caniuse.com/#feat=queryselector (http://caniuse.com/#feat=queryselector)

Le support navigateur de querySelector est très bon (IE9+), il n'est plus justifié aujourd'hui de s'en passer. Les méthodes getElementXXXX ne sont quasi plus utilisées.

MODIFIER UN ÉLÉMENT

- **element.innerHTML** : lire/modifier le contenu html
- **element.getAttribute() / setAttribute()** : lire/modifier un attribut html

```
<div id="king">Joffrey</div>

<script>
    const king:Element = document.querySelector( '#king' );
    console.log( king.innerHTML + ' is dead ! ヽ(°◇° )ノ');
    king.innerHTML = 'Tommen';
    king.setAttribute( 'class', 'willDieSoon' );
</script>
```

```
<div id="king" class="willDieSoon">Tommen</div>

<script>
    const king:Element = document.querySelector( '#king' );
    console.log( king.innerHTML + ' is dead ! ヽ(°◇° )ノ');
    king.innerHTML = 'Tommen';
    king.setAttribute( 'class', 'willDieSoon' );
</script>
```

ÉVÉNEMENTS

- Attributs HTML : `onload`, `onclick`…
- Programmation : `element.addEventListener()`

```
<a href="#" onclick="onLinkClick();return false">Ceci est un lien</a>

<script>
    function onLinkClick(){
        console.log( 'on a cliqué sur le lien !' );
    }
</script>
```

```
<a href="#">Ceci est un lien</a>

<script>
    function onLinkClick(event){
        event.preventDefault();
        link.innerHTML = 'OMG, on m'a cliqué dessus !';
    }
    const link = document.querySelector( 'a' );
    link.addEventListener('click', onLinkClick );
</script>
```

Notes :

Pour capter les événements déclenchés par le DOM on utilise des écouteurs d'événement, cela peut se faire de deux manières :

1. via les attributs HTML par exemple :

   ```
   <body onload="body_loadedHandler()">
   ```

   body_loadedHandler étant le nom de la fonction à exécuter lorsque l'événement load se produit sur l'élément body ;

2. En JavaScript :

   ```
   document.querySelector( '.myButton' ).addEventListener( 'click', myButton_clickHandler );
   ```

   myButton_clickHandler étant une référence d'une fonction appelée lors du clic sur le bouton de classe myButton.

En général on évite les attributs html onXXXXX="" car ils ont l'inconvénient de "polluer" le code HTML avec du code JS. La méthode addEventListener() a l'avantage de permettre de bien séparer les responsabilités entre le contenu (le document HTML) et l'applicatif (le JS).

HTML CSS JS

2 . 22

```
function showDragons( event ){
  // on empêche le navigateur de traiter le click
  event.preventDefault();
  // on récupère un tableau de références vers les dragons
  const dragons = document.querySelectorAll('.dragon');
  dragons.forEach( dragon => {
    // on ajoute un attribut style à chaque dragon
    dragon.setAttribute('style', 'display:block;');
    // ou
    //dragon.style.display = 'block';
  });
}
// on écoute le click sur le lien
document.querySelector('a').addEventListener( 'click', showDragons );
```

Resources

Notes :

Exemple de mise en oeuvrve de l'API DOM pour :

1. détecter le clic sur un lien
2. afficher des éléments jusque là masqués

http://codepen.io/kumquats/pen/zBZwJm/ (http://codepen.io/kumquats/pen/zBZwJm/)

LES OBJETS DE TYPE EVENT

- propriétés :

  `currentTarget` : l'élément sur lequel on a écouté l'événement

  `target` : l'élément qui a déclenché l'événement

- Méthodes :

  `preventDefault()` : empêche le comportement par défaut du browser

  `stopPropagation()` : empêche l'événement de remonter à la balise parente

Notes :

Les fonctions associées à un écouteur d'événement, reçoivent automatiquement en paramètre un objet de type `Event`. Cet objet a plusieurs propriétés et méthodes qui vont être utile, mais les principales sont :

- `currentTarget` : élément HTML sur lequel on a écouté l'événement (celui sur lequel on a fait addEventListener())
- `target` : élément HTML qui a déclenché l'événement (peut être un enfant de l'élément sur lequel on écoute l'événement)
- `initEvent()` : permet d'initialiser un événement créé manuellement
- `preventDefault()` : annule l'événement, les traitements par défaut liés à l'événement ne se produiront pas
- `stopPropagation()` : arrête la propagation de l'événement

EVENT & SCOPE

"this" est toujours l'Element qui a déclenché l'événement

```
class Listener {
  constructor(element:Element) {
    this.element = element;
    this.element.addEventListener('click', this.onClick );
  }
  onClick(event:Event){
    this.element.innerHTML = 'hodor'; // Erreur !
  }
}
const a = new Listener(document.querySelector('#link'));
```

```
class Listener {
  constructor(element:Element) {
    this.element = element;
    this.element.addEventListener('click', event=>this.onClick(event));
  }
  onClick(event:Event){
    this.element.innerHTML = 'hodor';
  }
}
const a = new Listener(document.querySelector('#link'));
```

```
class Listener {
  constructor(element:Element) {
    this.element = element;
    this.element.addEventListener('click', this.onClick.bind(this));
  }
  onClick(event:Event){
    this.element.innerHTML = 'hodor';
  }
}
const a = new Listener(document.querySelector('#link'));
```

```
class Listener {
  constructor(element:Element) {
    this.element = element;
    this.onClick = this.onClick.bind(this);
    this.element.addEventListener('click', this.onClick);
  }
  onClick(event:Event){
    this.element.innerHTML = 'hodor';
  }
```

```
}
const a = new Listener(document.querySelector('#link'));
```

FORMULAIRES (1/3)

API DOM va permettre de :

- Vérifier ce que saisi l'utilisateur
- Modifier les valeurs des champs (corriger une valeur)
- Soumettre les données en AJAX (cf. prochain cours)

FORMULAIRES (2/3)

```
<form>
    <input type="text" name="message">
    <input type="submit" value="Valider">
</form>
```

Détecter la modification d'un champ :

```
const input = document.querySelector('form input[name=message]');
input.addEventListener('change', event=>console.log('input changed'));
```

Lire / modifier la valeur d'un champ :

```
input.value = 'Hodor !'
```

FORMULAIRES (3/3)

écouter la soumission du formulaire

```
let form = document.querySelector('form');
form.addEventListener('submit', event => {
    event.preventDefault();
    // ...
});
```

JS : DOM : Formulai...

A PEN BY Kumquats

Save | Fork | Settings | Change View

2 . 29

HTML

```
1 ▾ <h1>Ce qu'il va se passer en saison
    8</h1>
2 ▾ <form>
3     Le personnage
```

CSS

JS

```
1  // lorsqu'on soumet le formulaire on vérifie
   les règles de gestion suivantes :
2  // le personnage ne peut pas être vide
3  // le futur ne peut pas être vide
```

960px

Ce qu'il va se passer en saison 8

Le personnage va Valider

Console Assets Comments Shortcuts Last saved 5 days ago Delete Collections Embed Export Share

Notes :

Exemple de mise en oeuvre de l'API DOM avec un formulaire : https://codepen.io/kumquats/pen/bzxNgX?editors=1010 (https://codepen.io/kumquats/pen/bzxNgX?editors=1010)

MERCI !