VueX

DEVinHouse

Parcerias para desenvolver a sua carreira

SENAI

<LAB365>

AGENDA DA SEMANA

- Transition
- Transition Group
- VueRouter
- **VueX e gerenciamento de estado**

Conceito

O VueX é um padrão de gerenciamento de estado + biblioteca para aplicações Vue.js. Ele serve como um *store* centralizado para todos os componentes em uma aplicação, com regras garantindo que o estado só possa ser mutado de forma previsível.

O VueX até a versão 4, era a biblioteca oficial de gerenciamento do Vue. A partir dessa versão, o gerenciador de estado oficial passou a ser o Pinia. A versão 5 do VueX têm quase a mesma API do Pinia, então pode-se considerar que a versão 5 do VueX é o Pinea com outro nome, mas considere utilizar a versão recente do Pinia pois é a que ganhará novas atualizações com melhorias e novas funcionalidades. Fonte: https://vuex.vuejs.org/

Comunicação de componentes

Atualmente, a comunicação entre os componentes é efetuada através de parâmetro (Do pai para o filho) e envio de eventos pelo $emit (Do filho para o pai), mas esse processo torna a comunicação com os demais elementos da tela muito complexa e cansativa, onde o compartilhamento de eventos devem ser sempre declaradas.

Padrão de gerenciamento de estado

No centro de cada aplicação Vuex existe o *store*. Um "*store*" é basicamente um container que mantém o estado da sua aplicação. Há duas coisas que tornam um *store* Vuex diferente de um objeto global simples:

1. Os *stores* Vuex são reativos. Quando os componentes do Vue obtêm o estado dele, eles atualizarão de forma reativa e eficiente se o estado do *store* mudar.
2. Você não pode alterar diretamente os estados do *store*. A única maneira de mudar o estado de um *store* é explicitamente confirmando (ou fazendo *commit* de) mutações. Isso garante que cada mudança de estado deixe um registro rastreável, e permite ferramentas que nos ajudem a entender melhor nossas aplicações.

Criação da Store

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


// Declaração da store
const store = createStore({
    // modules, state, mutation, getters, actions...
})

const app = createApp(App);

app.use(store);

app.mount('#app');
```

Estado

Vuex usa uma única árvore de estado - ou seja, este único objeto contém todo o estado da sua aplicação e serve como "fonte única da verdade". Isso também significa que normalmente você terá apenas uma *store* para cada aplicação. Uma única árvore de estado facilita a localização de uma parte específica do estado e nos permite capturar facilmente momentos do estado atual da aplicação para fins de depuração.

```
// Declaração da store
const store = createStore({
    // Declaração de estados
    state() {
        return {
            count: 1
        }
    }
})
```

Getters

O Vuex nos permite definir *getters* no *store*. Você pode pensar neles como dados computados para os *stores*. Como os dados computados, o resultado de um *getter* é armazenado em *cache* com base em suas dependências e só será reavaliado quando algumas de suas dependências forem alteradas.

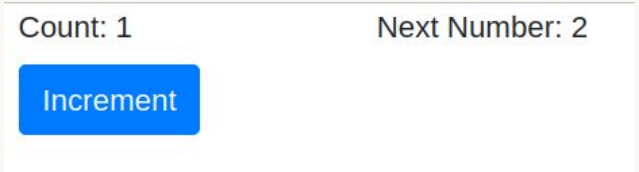
```
// Declaração de getters
getters: {
    nextNumber(state) {
        return state.count + 1
    }
},
```

Mutations

A única maneira de realmente mudar de estado em um *store* Vuex é por confirmar (ou fazer *commit* de) uma mutação. As mutações do Vuex são muito semelhantes aos eventos: cada mutação têm uma *String type* e um *handler*. Na função manipuladora (ou *handler*) é onde realizamos modificações de estado reais e ele receberá o estado como o 1º argumento

```
<template>
    <div class="container">
        <div class="row">
            <div class="col-2">
                <label>Count: {{$store.state.count}}</label>
            </div>
            <div class="col-2">
                <label>Next Number: {{$store.getters.nextNumber}}</label>
            </div>
        </div>
        <button class="btn btn-primary" @click="increment">Increment</button>
    </div>
</template>
<script>
    export default {
        methods: {
            increment() {
                this.$store.commit('increment');
            }
        }
    }
</script>
```

```
// Declaração de mutations
mutations: {
    increment(state) {
        state.count++
    }
}
```



Mutations

Você pode passar um argumento adicional para o store.commit, que é chamado de *payload* para a mutação:

```
// Declaração de mutations
mutations: {
    increment(state, n) {
        state.count = state.count + n
    }
}
```

```
export default {
    methods: {
        increment() {
            this.$store.commit('increment', 2);
        }
    }
}
```

Obs.: Uma regra importante a lembrar é que as funções manipuladoras de mutação devem ser síncronas

Actions

As ações são semelhantes às mutações, as diferenças são as seguintes:

- Em vez de mudar o estado, as ações confirmam (ou fazem *commit* de) mutações.
- As ações podem conter operações assíncronas arbitrárias.

Actions

As funções manipuladoras de ação recebem um objeto *context* que expõe o mesmo conjunto de métodos/propriedades na instância do *store*, para que você possa chamar context.commit para confirmar uma mutação ou acessar o estado e os *getters* através do context.state e do context.getters. Podemos até chamar outras ações com context.dispatch. Você pode despachar ações em componentes com this.$store.dispatch('xxx'):

```
incrementAsync (context) {
    setTimeout(() => {
        context.commit('increment')
    }, 1000)
}
```

```
start() {
    this.$store.dispatch('incrementAsync');
}
```

Módulos

Devido ao uso de uma única árvore de estado, todos os estados da nossa aplicação estão contidos dentro de um grande objeto. No entanto, à medida que nossa aplicação cresce em escala, o *store* pode ficar realmente inchado.

Para ajudar com isso, o Vuex nos permite dividir nosso *store* em módulos. Cada módulo pode conter seu próprio estado, mutações, ações, *getters* e até módulos aninhados - o padrão se repete em todo o fluxo

Módulos

```
const usuarioModulo = {
    state() {
        return {
            name: 'Gilmar'
        }
    }
}

const autenticacaoModulo = {
    state() {
        return {
            autorizado: true
        }
    }
}

const store = createStore({
    modules: {
        user: usuarioModulo,
        auth: autenticacaoModulo
    }
})
```

```
<template>
    <div class="container">
        <div class="row">
            <div class="col-2">
                <label>Usuario: {{ $store.state.user.name }}</label>
            </div>
            <div class="col-2">
                <label>Autenticado: {{ $store.state.auth.autorizado }}</label>
            </div>
        </div>
    </div>
</template>
```

Usuario: Gilmar Autenticado: true

DEVinHouse

Parcerias para desenvolver a sua carreira

OBRIGADO!

SENAI

<LAB365>