Éric Sarrion

Vue.js

Cours et exercices

Éditions EYROLLES

Vue.js

Un ouvrage pratique sur le framework JavaScript le plus populaire

Vue.js fait partie des derniers frameworks JavaScript à avoir vu le jour. Après le long règne de jQuery, de nouvelles bibliothèques JavaScript sont apparues pour tenter de remédier aux défauts de leur aînée. Parmi celles actuellement en vogue, on peut citer React.js, Angular et Vue.js.

En introduisant la notion de réactivité et le développement d'application par composants, ces bibliothèques ont révolutionné la façon de développer une application web avec JavaScript. De nouvelles manières d'appréhender le développement sont devenues nécessaires.

Parmi ces nouvelles bibliothèques, Vue.js est la plus simple d'approche. Ses concepteurs ont délibérément gommé les imperfections entrevues dans React.js et Angular, pour faire mieux et plus simplement. Et ils y sont bien arrivés, comme on va le découvrir au fil de cet ouvrage.

À qui s'adresse cet ouvrage ?

- **Aux étudiants, développeurs et chefs de projet**
- **À tous les autodidactes férus de programmation qui veulent découvrir Vue.js**

Au sommaire

Formateur et développeur en tant que consultant indépendant, **Éric Sarrion** participe à toutes sortes de projets informatiques depuis plus de 30 ans. Auteur des best-sellers ***jQuery & jQuery UI, Programmation avec Node.js, Express.js et MongoDB,*** et ***jQuery mobile*** aux éditions Eyrolles, il est réputé pour la limpidité de ses explications et de ses exemples.

www.editions-eyrolles.com

Vue.js

DANS LA MÊME COLLECTION

C. DELANNOY. – **Programmer en C++ moderne** *De C++11 à C++20.*
N°67895, 10e édition, 2019, 884 pages.

A. TASSO. – **Le livre de Java premier langage** *Avec 109 exercices corrigés.*
N°67840, 13e édition, 2019, 600 pages.

H. BERSINI, P. ALEXIS, G. DEGOLS. – **Apprendre la programmation web avec Python et Django**.
N°67515, 2018, 396 pages.

H. BERSINI, I. WELLESZ. – **La programmation orientée objet**.
Cours et exercices en UML 2 avec Java, C#, C++, Python, PHP et LinQ.
N°67399, 7e édition, 2017, 696 pages.

J. ENGELS. – **PHP 7.**
N°67360, 2017, 585 pages.

P. ROQUES. – **UML 2.5 par la pratique.** *Études de cas et exercices corrigés.*
N°67565, 8e édition, 2018, 408 pages.

C. SOUTOU. – **Programmer avec MySQL.**
N°67379, 5e édition, 2017, 522 pages.

G. SWINNEN. – **Apprendre à programmer avec Python 3.**
N°13434, 3e édition, 2012, 435 pages.

Éric Sarrion

Vue.js

Cours et exercices

Éditions
EYROLLES

ÉDITIONS EYROLLES
61, bd Saint-Germain
75240 Paris Cedex 05
www.editions-eyrolles.com

Avant-propos

Pourquoi un livre sur Vue.js ?

Vue.js fait partie des derniers *frameworks* JavaScript à avoir vu le jour. Après le long règne de jQuery, de nouvelles bibliothèques JavaScript sont apparues pour tenter de remédier aux défauts de leur aînée. Parmi celles actuellement en vogue, on peut citer React.js, Angular et Vue.js.

En introduisant la notion de réactivité et le développement d'application par composants, ces bibliothèques ont révolutionné la façon de développer une application web avec JavaScript. De nouvelles manières d'appréhender le développement sont devenues nécessaires.

Parmi ces nouvelles bibliothèques, Vue.js est la plus simple d'approche. Ses concepteurs ont délibérément gommé les imperfections entrevues dans React.js et Angular, pour faire mieux et plus simplement. Et ils y sont bien arrivés, comme on va le découvrir au fil de cet ouvrage.

Guide de lecture

Ce livre se lit dans l'ordre des chapitres tels qu'ils sont écrits. En effet, les connaissances acquises dans chacun sont utilisées dans les suivants afin de construire des applications web toujours plus complexes.

