Vue JS

Le Store avec VueX

Vuex est un gestionnaire d'état (« state management pattern») et une bibliothèque pour des applications Vue.js Il sert de zone de stockage de données centralisée pour tous les composants dans une application, avec des règles pour s'assurer que l'état ne puisse subir de mutations que d'une manière prévisible.

Un composant est toujours composé de différents éléments :

- L'état, qui est la source de vérité qui pilote votre application.
- La vue, qui est une réflexion déclarative de l'état
- Les actions, qui sont les façons possibles pour l'état de changer en réaction aux actions utilisateurs depuis la vue.

Voici une représentation extrêmement simple du concept de «flux de donnée unidirectionnel »

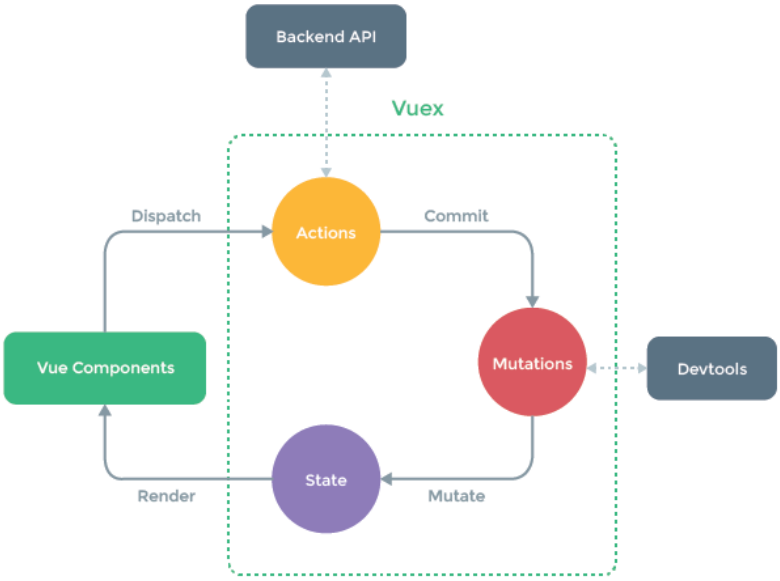

Figure 1: Vuex state management

Cependant, la simplicité s'évapore rapidement lorsque nous avons de multiples composants qui partagent un même état global.

Plusieurs vues peuvent dépendre de la même partie de l'état global. Des actions dans différentes vues peuvent avoir besoin de muter la même partie de l'état global.

- Pour le premier problème, passer des props peut être fastidieux pour les composants profondément imbriqués, et ça ne fonctionne tout simplement pas pour les composants d'un même parent.
- Pour le deuxième problème, on se retrouve souvent à se rabattre sur des solutions telles qu'accéder aux références d'instance du parent/enfant direct ou essayer de muter et synchroniser de multiples copies de l'état via des évènements
- Ces deux modèles sont fragiles et posent rapidement des problèmes de maintenabilité du code
- Alors pourquoi ne pas extraire l'état global partagé des composants, et le gérer dans un singleton global?
- De cette manière, notre arbre de composant devient une grosse « vue », et n'importe quel composant peut accéder à l'état global ou déclencher des actions, peu importe où il se trouve dans l'arbre !
- Au cœur de chaque application Vuex, il y a la zone de stockage (« store »)
- Un « store » est tout simplement un conteneur avec l'état («state ») de votre application

Il y a deux choses qui différencient un store Vuex d'un simple objet global

- Les stores Vuex sont réactifs

Quand les composants Vue y récupèrent l'état, ils se mettront à jour de façon réactive et efficace si l'état du store a changé.

- Vous ne pouvez pas muter directement l'état du store

La seule façon de modifier l'état d'un store est d'acter (« commit ») explicitement des mutations. Cela assure que chaque état laisse un enregistrement traçable, et permet à des outils de nous aider à mieux appréhender nos applications.

VueX

Tout d'abord, installons vuex

```
npm install vuex
```

Comme avec le routeur, nous allons gérer notre store dans un fichier à part

- Créer un dossier store et dedans un fichier index.js

Dans ce fichier, importez Vuejs ainsi que Vuex

```
// src: store/index.js
import Vue from 'vue'
import Vuex from 'vuex'
```

Puis appliquez Vuex à Vuejs et construisez l'objet store (vide pour le moment)

```
// src: store/index.js
Vue.use(Vuex);
export const store = new Vuex.Store({
})
```

Enfin, comme pour le router, appliquons la configuration à notre instance de Vuejs Dans le fichier main.js, ajoutez

```
// src: main.js
import { store } from './store'
```

Dans l'instanciation de vue, ajoutez le store

```
// src: main.js
store,
```

Et voilà ! Le store est configuré et existant sur l'ensemble de votre projet. Il est maintenant temps de concevoir différents fonctionnalités pour le store:

- stocker l'état
- modifier l'état
- récupérer l'état

Dans un premier temps, il faut définir la liste des propriétés du state. Dans le fichier `store/index.js`, ajoutez dans l'objet store.

```
// src: store/index.js
state: {
  name: 'toto'
},
```

Vous avez à présent une variable définie dans le store de votre application !

Maintenant il faut y accéder depuis tous les composants. Afin de simplifier la gestion du store, nous avons précédemment ajouté le store à notre instance de Vuejs.

Le store est donc accessible de partout. Mais il n'est pas possible d'accéder directement le state. Il faut concevoir un getter.

Toujours dans le fichier `store/index.js`

```
// src: store/index.js
getters: {
  nickName: state => state.name
}
```

Pour affichier le contenu, il suffit d'utiliser ce getter, dans n'importe quel composant.

- Dans le template :

```
{{ $store.getters.nickName }}
```

- Dans la logique

```
this.$store.getters.nickName
```

Enfin dernière étape, comment modifier les données. De la même manière, il nous faut définir une fonction pour modifier les données. Dans `store/index.js`

```
mutations: {
  // src: store/index.jss
  changeName(state, pseudo) {
    state.name = pseudo
  }
},
```

Notre store possède enfin une gestion complète du state de la variable name.Nous allons nous servir de cette fonction comme nous l'avons fait pour le getter.

- Dans la logique du composant

```
this.$store.commit('changeName', 'Merlin le druide')
```

Les stores ne permettent pas d'accéder aux fonctions de modifications directement. Il faut passer par la fonction `commit`. En premier paramètre, vous devez passer le nom de la fonction de mutation dans le store Dans l'exemple, `changeName`

** Exemple : exercices/vuejs-vuex **