Composants natifs

Ibrahim ALAME

28 décembre 2023

1 Introduction aux composants natifs

1.1 Les composants natifs

Les composants natifs sont des composants mis à disposition par Vue pour certains effets avancés comme par exemple la mise en cache, les transitions etc. Dans ce chapitre nous allons voir les composants natifs suivants :

- `<Component>`: qui permet de créer des composants dynamiques.
- `<KeepAlive>`: qui permet une mise en cache des composants.
- `<Teleport>`: qui permet de "téléporter" une partie d'un template d'un composant à l'extérieur de l'application Vue.
- `<Suspense>`: qui permet de gérer les dépendances asynchrones d'un arbre de composant.

1.2 Les composants dynamiques

Il est parfois utile de pouvoir changer dynamiquement de composant. Par exemple pour une interface avec des onglets. Avec le composant natif Component, et l'attribut is à lié une variable contenant le nom d'un composant, il est facile de mettre en place une telle interface :

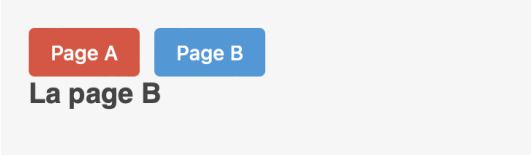
```
<Component :is="onglet" />
```

Cette fonctionnalité est assez rarement utilisée avec le Router que nous verrons plus tard, nous ne détaillerons donc pas plus.

1.3 Exemple

```
<template>
  <div class="p-20">
    <button @click="selectedComponant = 'PageA'" class="btn btn-danger mr-10">
      Page A
    </button>
    <button @click="selectedComponant = 'PageB'" class="btn btn-primary mr-10">
      Page B
    </button>
    <Component :is="composants[selectedComponant]" />
  </div>
</template>

<script setup lang="ts">
```

```
import PageA from './PageA.vue';
import PageB from './PageB.vue';
import { ref, type Component as C } from 'vue';

const composants: { [s:string]: C } = {
  PageA,
  PageB,
};

const selectedComponant = ref('PageA');
</script>

<style lang="scss">
@import './assets/scss/base.scss';
</style>
```



2 Le composant KeepAlive

2.1 Le composant KeepAlive

Le composant natif `<KeepAlive>` permet de mettre en cache conditionnellement des composants lorsque l'on change dynamiquement de composant . En effet, par défaut une instance d'un composant Vue est démontée (unmounted) lorsqu'il n'est plus affiché. Cela entraîne la perte de tout état du composant. Autrement dit, lorsqu'un composant n'est plus affiché, toutes les variables, notamment réactives, perdent leurs valeurs.

Lorsqu'un composant mis en cache avec `<KeepAlive>` est retiré du DOM, il est mis dans l'état désactivé (`deactivated`) au lieu d'être démonté (`unmounted`).

Lorsque le composant est remis sur le DOM aura alors l'état est activé (`activated`). Il suffit d'imbriquer, le ou les composants à l'intérieur du composant `<KeepAlive>`, par exemple :

```
<KeepAlive>
  <Component :is="onglet" />
</KeepAlive>
```

2.2 Définir les composants à maintenir en vie avec includes et exclude

Par défaut, `<KeepAlive>` mis en cache tous les composants imbriqués. Il est possible de modifier ce comportement avec les props include et exclude. Pour utiliser cette fonctionnalité, il faut préalablement que les composants soient nommés en utilisant une deuxième balise script et en utilisant la propriété `name`, comme ici :

```
<script lang="ts">
export default {
  name: 'PageB',
};
</script>
```

Il suffit ensuite de préciser les composants à mettre en cache en utilisant include ou exclude :

```
<KeepAlive include="PageB">
  <Component :is="composants[selectedComponant]" />
</KeepAlive>
```

Vous pouvez utiliser des Regex :

```
<KeepAlive include="/Page*/">
  <Component :is="composants[selectedComponant]" />
</KeepAlive>
```

Passer plusieurs noms en utilisant des virgules :

```
<KeepAlive exclude="PageA, PageC">
  <Component :is="composants[selectedComponant]" />
</KeepAlive>
```

Ou un tableau :

```
<KeepAlive exclude="['PageA', 'PageC']">
  <Component :is="composants[selectedComponant]" />
</KeepAlive>
```

2.3 Définir le maximum de composants mis en cache

Si vous avez un très grand nombre de composants susceptibles d'être mis en cache, vous pouvez limiter le nombre maximal de composants à mettre en cache avec la props max. Il faut utiliser v-bind car nous passons un nombre (comme nous l'avions vu, sans v-bind les nombres sont passés comme des chaînes de caractères) :

```
<KeepAlive :max="10">
  <Component :is="activeComponent" />
</KeepAlive>
```

2.4 Les hooks utilisables

Il est possible d'utiliser les `hooks:` activated() et deactivated().

