状态管理-Vuex

项目中数据信息存储

使用的技术是：storage

属于html5的一个知识点，专门去处理和解决客户端存储的一个问题。它的存储有两个：

- sessionStorage: 浏览关闭信息丢失
- localStorage：只要你不卸载浏览器，不清楚浏览缓存，信息一直都在。

共同的api方法：

- setItem(key,value)
- getItem(key)
- removeItem(key)

应用场景

- 添加文章的草稿
- 记录网站的个人信息，比如播放速率，播放进度，播放的音量
- 记录用户登录的信息

有一个缺陷

- 没有时间限制
- 没有隔离，隔离只能通过key的名字来隔离。
- 无状态

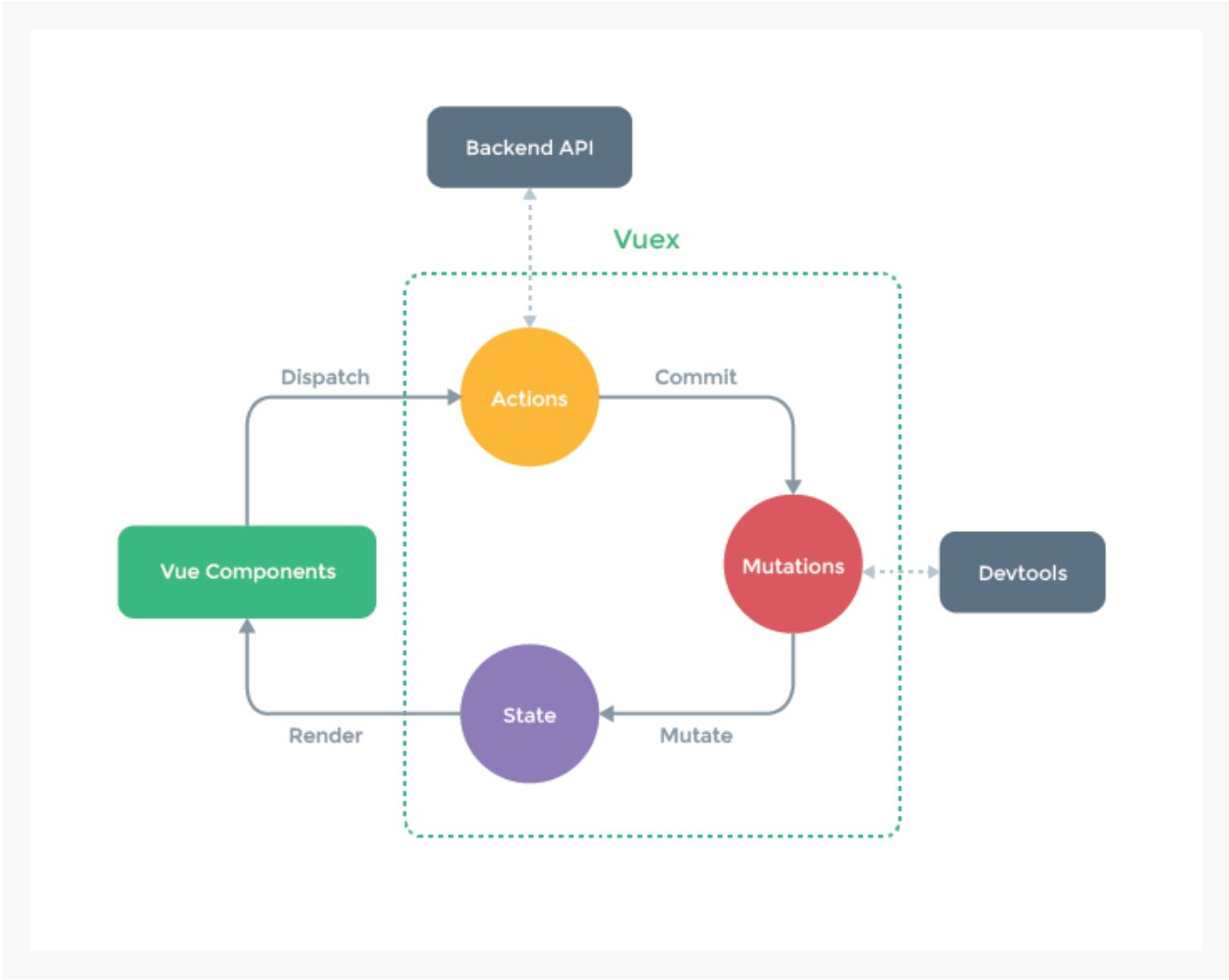
01、概述

官网：https://next.vuex.vuejs.org/installation.html

在前面我们知道父子组件之间是通过prop向子组件传递数据，子组件通过自定义事件$emit向父组件传递数据。然而是实际项目中，经常会遇到多个组件需要访问同一数据的情况，且需要根据数据的变化做出响应（computed）。而这些组件之间可能并不是父子组件这种简单的关系。在这种情况下，就需要一个全局的状态管理方案，在Vue开发中，官方推荐使用Vuex。

Vuex 是一个专为 Vue.js 应用程序开发的状态管理模式。它采用集中式存储管理应用的所有组件的状态，并以相应的规则保证状态以一种可预测的方式发生变化。Vuex 也集成到 Vue 的官方调试工具 devtools extension (opens new window)，提供了诸如零配置的 time-travel 调试、状态快照导入导出等高级调试功能。

02、工作原理图



什么情况下我应该使用 **Vuex**?

Vuex 可以帮助我们管理共享状态，并附带了更多的概念和框架。这需要对短期和长期效益进行权衡。

如果您不打算开发大型单页应用，使用 Vuex 可能是繁琐冗余的。确实是如此——如果您的应用够简单，您最好不要使用 Vuex。一个简单的 store 模式 (opens new window)就足够您所需了。但是，如果您需要构建一个中大型单页应用，您很可能会考虑如何更好地在组件外部管理状态，Vuex 将会成为自然而然的选择。

> 一句话：如果你开发中，你数据要做共享，并且数据修改以后要有状态（马上同步组件和*SPA*单页与之有关的数据。）

03、安装Vuex

官网安装：https://next.vuex.vuejs.org/installation.html

npm安装

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

yarn安装

```
yarn add vuex@next --save
```

@next 最新版，为了迎合vue3。但是你不加，vue2的版本.

