React

JSX en featuring sur le nouvel album de Virtual DOM

Quand un navigateur lit un fichier HTML, il l'analyse pour générer le **DOM**, pour **Document Object Model**). Le DOM est une représentation du HTML, dans laquelle chaque élément, par exemple une div, un span, un h1 etc, est représenté par un noeud avec ses enfants et ses parents. Le tout formant un arbre de noeuds.

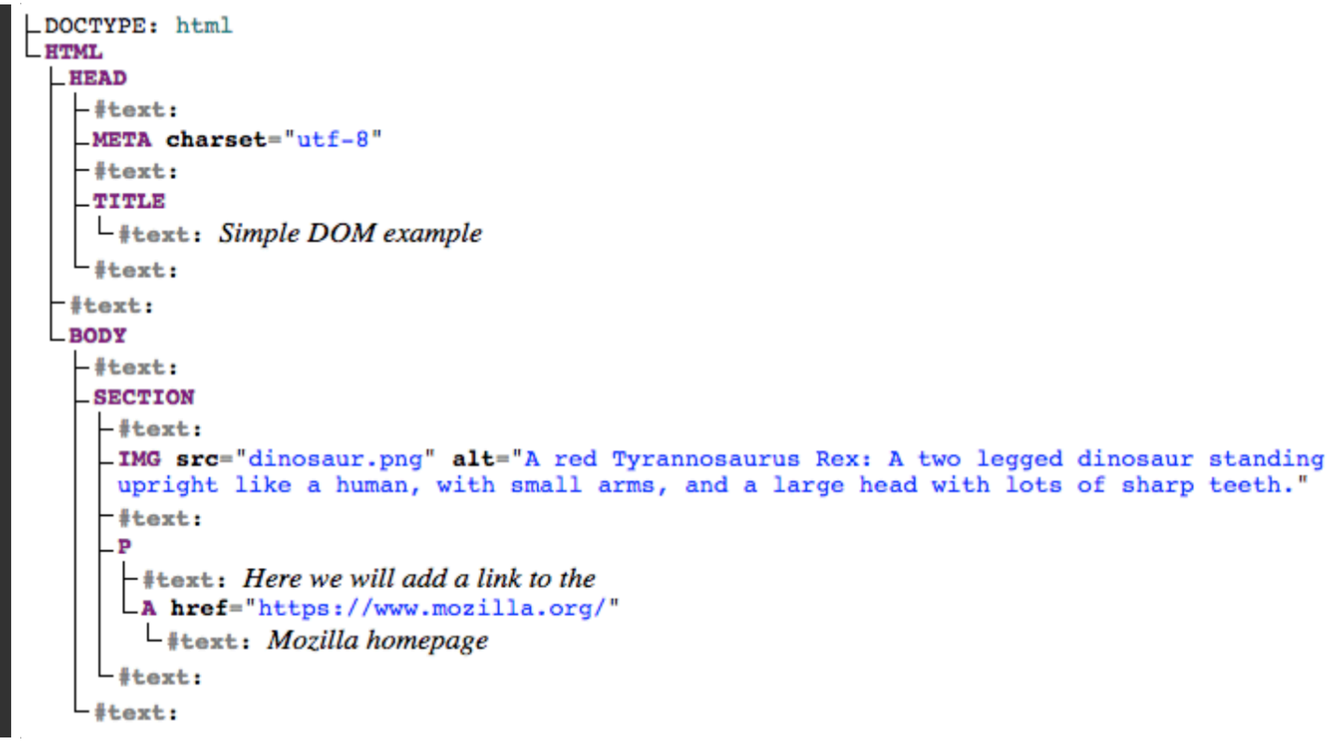
Ce DOM est ensuite exposé au Javascript, qui peut le lire et le modifier. En Javascript on peut modifier un node existant, en ajouter, en supprimer etc, grâce à ce que l'on appelle la DOM API. La modification de ces nodes dans le DOM aura pour effet de modifier le HTML visible sur le navigateur.

