export default new Vuex.Store({

// 单一状态树

state: {

thisUsers: [{

name:"wanglaosi"

},{

name:"malaosan"

},{

name:"wangmaily"

}]

},

// 更改 Vuex 的 store 中的状态的唯一方法是提交 mutation。 mutation类似于事件：每个 mutation 都有一个字符串的 事件类型 (type) 和 一个 回调函数 (handler)。这个回调函数就是我们实际进行状态更改的地方，并且它会接受 state 作为第一个参数。

mutations: {

add(state, data) {

state.thisUsers.push(data);

},

del(state, index) {

state.thisUsers.splice(index, 1)

}

},

// Action 类似于 mutation，不同在于：Action 提交的是 mutation，而不是直接变更状态。Action 可以包含任意异步操作。

actions: {

add({ commit }, data) {

commit('add', data) // commit提交一个mutation

},

del({ commit }, index) {

commit('del', index) // commit提交一个mutation

}

},

// 将 store 分割成模块（module）：每个模块拥有自己的 state、mutation、action、getter、甚至是嵌套子模块——从上至下进行同样方式的分割。

modules: {

},

// 可以认为是 store 的计算属性，Getter 接受 state 作为其第一个参数

getters: {

thisUsers: state => {

return state.thisUsers // 返回state.taskLists

}

}

})

<template>

<div class="about">

<h1>信息列表</h1>

<input v-model="inputUser" />

<button @click="addUser">新增</button>

<div v-for="(all, index) in this.$store.getters.thisUsers" :key="index">

{{index+1}}--{{all.name}} <button @click="del(index)">x</button> <button @click="modify()">√</button>

</div>

</div>

</template>

<script>

export default {

name:'about',

data(){

return {

inputUser:'',

}

},

methods: {

// 新增

addUser() {

if(this.inputUser == '') {

alert('请输入后再点击新增！');

return;

}

this.$store.dispatch('add', {id:this.$store.getters.thisUsers.length+1, name:this.inputUser})

alert('新增成功！');

},

// 删除

del: function(index) {

this.$store.dispatch('del', index);

alert('删除成功！');

}

}

}

</script>