04、定义和注册

当使用全局 script 标签引用 Vuex 时，不需要以上安装过程。

在Vue3.0的脚手架项目中使用，在main.js文件中导入createstore。并调用该方法创建一个store实例。之后使用vue.js应用程序实例的use()方法将实例作为插件安装。代码如下所示：

定义

在src目录下新建一个store/index.js

```

```

```
// 1: 导入状态管理对象
import {createStore} from "vuex"

// 2 : 实例化一个store对象
const store = createStore({
    // 定义你需要管理对象或者属性。把这个当中之前是spa或者组件data属性。
    state() {
        return {
            count: 1
        }
    }
})

// 3: 导出store
export default store;
```

注册

在main.js中注册store如下：

```
import {createApp} from 'vue'
import App from './App.vue'
import router from './router'
import Loading from './plugins/Loading'
import ElementPlus from 'element-plus'
import 'element-plus/dist/index.css'
import store from './store' // 导入


const app = createApp(App);
app.use(router);
app.use(store);//注册
app.use(Loading);
```

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

```

05、为什么要学习vuex而不用storage

两者的差异：一个有状态一个无状态。vuex是有状态的，修改了数据会及时的响应到视图上和数据中。而storage是无状态的，修改了storage的数据是不会及时的刷新到视图上和数据模中。

06、Vuex的基本语法-实现购物车

1：定义store/index.js

```
import {createStore} from "vuex"

const store = createStore({
    state() {
        return {

        }
    }
})


export default store;
```

2：注册

```
import {createApp} from 'vue'
import App from './App.vue'
import router from './router'
import Loading from './plugins/Loading'
import ElementPlus from 'element-plus'
import 'element-plus/dist/index.css'
import store from './store'


const app = createApp(App);
app.use(router);
app.use(store);
app.use(Loading);
app.use(ElementPlus)
app.mount('#app');

```

3：定义一个books.js

```
export default [{
    id: 1,
    title: "Java无难事",
    price: 188,
    count: 1
}, {
    id: 2,
    title: "JS面向对象编程",
    price: 19,
    count: 1
}, {
    id: 3,
    title: "JSPJavaWeb开发",
    price: 119,
    count: 1
```

```
}]
```

4：在books模块中导出一个图书商品的数组，将该数组作为store中状态数据的来源。编辑store目录下的index.js文件，修改后如下：

```
import {createStore} from "vuex"
import books from "@/mock/books";

const store = createStore({
    state() {
        return {
            // 导入数据进行初始化
            items: books
        }
    }
})

export default store;
```

5：定义购物车的SPA单页

```
<template>
  <div>
    <table>商品编号</table>
    <table>
      <tr>
        <td>商品编号：</td>
        <td><input type="text" v-
model.number="book.id"></td>
      </tr>
      <tr>
        <td>商品标题：</td>
        <td><input type="text" v-
model.number="book.title"></td>
```

```
12        </tr>
13        <tr>
14          <td>商品价格：</td>
15          <td><input type="text" v-
    model.number="book.price"></td>
16        </tr>
17        <tr>
18          <td colspan="2">
19            <button @click="addCart">添加购物车</button>
20          </td>
21        </tr>
22      </table>
23      <hr>
24      <table>
25        <thead>
26        <tr>
27          <th>编号</th>
28          <th>商品名称</th>
29          <th>价格</th>
30          <th>数量</th>
31          <th>金额</th>
32          <th>操作</th>
33        </tr>
34        </thead>
35        <tbody>
36        <tr v-for="(book,index) in books"
    :key="book.id">
37          <td>{{ book.id }}</td>
38          <td>{{ book.title }}</td>
39          <td>{{ book.price }}</td>
40          <td>
41            <button @click="add(index)">+</button>
42            {{ book.count }}
43            <button @click="minus(index)">-</button>
```

```
        </td>
        <td>{{ book.count * book.price }}</td>
        <td>
          <button>删除</button>
        </td>
      </tr>
      </tbody>
    </table>
    <div>总价：￥1.00</div>
  </div>
</template>

<script>

export default {
  name: "index",
  data() {
    return {
      book: {
        id: "",
        title: "",
        price: "",
        count: 1
      }
    }
  },
  computed: {
    books() {
      return this.$store.state.items;
    }
  },
  methods: {
    addCarts() {

```

```
    },
    minus() {
    },
    add() {

    }
  }
}
</script>

<style scoped>
table {
  width: 100%;
  border-collapse: collapse;
}
</style>
```

路由注册

```
//1:  导入路由
import {createRouter, createWebHistory} from 'vue-router'
import PugLoading from '@/plugins/PugLoading'

//2: 创建路由器对象，将模板全部进行路由匹配和注册
const router = createRouter({
    history: createWebHistory(),
    routes: [
        {
            path: "/",
            name: "layout",
            component: () => import('@/layout/Layout'),
            redirect: "/index",
```

```
            //这里只是重定向浏览器地址，但是渲染还是在父级，
            // 好处：就是可以打破不用和父同名默认是首页的规则
            children: [
                {
                    path: "/index",
                    name: "index",
                    meta: {title: "首页"},
                    component: () =>
import('@/views/index'),
                },
                {
                    path: "/user",
                    name: "user",
                    meta: {title: "用户管理", isAuth:
true},
                    component: () =>
import('@/views/user/index'),
                    children: [{
                        path: "/user/add",
                        name: "useradd",
                        meta: {title: "用户添加",
isAuth: true},
                        component: () =>
import('@/views/user/add'),
                    }]

                },
                {
                    path: "/course",
                    name: "course",
                    meta: {title: "课程管理"},
                    // 1: 守卫在全局守卫调用之后，只在进入
路由时触发。每次进入只触发一次。不会监听参数和hash发生变化时触
发。
```

