## **vue中的状态管理**

初始化项目环境

sudo npm install @vue/cli -g

vue ui 生成项目 vue-router + vuex

## **EventBus事件车**

在vue中传递数据是通过属性传递(父子关系)，子父通信是通过emit来触发父级事件。如果遇到平级组件可以通过共同的父级进行数据传递。但是在开发中，我们经常会遇到平级组件数据交互和跨组件数据交互就可以通过一个共同的实例来进行数据传递。 通过事件来共享数据（发布订阅模式）

创建bus实例

// lib/bus.js

import Vue from 'vue';

let $bus = new Vue();

Vue.prototype.$bus = $bus;

// main.js

import './lib/bus';

// Boy组件 发射dinner事件

<template>

<div>男孩

<button @click="sayToGirl()">对女孩说话</button>

</div>

</template>

<script>

export default {

methods: {

sayToGirl(){

this.$bus.$emit('dinner','你饿吗');

}

}

}

</script>

// Girl组件 监听dinner事件

<template>

<div>

女孩 <span>男孩对我说： {{message}}</span>

</div>

</template>

<script>

export default {

data(){

return {message:''}

},

mounted() {

this.$bus.$on('dinner',(data)=>{

this.message = data;

})

}

}

</script>

## **使用vuex来管理状态**



├── actions.js

├── getters.js

├── index.js

├── modules

│ └── teacher.js

├── mutations.js

└── state.js

## **state & getters**

* state类比组件中的状态

// state.js

let state = {

lesson:'珠峰架构'

}

export default state;

// 可以在组件中直接通过 this.$store.state来访问数据，也可以通过计算属性的方式

<template>

<div>

课程是:{{lesson}}

</div>

</template>

<script>

export default {

computed: {

lesson(){

return this.$store.state.lesson

}

}

}

</script>

// vue提供的辅助函数实现

import {mapState} from 'vuex';

export default {

computed: {

// 直接取出状态中的结果

...mapState(['lesson']),

// 给状态起名字

...mapState({lesson1:'lesson'}),

// 通过函数的方式获取结果

...mapState({lesson2:state=>state.lesson})

}

}

// 模块中的状态

let teacher = {

namespaced:true,

state:{

name:'姜文'

}

}

export default teacher;

// 取值时需要通过模块的名字来获取对应的状态

<template>

<div>

teacherName: {{name}}

</div>

</template>

<script>

import {mapState} from 'vuex';

export default {

computed: {

...mapState({name:state=>state.teacher.name}),

// 当模块中指定了namespaced:true时 可以使用第一个参数来限定取值的模块

...mapState('teacher',['name']),

// state指代的是teacher中的状态

...mapState('teacher',{teacherName:state=>state.name})

}

}

</script>

// 使用createNamespacedHelpers

import {createNamespacedHelpers} from 'vuex';

// 通过createNamespacedHelpers 方法来获取对应的mapstate

let {mapState} = createNamespacedHelpers('teacher');

export default {

computed: {

...mapState(['name'])

}

}

默认模块中的状态都是挂在在对应的模块内，并没有直接放到根状态上。像后面的getters/mutations/actions默认都会合并在根模块中

* getters类比组件中的计算属性

import {mapState,mapGetters} from 'vuex';

<div>

{{getLessonName}}

</div>

export default {

computed: {

// getName(){

// return this.$store.getters.getLessonName

// }

...mapGetters(['getLessonName'])

}

}

// 如果模块中有namespaced:true

<template>

<div>

teacherName: {{getTeacherName}}

</div>

</template>

<script>

import {createNamespacedHelpers} from 'vuex';

let {mapState,mapGetters} = createNamespacedHelpers('teacher');

export default {

computed: {

...mapGetters(['getTeacherName'])

}

}

</script>

把模块中的状态进行计算，映射出对应的结果

## **mutations & actions**

* mutation突变，唯一修改状态的方式

strict:process.env.NODE\_ENV !== 'production' // 严格模式修改状态只能通过mutation来修改

let mutations = {

SET\_LESSON\_NAME(state,payload){ // 载荷

state.lesson = payload;

// 修改时需要获取对应的属性

// state.lesson = payload.name;

}

}

export default mutations;

// 载在组件中调用commit方法 触发mutation对应的方法

changeName(){

this.$store.commit('SET\_LESSON\_NAME','node')

// 可以写成对象的方式传递

// this.$store.commit({

// type:'SET\_LESSON\_NAME',

// name:'node'

// });

}

给状态新增不存在的属性，需要通过Vue.set方法

Vue.set(state,'number',payload.number);

import {mapState,mapGetters,mapMutations} from 'vuex';

methods: {

...mapMutations(['SET\_LESSON\_NAME']),

changeName(){

this['SET\_LESSON\_NAME']({number:10});

}

}

mutation不能操作异步逻辑

* actions就是用来处理异步的请求，异步更新状态

// 派发动作到action中

this.$store.dispatch('SET\_LESSON\_NAME');

// 在action中处理异步逻辑后将结果提交给mutation

import {getLesson} from '../api/lesson'

let actions = {

// 在action中需要将数据提交给mutation，这里可以做异步逻辑

SET\_LESSON\_NAME({commit},payload){

getLesson().then(data=>{

// data => {name:node}

commit({type:'SET\_LESSON\_NAME',...data});

})

}

}

export default actions;

// 使用mapActions

methods: {

...mapActions(['SET\_LESSON\_NAME']),

changeName(){

this['SET\_LESSON\_NAME']();

}

}

action中可以做封装异步请求，多个组件相同的异步处理，可以直接调用action,action中可以多次commit,也可以在action中再次调用action

## **vuex进阶**

自动保存到本地插件

// vuex中的store容器

// vuex中的store容器

export default (store)=>{

// 用新的状态 替换掉老的状态

store.replaceState(JSON.parse(localStorage.getItem('state'))|| store.state);

store.subscribe((mutation,state)=>{ // 订阅每次commit都会触发此函数

localStorage.setItem('state',JSON.stringify(state));

});

}

// 使用插件

import saveLocale from './plugins/saveLocale'

export default new Vuex.Store({

state,

mutations,

getters,

actions,

strict:process.env.NODE\_ENV !== 'production',

modules:{

teacher

},

plugins:[saveLocale]

});

logger插件(vuex中自己实现了这个插件)

import deepClone from 'lodash/cloneDeep'

export default (store)=>{

let prevState = deepClone(store.state);

store.subscribe((mutation,state)=>{

console.log('prev',prevState.lesson);

console.log(mutation);

console.log('next',state.lesson);

prevState = deepClone(state);

});

}

vuex双向绑定,当更新数据时手动提交数据的更改

<input type="text" v-model="teacherName">

computed: {

...mapState('teacher',['name']),

teacherName:{

get(){

return this.name;

},

set(val){

this['SET\_TEACHER\_NAME'](val);

}

}

}