À l'issue de la lecture, vous aurez une vision très concrète sur l'utilisation de Vue.js, aussi bien dans la globalité que dans les détails.

Public concerné

Les décideurs informatiques trouveront dans ce livre un intérêt pour l'utilisation de Vue.js dans les projets de leur société.

Les étudiants et professeurs des écoles d'ingénieurs le liront également pour se former à l'utilisation de ce *framework* en vogue et très utilisé en entreprise.

Enfin, les développeurs et les chefs de projets auront un manuel de référence quotidien sur l'utilisation de Vue.js.

Remerciements

Je remercie les éditions Eyrolles, particulièrement Alexandre Habian, pour la confiance qui m'est témoignée depuis de nombreuses années.

Table des matières

1

JavaScript ES6

JavaScript, langage facilitant les interactions entre une page web et l'internaute, a connu diverses évolutions ces dernières années, dont la plus significative est celle de la version ES6 (abréviation de ECMAScript 6). Cette nouvelle version concerne particulièrement :

- les variables : déclaration, portée et mise en forme dans les chaînes de caractères ;
- les fonctions : paramètres par défaut, nouvelle forme de déclaration de fonction ;
- les objets et les tableaux : déstructuration et opérateur `"..."` ;
- les classes d'objets : création et dérivation ;
- les promesses : utilisation du processus asynchrone ;
- les modules : pour mieux structurer le code JavaScript.

Dans la suite du chapitre, nous utiliserons un fichier `index.html` qui contiendra le code JavaScript utilisé (au moyen de balises `<script>`). Voici sa structure :

Fichier index.html

```
<html>

  <head>
  </head>

  <body>
  </body>

  <script>
```

```
    // ici le code JavaScript
    // ...

  </script>

</html>
```

Ce fichier HTML peut être exécuté soit en le faisant glisser dans la fenêtre d'un navigateur (depuis le gestionnaire de fichiers), soit en tapant l'URL http://localhost/vue en supposant que vous avez lancé un serveur (PHP, Node.js, J2EE ou autre) et que vous avez déposé le fichier `index.html` dans le répertoire `vue` du serveur. Dans ce chapitre, nous utilisons la première solution (glissement dans la fenêtre du navigateur).

Bien sûr, ce fichier étant vide de code pour l'instant, son exécution produit une page blanche à l'écran.

Les variables

Divers mots-clés ont été ajoutés au langage JavaScript afin de modifier la portée des variables. Le mot-clé `var` permettant de définir une variable locale est toujours actif, mais ses effets de bord ont été corrigés comme on va le voir.

Utilisation de const

Le mot-clé `const` désigne une constante qui, par définition, ne pourra plus être modifiée. Une constante ne peut être affectée qu'une seule fois ; toute autre tentative conduit à une erreur JavaScript.

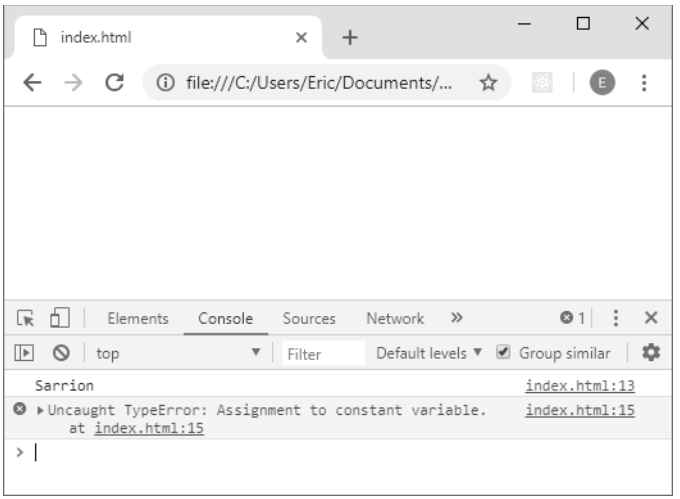
Écrivons le code suivant dans le fichier `index.html`, dans la partie réservée (balise `<script>`).

Utilisation de const

```
const nom = "Sarrion";
console.log(nom);
nom = "Martin";   // erreur
```

La modification de la constante `nom` provoque une erreur que l'on peut voir dans un navigateur, par exemple Chrome en utilisant ses outils de développement (touche *F12* puis onglet *Console*).

