Vue.js : Mixin, Composable et Plugin

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en programmation par contrainte (IA)
Ingénieur en génie logiciel

elmouelhi.achref@gmail.com

Plan

1. Mixin
2. Composable
3. Plugin

Vue.js

Mixin

- Objet **JavaScript**.
- Permettant la réutilisation du code par les composants **Vue.js**.
- Pouvant contenir toute option valide pour un composant : `data`, `methods`, `hooks`...

Vue.js

Limite

- Ne pouvant pas être paramétré
- Problème de conflits de nom en cas d'utilisation de plusieurs mixins
- Impossible de différencier les attributs du composant de ceux du mixin
- **Ne figure plus dans la documentation officielle de Vue.js**

Vue.js

Dans le composant `Produit.view`, supposons qu'on souhaite

- ajouter une partie quantité réservée pour chaque produit
- permettre à l'utilisateur d'incrémenter/décrémenter la quantité réservée de chaque produit

Vue.js

Dans le composant `Produit.view`, supposons qu'on souhaite

- ajouter une partie quantité réservée pour chaque produit
- permettre à l'utilisateur d'incrémenter/décrémenter la quantité réservée de chaque produit

Rappel

Même chose a été faite dans `CompteurView.vue`

Vue.js

Question

Faudrait-il dupliquer le code ?

Vue.js

Question

Faudrait-il dupliquer le code ?

Réponse : deux solutions possibles

1. déclarer `CompteurView.vue` comme enfant de `Produit.vue` : le `template` de `CompteurView.vue` n'est pas adapté à `Produit.vue`
2. déclarer le `script` de `CompteurView.vue` dans un fichier **JavaScript** (c'est ce qu'on appelle **mixin**) et le référencer dans `CompteurView.vue` et `Produit.vue`.

Vue.js

Commençons par créer un fichier `counter.js` dans `src/mixins` avec le code suivant

```
export default {
    data() {
        return {
            counter: { valeur: 0, etat: 'nul' }
        }
    },
    methods: {
        incrementer () {
            this.counter.valeur++
        },
        decrementer () {
            this.counter.valeur--
        }
    },
    updated() {
        if (this.counter.valeur > 0) {
            this.counter.etat = 'positif'
        } else if (this.counter.valeur < 0) {
            this.counter.etat = 'négatif'
        } else {
            this.counter.etat = 'nul'
        }
    }
}
```

Vue.js

Dans `CompteurView.vue`**, supprimons les parties** `data`, `methods` **et** `updated`

```
<template>
    <h1>Compteur : {{ counter.etat }}</h1>
    <button @click="decrementer">-</button>
    {{ counter.valeur }}
    <button @click="incrementer">+</button>
</template>

<script>
export default {
}
</script>
```

Vue.js

Importons et déclarons le `mixin`

```
<template>
    <h1>Compteur : {{ counter.etat }}</h1>
    <button @click="decrementer">-</button>
    {{ counter.valeur }}
    <button @click="incrementer">+</button>
</template>

<script>

import CounterMixin from '../mixins/counter';

export default {
    mixins: [CounterMixin]
}
</script>
```

Vue.js

Importons et déclarons le `mixin`

```
<template>
    <h1>Compteur : {{ counter.etat }}</h1>
    <button @click="decrementer">-</button>
    {{ counter.valeur }}
    <button @click="incrementer">+</button>
</template>

<script>

import CounterMixin from '../mixins/counter';

export default {
    mixins: [CounterMixin]
}
</script>
```

Vérifier que le compteur s'incrémente se se décrémente en cliquant sur les boutons respectifs.

Vue.js

Dans `template` de `Produit.vue`, préparons la place pour la quantité réservée qui sera gérée par le `mixin`

```
<template>
    <ul>
        <li v-mouvement.color="{color: 'white', bgColor: 'skyblue'}">
            {{ produit.nom }}
        </li>
        <li v-mouvement="{color: 'white', bgColor: 'pink'}">
            Quantité en stock : {{ produit.quantite }}
        </li>
        <li>
            Quantité réservée :
            <button @click="decrementer">-</button>
            {{ counter.valeur }}
            <button @click="incrementer">+</button>
        </li>
        <li>
            <PrixComponent :prix="produit.prix" />
        </li>
    </ul>
</template>
```

Dans `script` de `Produit.vue`, importons puis utilisons le `mixin`

```
<script>
import PrixComponent from './Prix.vue'
import CounterMixin from '../mixins/counter'

export default {
    name: 'ProduitComponent',
    mixins: [CounterMixin]
}
</script>

<script setup>
import { defineProps } from 'vue';

defineProps({
    produit: Object
})
</script>
```

Dans `script` de `Produit.vue`, importons puis utilisons le `mixin`

```
<script>
import PrixComponent from './Prix.vue'
import CounterMixin from '../mixins/counter'

export default {
    name: 'ProduitComponent',
    mixins: [CounterMixin]
}
</script>

<script setup>
import { defineProps } from 'vue';

defineProps({
    produit: Object
})
</script>
```

Vérifier que le compteur s'incrémente se se décrémente en cliquant sur les boutons respectifs.