```
41                      beforeEnter(to, from) {
42                          console.log(to, from)
43                      },
44                      component: () =>
    import('@/views/course/index')
45                  },
46                  {
47                      path: "/order",
48                      name: "order",
49                      meta: {title: "订单管理"},
50                      component: () =>
    import('@/views/order/index')
51                  },
52
53                  {
54                      path: "/product",
55                      name: "product",
56                      meta: {title: "产品管理"},
57                      component: () =>
    import('@/views/product/index')
58                  },
59
60                  {
61                      path: "/book",
62                      name: "book",
63                      meta: {title: "图书管理"},
64                      component: () =>
    import('@/views/book/index')
65                  }
66
67              ]
68          },
69          {
70              path: "/login",
```

```
            name: "login",
            component: () => import('@/views/login')
        },
        {
            path: "/toLogin",
            redirect: "/login"
        },
        {
            path: "/error",
            name: "error",
            component: () => import('@/views/error')
        }
    ]
});


// 1：路由请求进入前置守卫
// 参数：
// 参数1：to ： 当前你访问路由
// 参数2： from：上次访问的路由
// 参数3： next： 进入下一个守卫 ,执行的时候必须是next()
// 返回值：
// 返回值1： return true 相当于 next() 继续执行请求
// 返回值2： return false ：  终止请求
// 返回值3： return {path:""} ，如果你指定path，就直接转发到你指定的地址，相当于$router.push({path:"/xxx"});
// 返回值4： next()// 继续请求
router.beforeEach((to) => {
    if (to.matched.length == 0) {
        return {path: "/error"}
    }
})


```

```
router.beforeEach((to) => {
    // 1: 排除不需要拦截的授权的页面
    if (to.path == "/login") {
        return true;
    }

    if (to.meta.isAuth) {
        //1: 如果登录成功，直接访问成功
        if (sessionStorage.isAuth) {
            return true;
        } else {
            // 2: 如果用户访问的受保护的资源，且没有登录，则跳转到登录页面
            // 举个场景：在操作系统，但是突然要吃饭了，过了30分钟以后继续来访问系统。因为token有时间限制。这个肯定会转发登录去、
            // 为了良好的体验
            // 将当前路由的完整的地址做为参数传递给Login.vue的组件，以便登录成功以后继续回到该位置
            return {path: "/login", query: {back: to.fullPath}}
        }
    }

    PugLoading();

})

// 后置防卫
router.afterEach(to => {
    setTimeout(() => {
        PugLoading("remove");
    }, 1000)

```

```
133     document.title = to.meta.title;
134 })
135
136
137 // 3: 导出模板router模块即可
138 export default router
```

07：实现状态管理监听-Computed+$store

在main.js引入了store实例以后，该store实例会被注入根组件下的所有子组件中，因此在组件中，就可以通过this.$store访问store了。如果在组件中要展示store中的状态，应该使用计算属性返回store的状态。

```
computed: {
    books() {
        return this.$store.state.items;
    }
}
```

08、实现购物车添加功能

那么如何更改store中的状态呢？注意不要直接修改items的值，例如：

```
addCarts() {
    this.$store.state.books.push(this.book);
}
```

上面的方式，是不会有效，state的状态的数据是没办法直接修改，既然我们选择了vuex作为应用的状态管理方案，那么就应该遵照vuex的要求，通过提交mutation()函数更改store中的状态。在严格模式下，如果store中的状态改变不是由mutation()函数引起的，否则会抛出错误，如果直接去修改store中的状态，Vue的调试工具也无法跟踪状态的改变，在开发阶段，可以启用严格模式，以避免直接的修改状态，在创建store时，传入strict:true。来做严格限制：

```
import {createStore} from "vuex"
import books from "@/mock/books";

const store = createStore({
    strict:true,//严格模式，不允许直接通过$store.state修改
    state() {
        return {
            // 导入数据进行初始化
            items: books
        }
    }
})


export default store;
```

09、Vuex的mutation()函数

> 定义修改 *state*数据的行为，它也是唯一一个能够修改*state*数据方式。

vuex的mutation()函数类似于事件，每个mutation()函数都有一个字符串的事件类型和一个处理函数。这个处理器函数就是实际运行状态更改的地方，它接受state作为第1个参数。如下：

```
import {createStore} from "vuex"
import books from "@/mock/books";

const store = createStore({
    strict: true,
    state() {
        return {
            // 导入数据进行初始化
            items: books
        }
    },

    mutations: {
        pushBookToCart(state, book) {
            state.items.push(book);
        }
    }
})


export default store;
```

然后在book/index.vue的单页中使用this.$store.commit("pushBookToCart",this.book)添加商品到购物车，会将状态管理同步到store的state的items中。

```
<template>
  <div>
    <table>商品编号</table>
    <table>
```

```
5         <tr>
6           <td>商品编号：</td>
7           <td><input type="text" v-
  model.number="book.id"></td>
8         </tr>
9         <tr>
10          <td>商品标题：</td>
11          <td><input type="text" v-
  model.number="book.title"></td>
12        </tr>
13        <tr>
14          <td>商品价格：</td>
15          <td><input type="text" v-
  model.number="book.price"></td>
16        </tr>
17        <tr>
18          <td colspan="2">
19            <button @click="addCarts">添加购物车</button>
20          </td>
21        </tr>
22      </table>
23      <hr>
24      <table>
25        <thead>
26        <tr>
27          <th>编号</th>
28          <th>商品名称</th>
29          <th>价格</th>
30          <th>数量</th>
31          <th>金额</th>
32          <th>操作</th>
33        </tr>
34        </thead>
35        <tbody>
```

