Rendre les composants réactifs

Ibrahim ALAME

14/02/2023

1 Introduction à la réactivité

1.1 La réactivité avec Vue.js

Avec Vue.js les composants sont réactifs. Cela signifie que lorsque vous modifiez des objets JavaScript utilisés dans le template, Vue.js va automatiquement mettre à jour la vue . Vue.js met à disposition une API permettant de créer des objets réactifs et de mettre à jour automatiquement le DOM lorsqu'ils sont modifiés. C'est ce que nous allons étudier dans ce chapitre.

1.2 Fonctionnement de la réactivité avec Vue.js

Il n'est pas possible de suivre directement l'accès et l'écriture dans à des variables en JavaScript. Ce qu'il est possible de faire est d'intercepter l'accès et l'écriture à des objets JavaScript. A partir de la version 3 de Vue.js, des **proxys** JavaScript sont utilisés pour réagir à l'accès et à l'écriture d'objets en utilisant des getters et des setters.

Le **Proxy** est un objet JavaScript qui réagit à différents événements, comme par exemple le changement de valeur d'une propriété. Vous n'aurez pas à utiliser vous-même directement des proxies.

2 La fonction reactive()

2.1 La fonction reactive()

Avec Vue.js, nous pouvons créer des objets, tableaux, Maps et Sets réactifs avec la fonction `reactive()`. Par exemple :

```
import { reactive } from 'vue'

const state = reactive({ compteur: 0 });
```

Nous avons créé un objet réactif avec une propriété compteur initialisée à 0. Par convention, on utilise state pour état pour stocker les objets réactifs d'un composant. *L'état state d'un composant est l'état des données de sa partie logique et qui peuvent changer au cours du temps en réaction à des changements.* Comme nous l'avons vu dans la leçon précédente, la fonction `reactive()` va en

fait utiliser des `Proxies` JavaScript pour pouvoir réagir aux accès et aux modifications des objets réactifs.

Retenez bien que seul le proxy est réactif (l'objet retourné par la fonction `reactive()`. Si vous modifiez un objet directement sans passer par la fonction `reactive()`, il ne se passera rien. C'est pour cette raison qu'on déclare souvent un objet state avec tous les objets réactifs du composant. Une fois un objet rendu réactif, les modifications seront automatiquement propagées au template et rendues par Vue.js !

2.2 Mise à jour du DOM

Comme nous l'avons vu, lorsque vous modifiez un ou plusieurs objets réactifs, le DOM est mis à jour automatiquement. Ces mises à jour sont appliquées à chaque tick (le nom vient simplement du déplacement de la trotteuse d'une montre), c'est-à-dire à chaque nouveau cycle de mise à jour des composants.

Ce système est mis en place par Vue.js pour s'assurer que chaque composant n'est mis à jour qu'une seule fois pendant chaque cycle de mise à jour, peu importe le nombre de modifications effectuées. Cela permet de meilleures performances. Il est possible d'exécuter une action spécifique à chaque mise à jour du DOM en utilisant la fonction `nextTick()` et en lui passant une fonction de rappel :

```
import { nextTick } from 'vue'

function incCompteur() {
  state.compteur++
  nextTick(() => {
    // le DOM a été mis à jour
  })
}
```

Ici, dans la fonction passée à `nextTick()` nous sommes sûr que le DOM a été mis à jour et que la nouvelle valeur du compteur est bien affichée. Essayez l'exemple interactif pour bien comprendre la réactivité avant de passer à la leçon suivante.

2.3 Limitation de la fonction reactive()

La fonction reactive() ne peut pas rendre une valeur primitive réactive :

```
const state = reactive({ compteur: 0 })

let n = state.compteur
n++
```

Cela ne fonctionnera pas car JavaScript passe les valeurs primitives par valeur et non par référence. Cela signifie qu'avec ce code, n contiendra la valeur 0 et non la référence vers la propriété reactive compteur. `n++` modifiera donc seulement n et pas `state.compteur` et aucune mise à jour

ne sera déclenchée ! Nous verrons dans la prochaine leçon qu'il faut dans ce cas utiliser la fonction `ref()`.

