Les directives structurelles

Ibrahim ALAME

14/02/2023

1 Affichage conditionnel avec les directives v-if, v-else et v-else-if

1.1 La directive v-if

La directive v-if permet d'afficher ou non un bloc conditionnellement.

```
<h1 v-if="condition1 && condition2">
        Affiché si condition1 et condition2 valent true
</h1>
```

Autrement dit, elle permet d'insérer dynamiquement sur le DOM un bloc d'un ou plusieurs éléments HTML si une ou plusieurs conditions est/sont remplie(s), et de les enlever du DOM dans le cas contraire.

v-if doit forcément être utilisé sur un élément. Si vous voulez l'utiliser sur un ensemble d'éléments HTML de même niveau, il faut les imbriquer dans un template qui est un élément invisible (il ne sera pas inséré sur le DOM).

```
<template v-if="condition">
  <h1>Titre</h1>
  <p>Paragraphe 1</p>
  <p>Paragraphe 2</p>
</template>
```

1.2 La directive v-else

Comme son nom l'indique, v-else permet d'afficher un bloc alternatif si la condition passée à v-if n'est pas remplie. L'élément sur lequel est attaché v-else doit être immédiatement après l'élément auquel est attaché un v-if.

```
<button @click="condition = !condition">Toggle</button>
```

```
<h1 v-if="condition">Condition vaut true</h1>
<h1 v-else>Condition vaut false</h1>
```

1.3 La directive v-else-if

v-else-if permet d'afficher un bloc alternatif si la condition de v-if n'est pas remplie. Il est possible de chaîner les v-else-if :

```
<template>
  <input type="text" v-model="letter" />
  <h1 v-if="letter === 'A'">A</h1>
  <h1 v-else-if="letter === 'B'">B</h1>
  <h1 v-else-if="letter === 'C'">C</h1>
  <h1 v-else>Autre</h1>
</template>

<script setup lang="ts">
import { ref } from 'vue';

const letter = ref('');
</script>

<style scoped lang="scss"></style>
```

2 Les directives v-show et v-pre

2.1 La directive v-show

La directive v-show est très similaire à la directive v-if. Elle permet également d'afficher conditionnellement un élément. L'usage est identique et nous n'allons donc pas nous y attarder :

```
Visible !
```

2.2 Quelle est la différence avec la directive v-if ?

- v-if permet d'insérer ou de supprimer un élément du DOM suivant une condition.
- v-show permet d'afficher ou de ne pas afficher un élément suivant une condition.

v-show agit en fait sur la propriété CSS display de l'élément. L'élément est donc toujours présent sur le DOM et rendu dès l'affichage de la page. Cependant, il est affiché ou non suivant que la condition est remplie. Il est à noter que v-show ne peut pas être utilisée avec template ou avec v-else et v-else-if.

2.3 Quand est-il recommandé d'utiliser v-if et v-show ?

Il faut utiliser v-show lorsqu'un élément sera affiché puis non affiché de manière répétée. En effet, le coût en performances de rendre visible et de cacher un élément est faible. Il n'y a pas de manipulation du DOM mais seulement la modification d'une propriété CSS.

A l'inverse, il vaut mieux utiliser v-if si un élément sera affiché ou non suivant une condition et que cela ne changera pas après l'affichage de la page. En effet, contrairement à v-show, si la condition n'est pas remplie, l'élément ne sera pas inséré sur le DOM et cela coûtera donc moins cher en termes de performances.

2.4 La directive v-pre

La directive v-pre permet de sauter la compilation par Vue.js pour un élément. Cela sert très rarement et uniquement pour afficher des syntaxes Vue.js sans les compiler, par exemple :

```
<span v-pre>{{ Cela sera affiché tel quel }}</span>
```

3 La directive v-for

3.1 Syntaxe de la directive v-for

v-for est une directive permettant d'afficher une liste d'éléments contenus dans un itérable (par exemple un tableau).