```
      <tr v-for="(book,index) in books"
:key="book.id">
        <td>{{ book.id }}</td>
        <td>{{ book.title }}</td>
        <td>{{ book.price }}</td>
        <td>
          <button @click="add(index)">+</button>
          {{ book.count }}
          <button @click="minus(index)">-</button>
        </td>
        <td>{{ book.count * book.price }}</td>
        <td>
          <button>删除</button>
        </td>
      </tr>
      </tbody>
    </table>
    <div>总价：￥1.00</div>
  </div>
</template>

<script>

export default {
  name: "index",
  data() {
    return {
      book: {
        id: "",
        title: "",
        price: "",
        count: 1
      }
    }
```

```
  },
  computed: {
    books() {
      return this.$store.state.items;
    }
  },
  methods: {
    addCarts() {
      this.$store.commit("pushBookToCart",this.book);
    },

    minus() {

    },

    add() {

    }
  }
}
</script>

<style scoped>
table {
  width: 100%;
  border-collapse: collapse;
}
</style>
```

同时为了体验store的全局同步性。你可以在layout/PugHeader.vue中增加状态管理的监听如下：

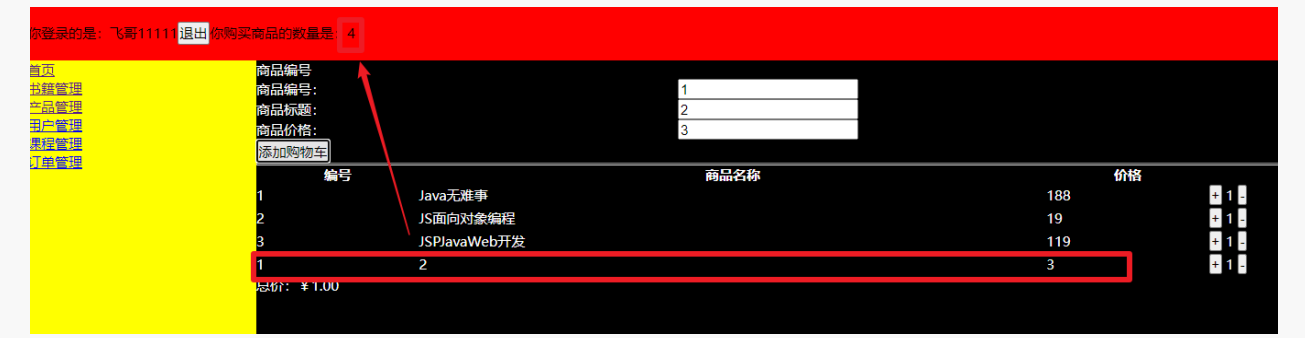
```
<template>
  <header class="pug-ui-header">
```

```
    <router-link v-if="!isAuth" to="/login">去登录</router-link>
    <span>你登录的是：{{user.nickname}}</span><button v-if="isAuth" @click="logout"> 退出</button>
    <span class="fr">你购买商品的数量是：{{books.length}}</span>
  </header>
</template>

<script>
export default {
  name: "PugHeader.vue",
  components:{
  },

  computed:{

    books() {
      return this.$store.state.items;
    },

    isAuth(){
      return sessionStorage.userId;
    },

    user(){
      const nickname = sessionStorage.nickname;
      const username = sessionStorage.username;
      return {nickname,username}
    }
  },

  methods:{
    logout(){
```

```
34          sessionStorage.removeItem("isAuth");
35          var currentPath =
    sessionStorage.getItem("currpath");
36          this.$router.push({name:"login",query:
    {back:currentPath}});
37      }
38    }
39  }
40  </script>
41
```

在执行添加商品，可以看见购物车的列表添加了商品。头部的购物车的数量也同步更新了。

添加后：



注意：头部PugHeader.vue和book/index.vue是两个SPA页面，是不同的，可以很清楚的体验到了vuex状态管理可以跨越组件和SPA的数据同步。

10、定义常量函数名

1、定义

在store下定义mutation-types.js如下：

```
// 1: 添加购物车
export const PUSH_BOOK_TO_CART = "pushBookToCart";
```

2、函数使用常量

在store/index.js导入mutation-types.js即可。如下：

```
import {createStore} from "vuex"
import books from "@/mock/books";
import {PUSH_BOOK_TO_CART} from './mutation-types'

const store = createStore({
    strict: true,
    state() {
        return {
            // 导入数据进行初始化
            items: books
        }
    },

    mutations: {
        // 可以使用es2015的计算属性命名功能来使用一个常量作为函数名，达到一个维护更方便的目的
        [PUSH_BOOK_TO_CART](state, book) {
            state.items.push(book);
```

```
        }
    }
})


export default store;
```

11、Vuex的mapMutations

继续完善购物车功能更，为购物车添加删除商品功能。删除商品同样要修改store中保存的购物车商品数量，因此继续在mutations选项中定义个deleteitem的方法。如下：

1：在store/index.js目录中定义如下：

在mutation-types.js常量中定义如下：

```
// 1: 添加购物车
export const PUSH_BOOK_TO_CART = "pushBookToCart";
// 2: 删除购物车
export const DELETE_ITEMS = "deleteItem";
```

2：定义状态管理的方法

```
import {createStore} from "vuex"
import books from "@/mock/books";
import {PUSH_BOOK_TO_CART,DELETE_ITEMS} from
'./mutation-types'

const store = createStore({
    strict: true,
```

```
    state() {
        return {
            // 导入数据进行初始化
            items: books
        }
    },

    mutations: {
        // 可以使用es2015的计算属性命名功能来使用一个常量作为函数名，达到一个维护更方便的目的
        [PUSH_BOOK_TO_CART](state, book) {
            state.items.push(book);
        },