Figure 1–1



Utilisation de let

Le mot-clé `let` définit de vraies variables locales, qui disparaissent lorsque le bloc de code dans lequel elles sont définies n'est plus exécuté. De plus, si une variable du même nom est définie à un niveau supérieur dans le code, la nouvelle variable définie avec `let` ne l'écrase pas. Le comportement de `let` est bien différent de celui de `var`.

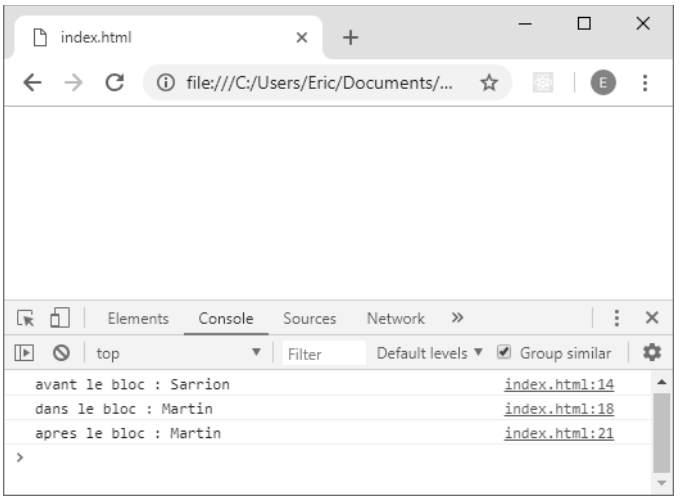
Voyons sur un exemple la différence entre `var` et `let` utilisés dans un bloc de code (entouré des accolades).

Avec utilisation de var dans un bloc

```
var nom = "Sarrion";

console.log("avant le bloc : " + nom);    // "Sarrion"

if (true) {
  var nom = "Martin";
  console.log("dans le bloc : " + nom);   // "Martin"
}

console.log("apres le bloc : " + nom);    // "Martin"
```

La variable `nom` est d'abord définie hors du bloc (à la valeur `"Sarrion"`). Une variable du même nom est créée dans le bloc et reçoit une autre valeur (`"Martin"`) ; déclarée à l'aide de `var`, elle vient écraser la variable du même nom définie avant le bloc et est ensuite affichée avec cette nouvelle valeur après le bloc.

Figure 1–2



La variable définie dans le bloc avec `var` n'est pas vue comme étant locale à ce dernier. En fait, on n'a pas deux variables, mais une seule en mémoire.

Note

L'effet d'écrasement que l'on peut observer vient du fait que la variable `nom` définie dans le bloc à l'aide de `var` n'est pas locale à ce bloc, mais vient prendre la place d'une éventuelle variable du même nom définie précédemment.

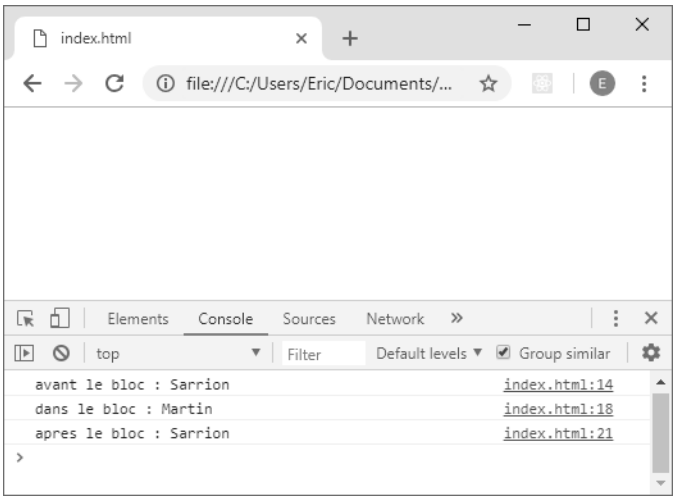
Avec `let`, on va créer réellement des variables locales à un bloc, sans interférer avec d'autres variables du même nom définies ailleurs.