3 La fonction ref()

3.1 La fonction ref()

Vue.js met à disposition la fonction ref qui permet de créer des variables réactives contenant tout type de valeur, y compris des primitives. Par exemple

```
import { ref } from 'vue'

const compteur = ref(0);
```

ref() prend en la valeur en argument et va en fait tout simplement créer un objet contenant une propriété `value` avec la valeur passée. Cela permet de contourner le problème que nous avions vu dans la leçon précédente : compteur contient donc bien un objet et celui-ci est passé par référence. Nous pouvons donc le modifier et Vue.js pourra intercepter les changements pour effectuer les mises à jour nécessaires. Pour modifier la valeur, il faut modifier la propriété value automatiquement créée :

```
compteur.value++
```

Si vous modifiez, compteur il ne se passera rien. En revanche, côté template, vous n'avez pas besoin d'utiliser `compteur.value`. Vue.js va automatiquement accéder à la propriété `.value` pour les variables réactives créées avec ref().

3.2 Accès automatique aux propriétés réactives

Lorsque vous créez une propriété réactive avec ref() sur un objet réactif (créé avec `reactive()` , Vue.js va automatiquement utiliser le `.value` pour vous lorsque vous essayez de modifier ou d'accéder à cette propriété :

```
import {ref, reactive} from 'vue'
let compteur = ref(0);
const state = reactive({
  compteur
})

console.log(state.compteur) // 0
state.compteur = 2
console.log(state.compteur) // 2
```

Ici nous n'avons donc pas besoin de faire `state.compteur.value = 2.` mais directement `state.compteur = 2.` Vue.js s'en charge pour nous.

3.3 Rappels sur l'inférence de type TypeScript

Comme nous l'avons vu, l'inférence de type permet de déduire le type de vos assignations. Par exemple :

```
let x = 42;
```

Par l'inférence de type, TypeScript déclare implicitement que la variable est de type `number`, vous ne pourrez donc pas lui assigner une chaîne de caractères plus tard dans votre code. Ce type d'inférence se produit lors des assignations de variable, lors de la déclaration de paramètres par défaut pour des fonctions et pour les retours de fonctions.

C'est cette fonctionnalité qui pousse les débutants à mettre any partout car le compilateur va se plaindre d'assignation successible de types différents sans que vous ayez indiqué que la variable pouvait avoir plusieurs types.

4 La fonction computed()

4.1 La fonction computed()

Nous avions vu qu'il était possible de mettre toute expression JavaScript côté template, par exemple :

```
<p>Est majeur :</p>
<span>{{ personne.age >= 18 ? 'Oui' : 'Non' }}</span>
```

Cependant, dès qu'il y a de la logique incluant des données réactives il est recommandé d'utiliser des propriétés calculées (computed properties). Voici un exemple de propriété calculée utilisant la fonction `computed()` :

```
<script setup lang="ts">
import { reactive, computed } from 'vue'

const personne = reactive({
  name: 'Jean Dupont',
  age: 27
})

const estMajeur = computed(() => personne.age >= 18 ? 'Oui' : 'Non' );
</script>
```

La fonction `computed()` attend en argument une fonction de rappel qui retourne la valeur calculée. La fonction retourne une référence calculée (computed ref). Il est possible, comme pour toutes les refs, d'accéder à la valeur de la propriété en utilisant `value`. Ici par exemple en faisant `estMajeur.value`. Mais comme pour les autres refs, l'accès à la propriété `value` est automatique côté template. C'est pour cette raison que nous pouvons directement faire :

```
<span>{{ estMajeur }}</span>
```

Une propriété calculée suit automatiquement le changement de ses dépendances réactives. Cela signifie que Vue.js sait quelles propriétés réactives sont utilisées pour déterminer la valeur de la propriété calculée.

Lorsque l'une des valeurs des propriétés réactives changent, Vue.js déclenche le calcul de la propriété calculée automatiquement. Lorsqu'aucune dépendance change, Vue.js retourne la valeur pré-calculée qui est gardée en cache. Ce mécanisme permet de maintenir la réactivité des propriétés calculées tout en optimisant les performances.

A noter qu'il ne faut pas effectuer de modification du DOM ou effectuer des actions asynchrones dans une propriété calculée, elles ne sont pas prévues pour cela !