En ayant accès à la modification du DOM, le Javascript permet ainsi d'ajouter de l'interactivité sur les pages web : afficher ou cacher du contenu au clic sur un bouton, charger de nouveaux articles (via des requêtes **AJAX**) dès que le scroll atteint le bas de page sur un blog, afficher une nouvelle image dans un slider toutes les x secondes etc…

HTML

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Simple DOM example</title>
  </head>
  <body>
      <section>
        <img src="dinosaur.png" alt="A red Tyrannosaurus Rex: A two
legged dinosaur standing upright like a human, with small arms, and a
large head with lots of sharp teeth.">
        <p>Here we will add a link to the <a
href="https://www.mozilla.org/">Mozilla homepage</a></p>
      </section>
  </body>
</html>
```

DOM généré



Images : https://developer.mozilla.org/

Historiquement, le fichier HTML que le navigateur utilise est d'abord généré et envoyé par le serveur (**SSR**, voir quête précédente). Le Javascript est ensuite exécuté sur le DOM existant pour ajouter de l'interactivité. Avec la demande d'interactivité grandissante, les frameworks Javascript comme **React** ont choisi une autre approche : générer la quasi intégralité du DOM en Javascript afin de le manipuler plus facilement. On appelle ça le **Client Side Rendering** (**CSR**).

Avec le CSR, le fichier HTML initialement lu par le navigateur ne contient qu'une seule div vide, dont le but est seulement de servir de « conteneur » pour injecter le HTML généré en Javascript.

Exemples de CSR très basique (sans React) :

HTML

```html
<html>
  <body>
    <div id="root"></div>
  </body>
</html>
```

Javascript

```javascript
const rootElement = document.getElementById('root')
const element = document.createElement('div')
element.textContent = 'Hello World'
element.className = 'container'

rootElement.append(element)
```

React utilise la même DOM API que celle utilisée en JS pour créer des nodes (c'est à dire des éléments HTML) et les insérer dans le DOM, avec une syntaxe plus déclarative.

Mais il faut savoir que ce que l'on appelle communément React est en fait deux librairies :
- **React** pour créer des éléments React (similaire à createElement() en Javascript)
- **ReactDom**, pour insérer les éléments React éléments dans le DOM (similaire à append() en Javascript)

React peut être aussi utilisé avec **ReactNative** à la place de **ReactDOM** pour les applications mobiles.

Exemples de CSR très basique avec React :

HTML

```html
<html>
  <body>
    <div id="root"></div>
  </body>
</html>
```

React + ReactDOM

```
const reactElement = React.createElement('div', {
  className: 'container',
  children: 'Hello World',
})

const rootElement = document.getElementById('root')

ReactDOM.render(element, rootElement)
```

Même si la syntaxe est un peu plus déclarative avec React, ce n'est toujours pas très pratique de créer et insérer des éléments dans le DOM. Pour rendre la création de ces éléments encore plus déclarative et donc plus agréable à écrire, React utilise la syntaxe **JSX**.

Cela permet d'avoir une syntaxe qui ressemble à première vue à du HTML, mais qui sera transformée, grâce à **Babel**, en Javascript. L'étape de compilation du JSX en Javascript est indispensable car le navigateur ne sait pas lire le JSX.
Il existe un outil sur le site de Babel pour tester la syntaxe JSX et voir le Javascript généré : https://babeljs.io/repl.

Les trois exemples suivants permettent de créer un élément pour le DOM. On y voit que la syntaxe JSX est clairement la plus concise.

Javascript

Javascript + React

React + JSX

Reprenons depuis le début :

Quand on utilise React, le navigateur reçoit un ficher HTML comportant seulement une div vide. Une fois que ce HTML nu est chargé et que le DOM est généré par le navigateur, React va pouvoir le manipuler.
La librairie React et la syntaxe JSX permettent de créer des éléments avec une syntaxe très concise.
La librairie React DOM permet elle de les insérer dans le DOM.

On peut donc dire que ces éléments créés (et qui seront insérer dans le DOM) représentent ce qui sera affiché à l'utilisateur dans le navigateur : c'est ce que l'on appelle le **DOM Virtuel**.
C'est une représentation du DOM, en attente d'être synchronisée avec le véritable DOM (dans un processus que l'on appelle la « **réconcilation** »).