Avec utilisation de let dans un bloc

```
var nom = "Sarrion";

console.log("avant le bloc : " + nom);    // "Sarrion"

if (true) {
  let nom = "Martin";
  console.log("dans le bloc : " + nom);   // "Martin"
}

console.log("apres le bloc : " + nom);    // "Sarrion"
```

On utilise maintenant le mot-clé `let` pour définir la variable dans le bloc. Le résultat est très différent du précédent exemple : la variable `nom` a maintenant une valeur différente dans le bloc et en dehors de celui-ci.

Figure 1–3



Voyons dans un second exemple la différence entre `var` et `let`. Dans une boucle `for`, une variable déclarée par `var` ou par `let` a une incidence sur le comportement du code JavaScript.

Modifions le fichier `index.html` pour créer cinq éléments `<div>` et interceptons le clic sur chacun d'eux. On utilise tout d'abord le mot-clé `var` pour définir l'indice `i` dans la boucle.

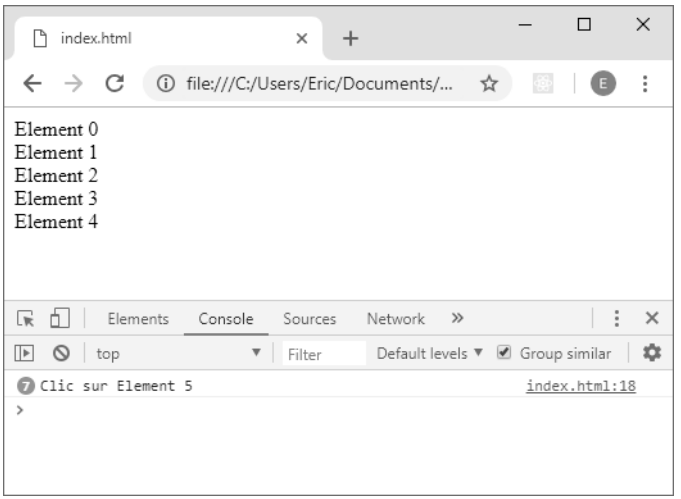
Clic sur les éléments créés avec un indice de boucle défini par var dans une boucle for

```
for (var i=0; i<5; i++) {   // utilisation de var pour définir i dans le bloc
  var div = document.createElement("div");
  var text = document.createTextNode("Element " + i);
  div.appendChild(text);
  document.getElementsByTagName("body")[0].appendChild(div);

  div.onclick = function() {
    console.log("Clic sur Element " + i);
  }
}
```

Ce programme effectue simplement une boucle de `0` à `4` inclus, afin de créer des éléments `<div>` sur lesquels on place un gestionnaire d'événements `onclick`. Chaque clic sur un élément `<div>` affiche `"Clic sur Element "` suivi de l'index de l'élément `<div>` sur lequel on a cliqué.

Quel que soit l'élément `<div>` sur lequel on clique, le programme affiche `"Clic sur Element 5"`, alors que cet élément `5` n'existe même pas dans la page !

Figure 1–4



En effet, la variable `i` définie par `var` n'est pas locale au bloc dans lequel elle est définie et vient donc écraser son ancienne valeur. À la fin de la boucle, elle atteint la valeur `5` et c'est cela qui provoque la fin de la boucle. Par la suite, le clic sur n'importe quel élément `<div>` récupère la valeur finale de cette variable, ici la valeur `5`.

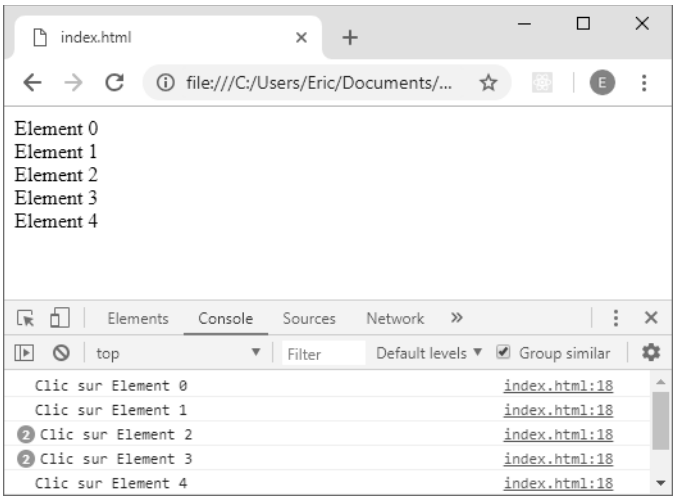
On observe un comportement bien différent avec le mot-clé `let` pour définir la variable `i` dans la boucle `for`.