La syntaxe de v-for est la suivante : `v-for="element in elements"`. Vous pouvez également utiliser l'opérateur of à la place de in, cela ne fait aucune différence. elements étant le tableau contenant une liste d'éléments.

3.2 Accès à l'index de l'élément dans le tableau

La directive v-for permet de nous donner très facilement accès à l'index de l'élément dans le tableau. La syntaxe est la suivante : `v-for="(element, index) in elements"`. Par exemple :

```
<li v-for="(elem, index) in elements">
  {{ index }} - {{ elem }}
</li>
```

Avec un tableau côté script :

```
const elements = ref([22, 12, 17]);
```

3.3 Itérer sur les propriétés d'un objet

Vous pouvez également utiliser v-for pour itérer sur les propriétés d'un objet. Si vous utilisez un seul alias, comme dans l'exemple suivant valeur, vous aurez accès aux valeurs de l'objet : `v-for="valeur in objet"`.

- Si vous en utilisez deux, vous aurez accès aux valeurs et aux clés : (valeur, clé).
- Si vous en utilisez trois, vous aurez également accès aux index : (valeur, clé, index).

```
const monObjet = reactive({
  titre: 'Boucler sur des listes en Vue',
  auteur: 'Dyma',
  date: '2023-04-10'
})
```

Et côté template :

```
<ul>
  <li v-for="valeur in monObjet">
    {{ valeur }}
  </li>
</ul>
```

Vous pouvez également imbriquer un v-for dans un autre v-for et utiliser l'affectation par décomposition JavaScript :

```
<template>
  <ul>
    <li v-for="({ prenom, notes }, index) of utilisateurs">
      {{ `${index} : ${prenom}` }}
      <span v-for="note in notes">{{ note }} / </span>
    </li>
  </ul>
</template>

<script setup lang="ts">
import { reactive } from 'vue';

const utilisateurs = reactive([
  { prenom: 'Jean', notes: [14, 15, 18] },
  { prenom: 'Paul', notes: [12, 10, 8] },
  { prenom: 'Pierre', notes: [3, 15, 18] },
```

```
]);
</script>
```

N'hésitez pas à revoir l'affectation par décomposition dans la formation JavaScript si ce n'est pas clair pour vous.

3.4 Itérer sur un nombre

Avec v-for vous pouvez également itérer sur un nombre pour créer un intervalle (range) :

```
<span v-for="n in 10">{{ n }}</span>
```

A noter que la première itération commence à 1 et non pas 0 !.

3.5 Utilisation d'un élément template

Comme pour la directive v-if, vous pouvez utiliser l'élément invisible template pour itérer sur plusieurs éléments HTML de même niveau :

```
<ul>
  <template v-for="elem in elements">
    <li>{{ elem }}</li>
    <li> test </li>
  </template>
</ul>
```

4 Directive v-for, détection des changements et combinaison avec v-if

4.1 Utilisation combinée des directives v-if et de v-for

Lors d'une utilisation combinée sur un même élément HTML, v-if est prioritaire sur v-for. Concrètement cela veut dire que v-if n'a pas accès aux variables fournies par v-for (c'est-à-dire aux éléments itérés). Pour cette raison, Vue.js recommande officiellement de ne pas utiliser v-if et v-for sur un même élément. Si vous avez besoin de les cumuler, utilisez par exemple un élément invisible template :

```
<template v-for="todo in todos">
  <li v-if="!todo.done">
    {{ todo.name }}
  </li>
</template>
```

Ici seules les todos dont la propriété done vaut `false` seront affichées.

4.2 Optimisation des performances

Par défaut, lorsque Vue.js met à jour une liste d'éléments utilisant v-for, il utilise une stratégie de mise à jour des valeurs.

Autrement dit, Vue.js ne va pas effectuer des manipulations du DOM coûteuses en performance en déplaçant les éléments du DOM. Il va mettre à jour la valeur des éléments du DOM sans le déplacer.