Vue.js

Supposons qu'on souhaite

- définir un pas (le même pour l'incrémentation et la décrémentation)
- une borne max et min à ne pas dépasser par le compteur

Vue.js

Supposons qu'on souhaite

- définir un pas (le même pour l'incrémentation et la décrémentation)
- une borne max et min à ne pas dépasser par le compteur

Réponse : deux solutions possibles

1. Avec les mixins : envoyer les paramètres à chaque appel de méthode : impossible de configurer le mixin par rapport aux besoins du composant
2. Avec les composables : envoyer les paramètres une seule fois, au moment de l'instanciation du composable.

Vue.js

Composable

- Fonction **JavaScript** utilisant **Composition API** : paramétrable.
- Permettant la réutilisation du code par les composants **Vue.js**.
- Pouvant contenir toute option valide de **Composition API** : `ref`, `reactive`, `hooks`...

Vue.js

Commençons par créer un fichier `useCounter.js` (convention : nom préfixé par `use`) dans `src/composables` avec le code suivant

```
import { reactive, onUpdated } from 'vue';

export default function useCounter(pas = 2, min = -10, max = 10) {

    const compteur = { valeur: 0, etat: 'nul' }
    const counter = reactive(compteur)
    const incrementer = () => {
        if (counter.valeur < max) counter.valeur += pas
    }
    const decrementer = () => {
        if (counter.valeur > min) counter.valeur -= pas
    }
    onUpdated(() => {
        if (counter.valeur > 0) {
            counter.etat = 'positif'
        } else if (counter.valeur < 0) {
            counter.etat = 'négatif'
        } else {
            counter.etat = 'nul'
        }
    })
    return {
        counter,
        incrementer,
        decrementer
    }
}
```

Vue.js

Dans `script` de `CompteurView.vue`, il suffit d'importer puis d'instancier la fonction avec les paramètres de notre choix

```
<script setup>

import useCounter from '../composables/useCounter';

const { counter, incrementer, decrementer } = useCounter(2)

</script>
```

Vue.js

Même chose pour `Produit.vue`

```
<script>
import PrixComponent from './Prix.vue'

export default {
    name: 'ProduitComponent',
}
</script>

<script setup>
import { defineProps } from 'vue';
import useCounter from '../composables/useCounter';

defineProps({
    produit: Object
})

const { counter, incrementer, decrementer } = useCounter(1, 0, 10)
</script>
```

Vue.js

Plugin

- Objet **JavaScript**
- Permettant la réutilisation du code par les composants **Vue.js**
- Ayant accès à l'objet d'application `app`
- Pouvant contenir des `hooks`
- Déclaré avec le mot-clé `install`
- Importé globalement dans `main.js`

Vue.js

Exemple

Nous voudrons créer un plugin qui permet d'afficher un texte en italic

Vue.js

Dans `italic.plugin.js` (à créer dans `src/plugins`), commençons par définir la structure de notre plugin (`options` représentent les paramètres)

```
export default {
    install: (app, options) => {

    }
};
```

Vue.js

Dans `italic.plugin.js` (à créer dans `src/plugins`), commençons par définir la structure de notre plugin (`options` représentent les paramètres)

```
export default {
    install: (app, options) => {

    }
};
```


Définissons notre plugin

```
export default {
    install: (app) => {
        app.config.globalProperties.$italicHTML = (text) => {
            return `<i> text </i>`
        }
    }
};
```

Vue.js

Importons puis utilisons le plugin dans `main.js`

```
// les imports précédents

import ItalicPlugin from './plugins/italic.plugin'

createApp(App)
  .use(router)
  .use(ItalicPlugin)
  .mount('#app')
```

Vue.js

Importons puis utilisons le plugin dans `main.js`

```
// les imports précédents

import ItalicPlugin from './plugins/italic.plugin'

createApp(App)
  .use(router)
  .use(ItalicPlugin)
  .mount('#app')
```


Dans n'importe quel composant, nous pourrons maintenant écrire

```
<div v-html="$italicHTML('Hello')"></div>
```