4.2 Les propriétés calculées avec accès en écriture

Par défaut, les propriétés calculées prennent un seul argument : une fonction `getter` qui va calculer la propriété à retourner. Il est cependant possible de créer une propriété calculée avec un accès en écriture. Cette fonctionnalité est vraiment avancée, et vous pouvez donc sauter cette partie et y revenir plus tard lorsque vous en aurez besoin. Pour créer une telle propriété calculée, il faut lui passer en argument un objet avec une méthode `get()` (pour l'accès) et une méthode `set()` (pour l'écriture) :

```
<script setup>
import { ref, computed } from 'vue'

const prenom = ref('Jean')
const nom = ref('Dupont')

const nomComplet = computed({
  get() {
    return `${prenom.value} ${nom.value}`
  },
  set(nouvelleValeur) {
    [prenom.value, nom.value] = nouvelleValeur.split(' ')
  }
})
nomComplet.value = 'Francis Lapierre'
</script>
```

Avec cette fonctionnalité vous pouvez ainsi modifier `nomComplet`. Encore une fois, vous rencontrerez rarement cette fonctionnalité.

5 La directive v-model

5.1 La directive v-model

La directive v-model permet de créer une liaison dans les deux sens (entre partie template et partie script). Elle peut être utilisée sur :

- les input
- les select
- les textarea
- les composants

Prenons un exemple pour lier une propriété à la valeur d'un input :

```
<input
:value="text"
@input="event => text = event.target.value">
```

Ici nous lions la valeur du champ avec la directive v-bind à la propriété text (sens `script` vers `template`). Et nous lions l'événement input avec la directive v-on pour que la valeur de la propriété `text` soit égale à la valeur du champ lorsque l'utilisateur le modifie (événement `input`). C'est dans le sens `template` vers `script`.

Étant donné que cette liaison bidirectionnelle est très courante, Vue.js mets à disposition la directive v-model qui permet de faire exactement la même chose mais avec une syntaxe beaucoup plus concise et élégante :

```
<input v-model="text">
```

Nous la verrons plus en détail dans le chapitre sur les formulaires.

6 La fonction watch()

6.1 Les Watchers

Lorsque nous avons besoin d'effectuer des actions asynchrones ou des effets de bord (side effects) lorsque des propriétés réactives changent, il n'est pas possible d'utiliser des propriétés calculées. Par exemple, lorsque nous avons besoin d'effectuer une requête HTTP pour récupérer des données supplémentaires, ou encore lorsque nous avons besoin de modifier le DOM.

La fonction watch() permet de déclencher l'exécution d'une fonction si une ou plusieurs propriétés réactives enregistrées comme dépendances sont modifiées.

6.2 Les sources ou dépendances de la fonction watch()

Le premier argument est le ou les dépendances à enregistrer, elles sont appelées les sources de la fonction watch() : Cela peut être une seule propriété réactive, enregistrée en dépendance :

```
watch(prop, (nouvelleVal) => {
  console.log(`prop vaut ${nouvelleVal}`)
})
```

Cela peut être une fonction getter comme pour une propriété calculée :

```
watch(
  () => prop1.value + prop2.value,
  (somme) => {
    console.log(`La somme de x + y est ${somme}`)
  }
)
```

Ou encore un tableau de plusieurs propriétés réactives :

```
watch([prop1, () => prop2.value], ([nouvelleValProp1, nouvelleValProp2]) => {
  console.log(`prop1 vaut ${nouvelleValProp1} et prop2 vaut ${nouvelleValProp2}`)
})
```

Attention à bien passer une propriété réactive et non une primitive ! Pour les mêmes raisons qu'évoquées dans les leçons précédentes, il faut veiller à passer une référence et non une valeur ! Par exemple, cela ne fonctionnera pas car nous passons la valeur 0 en source :

```
const obj = reactive({ compteur: 0 })

watch(obj.compteur, (compteur) => {
  console.log(`Le compteur vaut: ${compteur}`);
})
```

Dans ce cas utilisez un getter car les fonctions sont passées par référence et ce sera la fonction getter qui sera enregistrée en source et non plus la valeur primitive 0 :

```
watch(
  () => obj.compteur,
  (compteur) => {
    console.log(`Le compteur vaut: ${compteur}`);
  }
)
```

6.3 La fonction de rappel de la fonction watch()

La fonction de rappel est la fonction passée en deuxième argument. Elle reçoit la nouvelle valeur de la source en argument. Si un tableau de sources est passé en premier argument, alors la fonction de rappel recevra un tableau de nouvelles valeurs dans le même ordre :

```
watch([prop1, prop2], ([nouvelleValProp1, nouvelleValProp2]) => {
  console.log(`prop1 vaut ${nouvelleValProp1} et prop2 vaut ${nouvelleValProp2}`)
})
```

Passons maintenant aux exemples.