Par défaut, Vue utilise l'index de l'élément pour savoir si la valeur est affichée au bon index. Cette stratégie par défaut est efficace, mais ne convient que lorsque le rendu de la liste ne repose pas sur l'état de composants enfants ou sur l'état temporaire du DOM (par exemple sur la valeur de champs que l'utilisateur peut manipuler). Dans ces cas là, il faut utiliser l'attribut key pour donner un identifiant invariant à chaque élément afin que Vue.js puisse correctement suivre et mettre à jour les bons éléments sur le DOM :

```
<div v-for="elem in elements" :key="elem.id">
</div>
```

Le plus souvent on utilise un identifiant unique provenant d'une base de données.Mais cela peut être toute autre propriété, il faut simplement qu'elle soit unique :

```
<template v-for="todo in todos" :key="todo.name">
  <li>{{ todo.name }}</li>
</template>
```

Il est recommandé d'utiliser l'attribut key dès que vous avez une propriété unique pour différencier vos éléments. Les performances seront meilleures pour le rendu et les mises à jour des listes sur le DOM.

Pour comprendre, remarquez dans cet exemple comment les bons éléments sont supprimés du DOM sans avoir à réafficher tous les autres éléments, c'est une très bonne optimisation :

4.3 La détection des changements des tableaux

En JavaScript, les tableaux et les objets sont passés par référence et non par valeur, il peut de ce fait avoir un problème de détection des changements dans les fonctions et les frameworks JavaScript.

Pour y remédier, Vue.js ajoute une surcouche aux principales méthodes de mutation d'un tableau afin qu'elles déclenchent la mise à jour du template suite à l'utilisation d'une de ces méthodes sur un tableau.

Les méthodes pour lesquelles une surcouche est présente sont :

- push()
- pop()
- shift()
- unshift()
- splice()

- sort()
- reverse()

L'exemple précédent fonctionne parfaitement car nous utilisons splice() qui déclenche une mise à jour lorsque nous supprimons un élément de la liste. Certaines méthodes ne modifient pas les tableaux mais retournent de nouveaux tableaux : ce sont toutes les méthodes de programmation fonctionnelle en JavaScript, par exemple map(), filter(), concat(), slice() etc.

Dans ce cas, il faut réassigner la valeur du tableau pour que Vue.js puisse détecter le changement.

Dans cet exemple, cela ne fonctionne pas lorsque vous supprimez une tâche car Vue.js ne peut pas détecter les changements : Décommentez la ligne dans la fonction `suppTodo()` pour que cela fonctionne.

5 Les directives v-once et v-memo

5.1 La directive v-once

La directive v-once permet de rendre un élément une seule fois et de ne plus le mettre à jour ensuite. Autrement dit, après le premier rendu par Vue.js, les éléments sur lesquels sont appliqués la directive v-once sont considérés comme étant du contenu statique. Cette directive permet d'optimiser les performances, par exemple sur un composant contenant des composants étant mis à jour très souvent et d'autres composants n'ayant pas besoin d'être mis à jour. Par exemple :

```
<ul>
  <li v-for="i in list" v-once>{{i}}</li>
</ul>
<enfant1></enfant1>
```

Ici nous supposons que cette liste ne sera pas mise à jour après l'affichage, mais qu'un composant enfant est mis à jour fréquemment. Dans ce cas, v-once permet de ne pas devoir mettre à jour la liste à chaque fois.

5.2 La directive v-memo

La directive v-memo est assez proche de la logique de computed() : elle permet de suivre des dépendances et de mettre à jour un ou plusieurs éléments du DOM uniquement si ces dépendances sont mises à jour.

```
<div v-memo="[valeurA, valeurB, valeurC]">
  ...
</div>
```

Ici, la div et tous les éléments imbriqués, ne seront mis à jour que si valeurA, valeurB et/ou valeurC sont modifiées.