        [DELETE_ITEMS](state, id) {
            // 根据提交的id。查询是否存在相同的商品id。返回商品的索引
            let index = state.items.findIndex(item => item.id === id);
            if (index >= 0) {
                state.items.splice(index, 1);
            }
        }
    }
})


export default store;
```

3：使用如下：

```
<template>
  <div>
    <table>商品编号</table>
```

```
    <table>
      <tr>
        <td>商品编号：</td>
        <td><input type="text" v-model.number="book.id"></td>
      </tr>
      <tr>
        <td>商品标题：</td>
        <td><input type="text" v-model.number="book.title"></td>
      </tr>
      <tr>
        <td>商品价格：</td>
        <td><input type="text" v-model.number="book.price"></td>
      </tr>
      <tr>
        <td colspan="2">
          <button @click="addCarts">添加购物车</button>
        </td>
      </tr>
    </table>
    <hr>
    <table>
      <thead>
      <tr>
        <th>编号</th>
        <th>商品名称</th>
        <th>价格</th>
        <th>数量</th>
        <th>金额</th>
        <th>操作</th>
      </tr>
      </thead>
```

```
35        <tbody>
36        <tr v-for="(book,index) in books"
  :key="book.id">
37          <td>{{ book.id }}</td>
38          <td>{{ book.title }}</td>
39          <td>{{ book.price }}</td>
40          <td>
41            <button @click="add(index)">+</button>
42            {{ book.count }}
43            <button @click="minus(index)">-</button>
44          </td>
45          <td>{{ book.count * book.price }}</td>
46          <td>
47            <button @click="deleteCart(book.id)">删除
  </button>
48          </td>
49        </tr>
50        </tbody>
51      </table>
52      <div>总价：￥1.00</div>
53    </div>
54  </template>
55
56  <script>
57  import {PUSH_BOOK_TO_CART,DELETE_ITEMS} from
  '@/store/mutation-types'
58  export default {
59    name: "index",
60    data() {
61      return {
62        book: {
63          id: "",
64          title: "",
65          price: "",
```

```
66          count: 1
67        }
68      }
69    },
70    computed: {
71      books() {
72        return this.$store.state.items;
73      }
74    },
75    methods: {
76      // 添加购物车
77      addCarts() {
78
    this.$store.commit(PUSH_BOOK_TO_CART,this.book);
79      },
80
81      // 删除购物车
82      deleteCart(id){
83        this.$store.commit(DELETE_ITEMS,id);
84      },
85
86
87      minus() {
88
89      },
90
91      add() {
92
93      }
94    }
95  }
96  </script>
97
98  <style scoped>
```

```
 99 table {
100   width: 100%;
101   border-collapse: collapse;
102 }
103 </style>
```

上面的代码可以很清楚的看到是没有任何问题的，如果组件中需要提交的mutation越来越多，使用this.$store.commit的方法提交就会越来越繁琐为了简化mutation的提交，可以使用mapMutations()辅助函数将组件中的方法映射为store.commit()调用。如下：

对象的方式：

```
1
2 import {DELETE_ITEMS, PUSH_BOOK_TO_CART} from
  '@/store/mutation-types'
3 import {mapMutations} from 'vuex'
4
5 methods: mapMutations({
6     addCarts: PUSH_BOOK_TO_CART,
7     deleteCart: DELETE_ITEMS
8 })
9
```

数组的方式

```
1
2 import {DELETE_ITEMS, PUSH_BOOK_TO_CART} from
  '@/store/mutation-types'
3 import {mapMutations} from 'vuex'
4 methods: mapMutations([
5     PUSH_BOOK_TO_CART,
6     DELETE_ITEMS
7 ])
```

无论用那种方式，都可以达到相同的效果。建议用对象的方式，可以更名，数组的方式就只能采用相同的名字了。

在大多数情况下，组件还有自己的方法，在这种情况西安，可以使用es6的展开运算符提取mapMutations()函数返回的对象属性。赋值到methods选项中。代码如下：

```
1  <template>
2    <div>
3      <table>商品编号</table>
4      <table>
5        <tr>
6          <td>商品编号：</td>
7          <td><input type="text" v-
   model.number="book.id"></td>
8        </tr>
9        <tr>
10         <td>商品标题：</td>
11         <td><input type="text" v-
   model.trim="book.title"></td>
12       </tr>
13       <tr>
14         <td>商品价格：</td>
15         <td><input type="text" v-
   model.number="book.price"></td>
```

```
      </tr>
      <tr>
        <td colspan="2">
          <button @click="addCarts">添加购物车</button>
        </td>
      </tr>
    </table>
    <hr>
    <table>
      <thead>
      <tr>
        <th>编号</th>
        <th>商品名称</th>
        <th>价格</th>
        <th>数量</th>
        <th>金额</th>
        <th>操作</th>
      </tr>
      </thead>
      <tbody>
      <tr v-for="(book,index) in books"
:key="book.id">
        <td>{{ book.id }}</td>
        <td>{{ book.title }}</td>
        <td>{{ book.price }}</td>
        <td>
          <button @click="add(index)">+</button>
          {{ book.count }}
          <button @click="minus(index)">-</button>
        </td>
        <td>{{ book.id }}</td>
        <td>
          <button @click="deleteCart(book.id)">删除
</button>
```

```
48          </td>
49        </tr>
50        </tbody>
51      </table>
52      <div>总价：0</div>
53    </div>
54  </template>
55
56  <script>
57
58  import {mapMutations} from 'vuex'
59
60  export default {
61    name: "index",
62    data() {
63      return {
64        book: {
65          id: "",
66          title: "",
67          price: "",
68          count: 1
69        }
70      }
71    },
72
73    // 获取状态管理的数据
74    computed: {
75      books() {
76        return this.$store.state.items;
77      }
78    },
79
80    // 1：原始写法，没问题
81    // methods: {
```

```
  //   addCarts() {
  //     // 往状态管理的数据中添加书籍，从而去改变它？
  //     this.$store.commit("putItemtoCarts", this.book);
  //   }
  // }
  // 2: 数组写法  好处：解决方法的名字定义问题，共享名字，坏处：传递参数只能在调用和执行的时候传递。
  //methods:mapMutations(['putItemtoCarts'])
  // 3: 对象写法 : 好处：可以自定义名字，还可以用行为去处理，只不过参数就是commit 坏处：传递参数只能在调用和执行的时候传递。
  // methods:mapMutations({
  //   addCarts(commit){
  //     commit('putItemtoCarts',this.book);
  //   }
  // })
  // 4: 解构写法
  methods:{
    // 数组方式
    //...mapMutations(['putItemtoCarts']),
    // 对象的方式
    ...mapMutations({
      addCarts(commit){
        commit('putItemtoCarts',this.book)
      }
    }),

    add(){

    },
  }
}
</script>

```

```
<style scoped>
.pug-ui-content table {
  width: 100%;
  border-collapse: collapse;
  margin: 10px 0
}