6.4 Deuxième exemple

Prenons un second exemple avec une requête HTTP.

6.4.1 Commençons par la partie template

Ici nous utilisons une double liaison de données avec la directive v-model pour récupérer la question posée par l'utilisateur dans le champ. Nous affichons la réponse avec la notation raccourcie de la directive v-text et avec la directive v-bind liée à la propriété src d'une image.

6.4.2 Pour la partie scripte

Dans la partie script nous avons simplement une variable réactive contenant la question posée et qui va être utilisée comme source de la fonction watch(). Ainsi, dès que la valeur de la variable question changera, la fonction de rappel de watch() sera exécutée. Nous avons également un objet réactif contenant le texte de la réponse (`yes` ou `no`) et l'image GIF correspondante.

Analysons la fonction watch() :

```
watch(question, async (newQuestion, oldQuestion) => {
  if (newQuestion.includes('?')) {
    reponse.res = 'Requête en cours...';
    try {
      const res = await fetch('https://yesno.wtf/api');
      const resJSON = await res.json();
      reponse.res = resJSON.answer;
      reponse.image = resJSON.image;
    } catch (error) {
      reponse.res = `Erreur : ${error}`;
    }
  }
});
```

La source de la fonction watch est la variable réactive question. La fonction de rappel reçoit la nouvelle valeur de la propriété et l'ancienne. Si la nouvelle valeur de question contient un point d'interrogation, nous considérons que la question posée par l'utilisateur est complète et nous envoyons une requête HTTP à une API.

L'API retourne aléatoirement `yes` ou `no` avec une image GIF amusante. Nous assignons le retour de l'API à notre objet réactif reponse. Dès que l'une des propriétés de reponse change, alors Vue

met à jour le template automatiquement et affiche la réponse et l'image s'il n'y a pas d'erreur. S'il y a une erreur, il affichera simplement la valeur de l'erreur.

7 La fonction watchEffect()

7.1 La fonction watchEffect()

La fonction watch() est paresseuse (lazy), cela signifie que la fonction de rappel passée en deuxième argument ne sera exécutée que si les sources sont modifiées. Dans certains cas, vous voulez que la fonction de rappel soit exécutée dès le chargement du composant, par exemple pour récupérer des données initiales.

La fonction fetch() permet d'exécuter immédiatement la fonction de rappel, appelée effet, et d'enregistrer automatiquement toutes les propriétés réactives utilisées comme étant des sources. Lorsque ces propriétés réactives enregistrées comme dépendances, seront modifiées, la fonction de rappel de watchEffect() sera automatiquement ré-exécutée.

7.2 Différences entre watch et watchEffect()

La fonction watch() : n'enregistre en sources (ou dépendances) que celles explicitement passées en premier argument. Elle n'enregistre pas automatiquement les propriétés réactives utilisées dans la fonction de rappel. La fonction de rappel ne s'exécute que si au moins une source change.

La fonction watchEffect() : enregistre automatiquement toutes les propriétés réactives, utilisées dans la fonction de rappel passée en premier argument, comme des dépendances. La fonction de rappel est exécutée une première fois dès le chargement puis à chaque fois qu'une des propriétés réactives utilisées dans la fonction de rappel sont modifiées.

7.3 Ordre d'exécution

Lorsque vous modifiez des propriétés réactives, Vue.js peut devoir modifier le DOM (si les propriétés réactives sont utilisées côté `template`) et devoir exécuter vos watchers (si les propriétés réactives sont enregistrées comme des sources / dépendances dans des watch() et/ou des watchEffect()).

Par défaut, Vue.js va d'abord exécuter vos watchers avant de mettre à jour le DOM. Cela veut dire que dans les fonctions de rappel de vos watchers, le DOM sera dans l'état avant la modification des propriétés réactives. Il faut y penser si vous utilisez le DOM dans les watchers.

Si vous voulez que des watchers soient exécutées après la mise à jour du DOM, il suffit d'utiliser la propriété flush: 'post' :

```
watch(source, fonctionDeRappel, {
  flush: 'post'
})

watchEffect(fonctionDeRappel, {
  flush: 'post'
})
```