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

onActivated(() => {
  // appelé à chaque activation du composant
})

onDeactivated(() => {
  // appelé à chaque désactivation du composant
})
</script>
```

2.5 Exemple

On reprend l'exemple précédant et on modifie `App.vue` de la façon suivante :

```
<KeepAlive :max="5">
      <Component :is="composants[selectedComponant]" />
</KeepAlive>
```

On modéfie `PageA.vue` et `PageB.vue` :

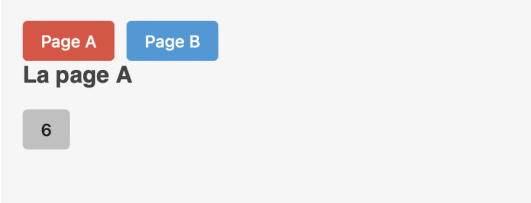
```
<template>
  <h3>La page A</h3>
  <br />
  <button class="btn mb-20" @click="count++">{{ count }}</button>
</template>

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

const count = ref(0);

onDeactivated(() => {
  count.value--;
});
</script>

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

puis on fait

```
<template>
  <h3>La page B</h3>
```

```
  <br />
  <span>Utilisateur : {{ username }}</span>
  <input v-model="username" />
</template>

<script lang="ts">
export default {
  name: 'PageB',
};
</script>

<script setup lang="ts">
import { ref, onActivated } from 'vue';
const username = ref('');

onActivated(() => {
  username.value += '!';
});
</script>

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

On teste le resultat :



3 Le composant Téléport

3.1 Le composant natif Teleport

`<Teleport>` est un composant natif permettant de *téléporter* une partie du template un composant sur un noeud du DOM qui existe en dehors de la branche du composant, ou même en dehors de l'application Vue.

Le scénario le plus commun est la modale : une partie d'un composant doit être affichée ailleurs sur le DOM, à l'extérieur de l'application. Nous voulons en effet que le bouton de la modale et la modale soient dans le même composant, car ils sont tous deux liés à l'état ouvert/fermé de la modale. Mais cela signifie que le modal sera rendu à côté du bouton, profondément imbriquée dans l'arbre des composants de l'application Vue. Cela peut créer des problèmes lors du positionnement de la modale avec du CSS. L'utilisation est très simple, il suffit d'imbriquer la partie du template "téléporter" dans des balises `<Teleport>` et d'indiquer avec le props to sélecteur CSS ciblé :

```
<button @click="open = true">Ouvrir</button>

<Teleport to="body">
```

```
  <div v-if="open" class="modal">
    <p>Contenu de la modale</p>
    <button @click="open = false">Fermer</button>
  </div>
</Teleport>
```

Il est possible de désactiver la téléportation du template en fonction de la valeur d'une variable en utilisant disabled :

```
<Teleport :disabled="isMobile">
  ...
</Teleport>
```

3.2 Exemple

App.vue

```
<template>
  <Page />
</template>

<script setup lang="ts">
import Page from './Page.vue';
</script>

<style lang="scss">
@import './assets/scss/base.scss';
</style>
```

Page.vue

```
<template>
  <div class="p-20 container">
    <h3>PAGE</h3>
    <Modal />
  </div>
</template>

<script setup lang="ts">
import Modal from './Modal.vue';
</script>

<style lang="scss">
.container {
  position: relative;
```

```
}
</style>
```

Modal.vue

```
<template>
  <button @click="open = true" class="btn btn-primary">
    Confirmer l'achat
  </button>
  <Teleport to="body">
    <div
      v-if="open"
      @click="open = false"
      class="calc d-flex flex-row justify-content-center align-items-center"
    >
      <div @click.stop class="modal-container">
        <h3>Confirmation de votre commande</h3>
        <ul>
          <li>Du contenu</li>
          <li>Du contenu</li>
          <li>Du contenu</li>
        </ul>
        <button @click.stop="open = false" class="btn btn-danger">
          Confirmer la commande
        </button>
      </div>
    </div>
  </Teleport>
</template>