.pug-ui-content tr th, .pug-ui-content tr td {
  border: 1px solid #ccc;
  padding: 10px;
}
</style>
```

12、Vuex的mapState

当一个主键需要使用多个store状态属性时，将这些状态都声明为计算属性，就会有些重复和冗余，为了解决这个问题，可以使用mapState()辅助函数生成计算属性。如下：

数组方式：

```
import {mapMutations, mapState} from 'vuex'
computed: mapState([
    // 自动映射为：this.items === this.$store.state.items
    items
]),

```

对象方式：

```
import {mapMutations, mapState} from 'vuex'
computed: mapState({
    books: 'items'
}),

# 或者
computed: mapState({
    books: function (state) {
        return state.items;
    }
}),

# 或者
computed: mapState({
    books:  (state) => state.items;
}),
```

上面的方式一样和mapMutations()存在相同的问题就是阻碍自身自身计算属性的定义和扩展，如何解决这个问题一样使用解构方式，如下：

解构方式：

```
import {mapMutations, mapState} from 'vuex'
computed: {
    othercomputed(){/*启动的计算属性*/},
    ...mapState({
        books: function (state) {
            return state.items;
        }
    })
},
```

13、Vuex的getter

定义过滤和公共行为的函数和逻辑。有点数据的加工和通用逻辑的获取。你可以这样理解和认为，getters是store状态管理的计算属性，与计算属性一样， getter的返回值会根据他的依赖项缓存起来，且只有在它的依赖项发生改变时才会重新计算。

> 一句话：使用*getter*定义方法是当做属性来使用，和*computed*一模一样。只不过是状态管理中计算

在store/index.js

```
import {createStore} from "vuex"
import books from "@/mock/books";
import {DELETE_ITEMS, PUSH_BOOK_TO_CART} from
'./mutation-types'

const store = createStore({
    strict: true,
    state() {
        return {
            // 导入数据进行初始化
            items: books
        }
    },

    mutations: {
        // 可以使用es2015的计算属性命名功能来使用一个常量作为
函数名，达到一个维护更方便的目的
        [PUSH_BOOK_TO_CART](state, book) {
            state.items.push(book);
        },

```

```
        [DELETE_ITEMS](state, id) {
            // 根据提交的id。查询是否存在相同的商品id。返回商品的索引
            let index = state.items.findIndex(item => item.id === id);
            if (index >= 0) {
                state.items.splice(index, 1);
            }
        }
    },

    // 公共的行为和计算属性
    getters: {
        // 计算总价
        totalPrice(state) {
            return state.items.reduce((total, item) => {
                return total + item.price * item.count;
            }, 0)
        },
        // 计算小计
        countPrice(state) {
            return function (id) {
                let citem = state.items.find(item => item.id === id);
                return citem ? citem.price * citem.count : 0;
            }
        },
        // 获取已售卖的书籍
        sellingBooks(state) {
            return state.filter(item => item.isSale === true)
```

```
        }
    }

})


export default store;
```

在book/index.vue的 spa页面中引入

```
<template>
  <div>
    <table>商品编号</table>
    <table>
      <tr>
        <td>商品编号：</td>
        <td><input type="text" v-
model.number="book.id"></td>
      </tr>
      <tr>
        <td>商品标题：</td>
        <td><input type="text" v-
model.number="book.title"></td>
      </tr>
      <tr>
        <td>商品价格：</td>
        <td><input type="text" v-
model.number="book.price"></td>
      </tr>
      <tr>
        <td colspan="2">
          <button @click="addCarts">添加购物车</button>
        </td>
      </tr>
```

```
22      </table>
23      <hr>
24      <table>
25        <thead>
26        <tr>
27          <th>编号</th>
28          <th>商品名称</th>
29          <th>价格</th>
30          <th>数量</th>
31          <th>金额</th>
32          <th>操作</th>
33        </tr>
34        </thead>
35        <tbody>
36        <tr v-for="(book,index) in books"
  :key="book.id">
37          <td>{{ book.id }}</td>
38          <td>{{ book.title }}</td>
39          <td>{{ book.price }}</td>
40          <td>
41            <button @click="add(index)">+</button>
42            {{ book.count }}
43            <button @click="minus(index)">-</button>
44          </td>
45          <td>{{ countPrice(book.id) }}</td>
46          <td>
47            <button @click="deleteCart(book.id)">删除
  </button>
48          </td>
49        </tr>
50        </tbody>
51      </table>
52      <div>总价：{{totalPrice}}</div>
53    </div>
```

```
</template>

<script>
import {DELETE_ITEMS, PUSH_BOOK_TO_CART} from
'@/store/mutation-types'
import {mapGetters, mapMutations, mapState} from
'vuex'

export default {
  name: "index",
  data() {
    return {
      book: {
        id: "",
        title: "",
        price: "",
        count: 1
      }
    }
  },
  computed: {
    ...mapState({
      books: function (state) {
        return state.items;
      }
    }),
    ...mapGetters({
      itemPrice: 'countItemPrice',
      totalPrice: 'totalItemPrice'
    })
  },
  methods: {
    ...mapMutations({
      addCarts: PUSH_BOOK_TO_CART,
```

```
86          deleteCart: DELETE_ITEMS
87        })
88    }
89  }
90  </script>
91
92  <style scoped>
93  table {
94    width: 100%;
95    border-collapse: collapse;
96  }
97  </style>
```

当然你也可以直接定义和调用，在book/index.vue中如下：

直接调用方式

