Vue3 狀態管理

在開發大型 Vue 應用時，我們常常需要一個集中管理和維護應用各個部分狀態的機制。Vuex 就是為了解決這一問題而生的。Vuex 是一個專為 Vue.js 應用程序開發的狀態管理模式和庫。它利用 Vue.js 的細粒度資料響應機制，讓你可以更方便地管理應用的狀態。

Vuex 簡介

Vuex 作為一個狀態管理工具，主要解決了組件間的狀態共享問題，保證狀態在多個組件中同步。Vuex 中的狀態存儲是響應式的，當 Vue 組件從 store 中讀取狀態的時候，若狀態發生變化，相關組件也會相應地得到高效更新。

Vuex 的核心概念包括：

- **State**：定義了應用狀態的數據結構，可以在這裡設置默認的初始狀態。
- **Getters**：允許組件從 Store 中獲取數據，比如過濾出一些數據。
- **Mutations**：是唯一更改 Store 中狀態的方法，且不能處理異步操作。
- **Actions**：用於提交 mutation，而不是直接變更狀態，可以包含任意異步操作。
- **Modules**：允許將單一 Store 分割成多個 Store。

在 Vue3 中設置 Vuex

Vue3 推出了 Composition API，對於狀態管理也提供了新的使用方式。為了在 Vue3 中使用 Vuex，我們需要安裝 Vuex 的最新版本，這個版本被稱為 Vuex 4，它專為 Vue3 設計。

1. 安裝 Vuex：

```
npm install vuex@next --save
```

2. 創建一個 Vuex store。在你的 Vue 應用的 `src` 目錄下創建一個 `store` 目錄，並在這個目錄下創建 `index.js`：

```
import { createStore } from 'vuex';

export default createStore({
  state() {
    return {
      count: 0
    };
  },
  mutations: {
    increment(state) {
      state.count++;
    }
  },
  actions: {
    increment({ commit }) {
      commit('increment');
    }
  },
  getters: {
    doubleCount: state => state.count * 2,
  }
});
```

3. 將 Vuex store 加入到 Vue 應用中，在 `main.js` 中引入創建的 store，並加入到 Vue 應用實例中：

```
import { createApp } from 'vue';
import App from './App.vue';
import store from './store';

createApp(App).use(store).mount('#app');
```

使用 Vuex 管理狀態

現在 Vuex 已經設置好了，我們可以在組件中使用 Vuex 的狀態了。以下是一個簡單的例子，展示如何在組件中使用 Vuex 的 state 和 getters：

```
<template>
  <div>
    <p>{{ count }}</p>
    <p>{{ doubleCount }}</p>
    <button @click="increment">Increment</button>
  </div>
</template>

<script>
import { mapState, mapGetters, mapActions } from 'vuex';

export default {
  computed: {
    ...mapState(['count']), // 使用 mapState 輔助函數來映射 state
    ...mapGetters(['doubleCount']), // 使用 mapGetters 輔助函數來映射 getters
  },
  methods: {
    ...mapActions(['increment']), // 使用 mapActions 輔助函數來映射 actions
  },
}
</script>
```

這個例子展示了如何在組件中讀取 Vuex store 中的 `state` 和 `getters`，以及如何使用 `actions` 來提交更改。通過這種方式，我們可以在組件中輕鬆地管理和使用全局狀態。

組件中使用 Vuex 的注意事項

當在 Vue3 組件中使用 Vuex 時，有幾個重要的實踐要點需要注意：

1. **組件解耦**: 儘量保持組件與 Vuex store 的解耦，可以透過封裝更多的 getters 或 actions 來實現。這樣做不僅使得組件更容易重用，也讓狀態管理更加清晰。
2. **使用 Namespaced Modules**: 當應用規模變得很大時，將 store 分割成模塊是很有幫助的。使用 namespaced 模塊可以幫助你更好地組織代碼和避免命名衝突。
3. **異步操作**: Vuex 的 mutations 只能處理同步操作，對於異步邏輯（例如從 API 獲取數據），應該放在 actions 中處理。
4. **遵循 Flux 流程**: Vuex 的設計靈感來自 Flux 架構。按照其模式，數據應該保持單向流動，這意味著你應該避免在 mutations 或 actions 之外直接修改 state。
5. **使用 Composition API**: Vue3 引入的 Composition API 為狀態管理提供了更多的靈活性。對於需要在多個組件共享狀態的情況，考慮使用 Composition API 結合 Vuex 或者 Vue3 的 `reactive`、`ref` 等響應式API來管理。

通過這些最佳實踐，你可以在 Vue3 應用中更高效、更靈活地使用 Vuex 來管理和維護狀態，從而開發出更加可靠和易於維護的應用。

Namespaced Modules Design

在 Vuex 中使用 namespaced modules 是一種組織大型應用中狀態管理代碼的有效方式。它允許你將 store 分割成模塊，每個模塊擁有自己的 state、mutations、actions 和 getters。這樣做有助於代碼的維護和重用，同時避免了命名衝突。下面我們將通過一個簡單的例子來演示如何在 Vuex 中使用 namespaced modules。

假設我們的 Vue 應用有用戶管理（user）和產品管理（product）兩個功能模塊，我們希望對這些功能進行模塊化管理。

步驟 1：創建模塊

首先，在 `store` 目錄下為每個模塊創建一個文件，例如 `user.js` 和 `product.js`。

store/user.js:

```
export default {
  namespaced: true,
  state: () => ({
    users: []
  }),
  mutations: {
    addUser(state, user) {
      state.users.push(user);
    }
  },
  actions: {
    addUser({ commit }, user) {
      commit('addUser', user);
    }
  },
  getters: {
    userCount: state => state.users.length,
  }
};
```

store/product.js:

```
export default {
  namespaced: true,
  state: () => ({
    products: []
  }),
  mutations: {
    addProduct(state, product) {
      state.products.push(product);
    }
  },
  actions: {
    addProduct({ commit }, product) {
      commit('addProduct', product);
    }
  },
  getters: {
    productCount: state => state.products.length,
  }
};
```

步驟 2：將模塊組合到主 store 文件

然後，在 `store/index.js` 中導入這些模塊並使用它們：

```
import { createStore } from 'vuex';
import user from './user';
import product from './product';

export default createStore({
  modules: {
    user,
    product
  }
});
```

步驟 3：在組件中使用模塊化的 state、getters、actions

當你需要在組件中使用這些模塊化的 state、getters 或 actions 時，你需要指定模塊的命名空間。

例如，如果你想在一個組件中使用 `user` 模塊的 `addUser` action 和 `userCount` getter，你可以這樣做：

```
<template>
  <div>
    <button @click="addUser({ name: 'New User' })">Add User</button>
    <p>User count: {{ userCount }}</p>
  </div>
</template>

<script>
import { mapActions, mapGetters } from 'vuex';

export default {
  methods: {
    ...mapActions('user', ['addUser']),
  },
  computed: {
    ...mapGetters('user', ['userCount']),
  },
};
</script>
```

使用 `namespaced: true` 使得模塊具有更好的封裝性和可重用性，同時也讓狀態管理的結構更加清晰。透過將應用分割成不同的模塊，你可以更容易地維護和擴展你的 Vuex store。