Clic sur les éléments créés avec un indice de boucle défini par let dans une boucle for

```
for (let i=0; i<5; i++) {   // utilisation de let pour définir i dans le bloc
  var div = document.createElement("div");
  var text = document.createTextNode("Element " + i);
  div.appendChild(text);
  document.getElementsByTagName("body")[0].appendChild(div);

  div.onclick = function() {
    console.log("Clic sur Element " + i);
  }
}
```

On voit clairement que la variable `i` définie par `let` est maintenant locale à la boucle `for` et que chaque clic indique bien l'élément pointé.

Figure 1–5



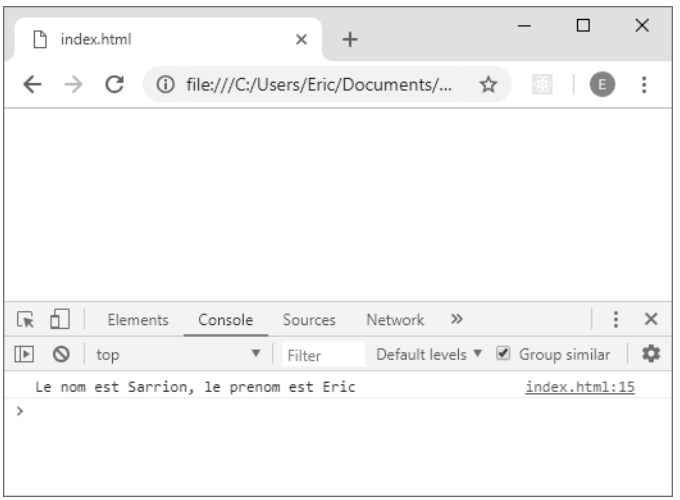
Mise en forme des chaînes de caractères

L'utilisation de variables dans une chaîne JavaScript est un peu fastidieuse, car il faut fermer chaque chaîne de caractères avant d'utiliser la variable à lui concaténer. On écrit souvent des lignes de code telles que celles-ci :

Concaténation de variables dans les chaînes de caractères

```
var nom = "Sarrion";
var prenom = "Eric";

var txt = "Le nom est " + nom + ", le prenom est " + prenom;
console.log(txt);
```

Figure 1–6



ES6 autorise une écriture plus claire, sans fermer la chaîne de caractères lors de la concaténation des variables. On utilise pour cela le guillemet simple inversé (`) pour définir la chaîne de caractères (au lieu du guillemet droit simple ' ou double " traditionnel) et chaque variable est insérée dans la chaîne au moyen de ${*variable*} :

Concaténation de variables dans les chaînes de caractères avec ES6

```
var nom = "Sarrion";
var prenom = "Eric";

var txt = `Le nom est ${nom}, le prenom est ${prenom}`;
console.log(txt);
```

Les fonctions

La déclaration des fonctions a été améliorée en ES6, pour la rendre plus performante et plus concise à écrire.

Utilisation de paramètres par défaut

On peut maintenant passer des valeurs par défaut à des paramètres de fonction (en utilisant le signe =), comme cela se pratique dans d'autres langages de programmation.

La règle est de définir les paramètres par défaut en fin de déclaration des paramètres de la fonction. Dès qu'une valeur par défaut est indiquée pour un paramètre, tous les autres paramètres qui suivent doivent également en avoir une (sinon une ambiguïté se crée lors de l'appel de la fonction et il faut dans ce cas indiquer la valeur `undefined` pour cet argument lors de l'appel).

Utilisation des valeurs par défaut dans les fonctions

```
function log(nom="Sarrion", prenom="Eric") {
  console.log(`${nom} ${prenom}`);
}

log("Martin", "Gerard");    // "Martin Gerard"
log();                      // "Sarrion Eric"
log("Martin");              // "Martin Eric"
log(undefined, "Gerard");   // "Sarrion Gerard"
log(null, "Gerard");        // "null Gerard"
log("", "Gerard");          // " Gerard"
```

Ici, la fonction `log()` a deux paramètres dotés de valeurs par défaut et nous l'utilisons avec 0, 1 ou 2 arguments afin de voir son comportement.