```
1  computed:{
2     sellingbooks(){
3          return this.$store.getters.sellingbooks;
4     }
5  }
```

数组解构方式

```
1  computed:{
2     ...mapGetters([
3         'sellingbooks'
4     ])
5  }
6
```

对象解构方式

```
computed:{
    ...mapGetters({
      csellingbooks:"sellingbooks"
    })
}
```

14、Vuex的Action

在定义mutation时，一条重要的原则就是mutation必须是同步函数，换句话说，在muations()处理函数中，不能存在异步的调用。例如：

```
mutations: {
    // 可以使用es2015的计算属性命名功能来使用一个常量作为函数名，达到一个维护更方便的目的
    [PUSH_BOOK_TO_CART](state, book) {
        setTimeout(() => {
            state.items.push(book);
        }, 1000)
    },
}
```

这个时候在去执行添加商品打购物车就会出现，没有反应的响应。因为mutations中不能存在异步执行的逻辑和处理，怎么处理和解决mutations不能执行异步更改state数据的问题，出现了Action.

如果确实要执行异步操作，那么应该使用action。action类似于mutation，不同之处在于：

- 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。

一个简单的Action定义如下：

```
import {createStore} from "vuex"
import books from "@/mock/books";
import {DELETE_ITEMS, PUSH_BOOK_TO_CART} from
'./mutation-types'

const store = createStore({
    strict: true,
    state() {
        return {
            // 导入数据进行初始化
            items: books
        }
    },

    mutations: {
        // 可以使用es2015的计算属性命名功能来使用一个常量作为
函数名，达到一个维护更方便的目的
        [PUSH_BOOK_TO_CART](state, book) {
            state.items.push(book);
        },

        [DELETE_ITEMS](state, id) {
            // 根据提交的id。查询是否存在相同的商品id。返回商
品的索引
            let index = state.items.findIndex(item =>
item.id === id);
            if (index >= 0) {
                state.items.splice(index, 1);
            }
        },
```

```
27
28          increment(state, index) {
29              state.items[index].count++;
30          },
31
32          minus(state, index) {
33              state.items[index].count--;
34          }
35      },
36
37      // 公共的行为和计算属性
38      getters: {
39
40          // 计算总价
41          totalItemPrice(state) {
42              return state.items.reduce((total, item) =>
    {
43                  return total + item.price *
    item.count;
44              }, 0)
45          },
46
47          // 计算小计
48          countItemPrice(state) {
49              return function (id) {
50                  let citem = state.items.find(item =>
    item.id === id);
51                  return citem ? citem.price *
    citem.count : 0;
52              }
53          },
54
55          // 获取已售卖的书籍
56          sellingBooks(state) {
```

```
            return state.filter(item => item.isSale === true)
        }
    },

    actions: {
        increment(context,index) {
            setTimeout(()=>{
                context.commit("increment",index);
            },1000)
        },
        minus({commit},index) {
            commit("minus",index)
        }
    }

})


export default store;
```

引用：book/index.vue

```
<template>
  <div>
    <table>商品编号</table>
    <table>
      <tr>
        <td>商品编号：</td>
        <td><input type="text" v-model.number="book.id"></td>
      </tr>
      <tr>
        <td>商品标题：</td>
```

```
11          <td><input type="text" v-
    model.number="book.title"></td>
12        </tr>
13        <tr>
14          <td>商品价格：</td>
15          <td><input type="text" v-
    model.number="book.price"></td>
16        </tr>
17        <tr>
18          <td colspan="2">
19            <button @click="addCarts">添加购物车</button>
20          </td>
21        </tr>
22      </table>
23      <hr>
24      <table>
25        <thead>
26        <tr>
27          <th>编号</th>
28          <th>商品名称</th>
29          <th>价格</th>
30          <th>数量</th>
31          <th>金额</th>
32          <th>操作</th>
33        </tr>
34        </thead>
35        <tbody>
36        <tr v-for="(book,index) in books"
    :key="book.id">
37          <td>{{ book.id }}</td>
38          <td>{{ book.title }}</td>
39          <td>{{ book.price }}</td>
40          <td>
41            <button @click="add(index)">+</button>
```

```
          {{ book.count }}
          <button @click="minus(index)">-</button>
        </td>
        <td>{{ itemPrice(book.id) }}</td>
        <td>
          <button @click="deleteCart(book.id)">删除</button>
        </td>
      </tr>
      </tbody>
    </table>
    <div>总价：{{totalPrice}}</div>
  </div>
</template>

<script>
import {DELETE_ITEMS, PUSH_BOOK_TO_CART} from '@/store/mutation-types'
import {mapGetters, mapMutations, mapState} from 'vuex'

export default {
  name: "index",
  data() {
    return {
      book: {
        id: "",
        title: "",
        price: "",
        count: 1
      }
    }
  },
  computed: {
```

```
    ...mapState({
      books: function (state) {
        return state.items;
      }
    }),
    ...mapGetters({
      itemPrice: 'countItemPrice',
      totalPrice: 'totalItemPrice'
    })
  },
  methods: {
    ...mapMutations({
      addCarts: PUSH_BOOK_TO_CART,
      deleteCart: DELETE_ITEMS
    }),

    add(index){
      this.$store.dispatch("increment",index);
    },

    minus(index){
      this.$store.dispatch("minus",index);
    }
  }
}
</script>

<style scoped>
table {
  width: 100%;
  border-collapse: collapse;
}
</style>
```

注意上面可以得出结论：

- action是一个动作，不能直接修改state状态的数据，只能通过上下文context参数提交mutations的函数、从而达到更改状态的目的。
- action定义的动作，可以用结构的方式直接结构commit，比如

```
minus({commit},index) {
    commit("minus",index)
}
```

- action动作的执行和分发是

```
this.$store.dispatch("action函数名",参数);
```

- action可以执行异步处理，没有任何问题

```
actions: {
    increment(context,index) {
        setTimeout(()=>{
            context.commit("increment",index);
        },1000)
    }
}
```

  可以看出来使用setTimeout模拟异步也能够完成同步的操作。状态也可以修改。

action动作和其他的mutation,getter一样提供了mapActions()辅助函数。可以将组件的方法映射为store.dispatch调用，非常的方便，如下：

数组方式：

