vuex

安状四模插严表

1**安**装（1）在模块化的打包系统中，须显示地通过Vue.use()来安装Vuex

import Vue from ”vue”

import Vuex from ”vuex”

Vue.use(Vuex)

（2）当使用全局script标签引用Vuex时，不需要以上安装过程

Vuex

1. Vuex是专为Vue.js应用程序开发的**状**态管理模式
2. Backend API

Dispatch Commit

Actions

Vue Component Mutations Devtools

State

Render Mutate

Vuex

2 Store：

1. const store = new Vuex.Store({

})

3 **state**

1. Vuex使用单一状态树，用一个对象包含了全部的应用层级状态
2. const store = new Vuex.Store({

state:{

状态名：状态值

}

})

1. 在Vue组件中获得Vuex状态：由于Vuex的状态存储是响应式的，从store中读取状态最简单方法是在计算属性中返回某个状态

computed:{

count:function(){

return store.state.count

}

}

1. 上面模式导致组件依赖全局状态单例。在模块化构建系统中，每个需要使用store的组件中需频繁导入，并在测试组件时需模拟状态。

因此在使用Vue.use(Vuex)时，可在组件中有store属性，将状态从根组件注入每一个子组件中。从此子组件可直接使用this.$store.state.count

1. mapState辅助函数

import {mapState} from ”vuex”

[1]computed:mapState({

count:state=>state.count,

countAlias:”count”,

//传字符串参数”count”等同于state=>state.count

countWithLocal(state){

return state.count+this.localCount

}

})

//若要使用this获取局部状态，必须使用常规函数

[2]当映射的计算属性名称与state的状态名相同时，可给mapState传一

个字符串数组

computed:mapState([”count”])

1. 局部状态：有些状态严格属于单个组件，应作为组件的局部状态
2. 对象展开运算符：使mapState与局部计算属性混合使用

computed:{

localComputed(){},

…mapState({

})

}

4 **getters**（可认为是store的计算属性）。其中方法接收state作为第一个参数，可接受getters作为第二个参数。暴露为store.getters对象，使用同state

5 **mutations**：更改Vuex的store中的状态的唯一方法是提交mutation。

1. 其中函数接受state作为其第一个参数

const store = new Vuex.Store({

……

mutations:{

突变名(state){

}

}

})

store.commit(”突变名”)

1. 载荷

mutations:{

突变名(state,变量名类型可为基本类型或对象){

}

}

store.commit(”突变名”,第二个参数);

当第二个参数是对象时可：

store.commit({

type:”突变名”,

对象中属性名:对象中属性值

})

1. 静默提交
2. mutation必须是同步函数
3. 在组件中提交mutation，在根节点注入store

[1]this.$store.commit形式

[2]import {mapMutations} from ”vuex”

methods:{

…mapMutations([”突变名”]),

…mapMutations({

突变名别名:”突变名”

})

}

6 **actions**：类似mutation，不同在于提交mutation而不是直接变更状态。可包含异步操作

1. 其中函数接受一个与store实例具有相同方法和属性的context对象
2. commit换成dispatch，其余用法与mutations一致
3. 在组件中分发action，mapMutations换成mapActions，其余用法与mutations一致
4. 判断action结束及组合actions

[1]actions:{

actionA({commit}){

return new Promise((resolve,reject)=>{

setTimeout(()=>{

commit(”突变名”);

resolve();

},1000);

})

}

}

store.dispatch(”actionA”).then(()=>{

//actionA执行结束后执行此处

})

[2]actions:{

//…

actionB({commit,dispatch}){

return dispatch(”actionA”).then(()=>{

commit(”突变名”);

})

}

}

7 modules:允许将store分割到**模**块，每个模块有自己的state，getters，mutations，actions

1. const moduleA = {……}

const moduleB = {……}

const store = new Vuex.Store({

modules:{

a:moduleA,

b:moduleB

}

})

store.state.a //moduleA的状态

store.state.b //moduleB的状态

模块内的state是模块的局部状态，rootState是根节点的状态

1. 模块动态注册

在store创建后，可store.registerModule(”模块名”,{})来注册模块

模块状态是store.state.模块名

可使用store.unregisterModule(模块名)动态卸载模块，不能用此方法卸载静态模块，即在创建store时声明的模块

8 项目结构

9 **插**件：是一个函数，接收store作为唯一参数

1. const 插件名 = store=>{

//当store初始化后调用

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

//每次mutation之后调用

//mutation的格式为{type , payload}

})

}

这样使用

const store = new Vuex.Store({

//…

plugins:[插件名]

})

（2）生成state快照：有时插件需获得状态的快照，比较前后状态，想实现这个功能，要对状态进行深拷贝

const 插件名 = store=>{

let prevState = \_.cloneDeep(store.state)

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

let nextState = \_.cloneDeep(state)

//比较prevState和nextState

//保存状态，用于下一次mutation

prevState=nextState

})

}

生成状态快照的插件应只在开发阶段使用，让构建工具webpack帮处理

const store = new Vuex.Store({

//…

plugins:process.env.NODE\_ENV!==”production”?[插件名]:[]

})

插件默认启用，发布阶段用webpack的DefinePlugin使

process.env.NODE\_ENV!==”production”为false

10 **严**格模式：若发生不是由mutation引起的状态变更，会抛出错误。不要在发布环境下使用严格模式

const store = new Vuex.Store({

//…

strict:true

})

11 **表**单处理

1. computed:{

message:{

get(){

return this.$store.state.obj.message

},

set(value){

this.$store.commit(”update”,value)

}

}

}

1. <input :value=”message” @input=”updateMessage” />

computed:{

…mapState({

message:state=>state.obj.message

})

},

methods:{

updateMessage(e){

this.$store.commit(”updateMessage”,e.target.value)

}

}

mutations:{

updateMessage(state,message){

state.obj.message=message

}

}

12 测试

13 热重载