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

const open = ref(false);
</script>

<style lang="scss">
.calc {
  position: absolute;
  top: 0;
  width: 100%;
  height: 100vh;
  background-color: rgba(0, 0, 0, 0.5);
  backdrop-filter: blur(2px);
}

.modal-container {
  background-color: white;
  border-radius: var(--border-radius);
  border: var(--border);
```

```
  padding: 30px;
}
</style>
```

4 Le composant Suspense

4.1 Le composant `<Suspense>`

`<Suspense>` est un composant natif permettant de gérer les dépendances asynchrones d'une branche de composants . Il permet d'afficher un template spécifique pendant le chargement des différentes ressources asynchrones. Les ressources asynchrones sont la plupart du temps des requêtes HTTP pour récupérer des données depuis une API. Il peut y avoir un composant qui a besoin d'une liste d'utilisateurs, un autre qui a besoin de favoris, un troisième de préférences utilisateur avec des requêtes différentes.

Sans Suspense il faut que chaque composant gère l'affichage d'un composant de chargement, des éventuelles erreurs etc. Cela permet de simplifier grandement la gestion des ressources asynchrones.

Le composant Suspense peut gérer deux types de dépendances asynchrones : les composants qui utilisent des await au premier niveau (c'est-à-dire pas imbriqué dans une fonction, directement dans les balises script) et les composants asynchrones .

Pour les composants avec await au premier niveau, le cas le plus courant est une requête HTTP :

```
<script setup>
const res = await fetch(...)
const posts = await res.json()
</script>
```

4.2 État de chargement

Le composant Suspense à deux slots nommés : #default et #fallback . Chaque slot permet l'utilisation d'un unique enfant : que ce soit un composant ou un élément HTML :

```
<Suspense>
  <Dashboard />
  <template #fallback>
    Chargement...
  </template>
</Suspense>
```

Lors du rendu initial, Suspense va afficher le slot par défaut (ici le composant Dashbord, car le slot default est l'élément qui n'a pas de nom). S'il rencontre une dépendance asynchrone, Suspense va entrer dans l'état en attente (pending state). Pendant toute la durée de cet état, il affichera le slot fallback, ici `<template #fallback>`.

Lorsque tous les contenus asynchrones sont chargés ou qu'il n'y a aucune dépendance asynchrone, alors Suspense à l'état résolu (resolved state) et le slot par défaut est affiché.

4.3 Les événements émis

Le composant Suspense émet trois événements :

- pending : émis lorsqu'il entre dans l'état pendinget donc qu'au moins une dépendance asynchrone a été détectée.
- resolve : émis lorsqu'aucune dépendance asynchrone ne reste à charger.
- fallback : émis lorsque le composant Suspense affiche le slot fallbacklors du chargement des dépendances asynchrones.

4.4 Option timeout

Lorsque le composant Suspense est dans l'état résolu, il ne retournera dans l'état en attente que si un composant dynamique est modifié et que la propriété timeout est définie. En effet, par défaut, Suspense affichera le composant par défaut et non le fallback pendant le chargement des dépendances asynchrones dans ce cas.

Il est possible de passer un nombre de millisecondes à timeout pour que Suspense affiche le fallback pendant le chargement, si les dépendances ne sont pas résolues pendant ce laps de temps.

4.5 Exemple

App.vue

```
<template>
  <h2>{{ evenement }}</h2>
  <Suspense
    @pending="myEvent('PENDING')"
    @fallback="myEvent('FALLBACK')"
    @resolve="myEvent('RESOLVE')"
  >
    <LazyList />
    <template #fallback>
      <h1>Chargement...</h1>
    </template>
  </Suspense>
```

```
</template>

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

const evenement = ref('');
const LazyList = defineAsyncComponent(() => import('./Liste.vue'));

function myEvent(eventName: string) {
  console.log(eventName);
  evenement.value = eventName;
}
</script>

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

Liste.vue

```
<template>
  <div class="p-20">
    <ul>
      <li v-for="user of state.users">{{ user.name }}</li>
    </ul>
  </div>
</template>

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

const state = reactive<{ users: any[] }>({ users: [] });

// Délai de 3secondes grâce à l'option delay de restapi
state.users = await (
  await fetch('https://restapi.fr/api/vueusers?delay=3')
).json();
</script>

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

FALLBACK

Chargement...