```
import {mapActions} from 'vuex'
  methods: {
    //数组的方式，这样就必须同名调用了
    ...mapActions(['increment','minus'])
  }

```

对象方式：

```
import {mapActions} from 'vuex'
  methods: {
    // 对象的方式，可以更名操作,更加灵活
    ...mapActions({
      add: 'increment',
      minus: 'minus'
    }),
  }

```

实现购物车，商品存在就追加数量，不存在就添加

```
import {createStore} from "vuex"
import books from "@/mock/books";
import {DELETE_ITEMS, PUSH_BOOK_TO_CART} from
'./mutation-types'

const store = createStore({
    strict: true,
    state() {
        return {
            // 导入数据进行初始化

```

```
            items: books
        }
    },

    mutations: {
        // 可以使用es2015的计算属性命名功能来使用一个常量作为函数名，达到一个维护更方便的目的
        [PUSH_BOOK_TO_CART](state, book) {
            state.items.push(book);
        },

        [DELETE_ITEMS](state, id) {
            // 根据提交的id。查询是否存在相同的商品id。返回商品的索引
            let index = state.items.findIndex(item => item.id === id);
            if (index >= 0) {
                state.items.splice(index, 1);
            }
        },

        increment(state, index) {
            state.items[index].count++;
        },

        minus(state, index) {
            state.items[index].count--;
        }
    },

    // 公共的行为和计算属性
    getters: {

        // 计算总价
```

```
41          totalItemPrice(state) {
42              return state.items.reduce((total, item) =>
    {
43                  return total + item.price *
    item.count;
44              }, 0)
45          },
46
47          // 计算小计
48          countItemPrice(state) {
49              return function (id) {
50                  let citem = state.items.find(item =>
    item.id === id);
51                  return citem ? citem.price *
    citem.count : 0;
52              }
53          },
54
55          // 获取已售卖的书籍
56          sellingBooks(state) {
57              return state.filter(item => item.isSale
    === true)
58          }
59      },
60
61      actions: {
62
63          addCarts(context, book) {
64              let index =
    context.state.items.findIndex(item => item.id ===
    book.id);
65              if (index != -1) {
66                  context.commit("increment", index);
67              } else {
```

```
                context.commit("pushBookToCart", book);
            }
        },

        // 递增
        increment(context, index) {
            // 可能执行异步处理修改redis缓存
            setTimeout(() => {
                context.commit("increment", index);
            }, 1000)
        },

        // 递减
        minus({commit}, index) {
            // 可能执行异步处理修改redis缓存
            commit("minus", index)
        },

        // 做购物车结算
        checkout() {
            // 1: 异步保存商品信息入口
            // 2: 发起异步请求，并清空购物车
            // 3: 开始结算生成订单，

        }
    }

})


export default store;
```

book/index.vue

```
<template>
  <div>
    <table>商品编号</table>
    <table>
      <tr>
        <td>商品编号：</td>
        <td><input type="text" v-model.number="book.id"></td>
      </tr>
      <tr>
        <td>商品标题：</td>
        <td><input type="text" v-model.number="book.title"></td>
      </tr>
      <tr>
        <td>商品价格：</td>
        <td><input type="text" v-model.number="book.price"></td>
      </tr>
      <tr>
        <td colspan="2">
          <button @click="addCarts">添加购物车</button>
        </td>
      </tr>
    </table>
    <hr>
    <table>
      <thead>
      <tr>
        <th>编号</th>
        <th>商品名称</th>
        <th>价格</th>
        <th>数量</th>
        <th>金额</th>
```

```
32          <th>操作</th>
33        </tr>
34        </thead>
35        <tbody>
36        <tr v-for="(book,index) in books"
:key="book.id">
37          <td>{{ book.id }}</td>
38          <td>{{ book.title }}</td>
39          <td>{{ book.price }}</td>
40          <td>
41            <button @click="add(index)">+</button>
42            {{ book.count }}
43            <button @click="minus(index)">-</button>
44          </td>
45          <td>{{ itemPrice(book.id) }}</td>
46          <td>
47            <button @click="deleteCart(book.id)">删除
</button>
48          </td>
49        </tr>
50        </tbody>
51      </table>
52      <div>总价: {{ totalPrice }}</div>
53    </div>
54  </template>
55
56  <script>
57  import {DELETE_ITEMS, PUSH_BOOK_TO_CART} from
'@/store/mutation-types'
58  import {mapActions, mapGetters, mapMutations,
mapState} from 'vuex'
59
60  export default {
61    name: "index",
```

```
62    data() {
63      return {
64        book: {
65          id: "",
66          title: "",
67          price: "",
68          count: 1
69        }
70      }
71    },
72    computed: {
73      ...mapState({
74        books: function (state) {
75          return state.items;
76        }
77      }),
78      ...mapGetters({
79        itemPrice: 'countItemPrice',
80        totalPrice: 'totalItemPrice'
81      })
82    },
83    methods: {
84      ...mapMutations({
85        //addCarts: PUSH_BOOK_TO_CART,
86        deleteCart: DELETE_ITEMS
87      }),
88
89      //数组的方式，这样就必须同名调用了
90      //...mapActions(['increment','minus']),
91      // 对象的方式，可以更名操作,更加灵活
92      ...mapActions({
93        add: 'increment',
94        addCarts:function(dispatch){
95          dispatch("addCarts",this.book)
```

```
96        },
97        minus: 'minus'
98      }),
99    }
100 }
101 </script>
102
103 <style scoped>
104 table {
105   width: 100%;
106   border-collapse: collapse;
107 }
108 </style>
```

15、vuex-persist数据持久化存储插件

官网：https://www.npmjs.com/package/vuex-persist

> *vuex* 解决了多视图之间的数据共享问题。但是数据并不能持久化，只要一刷新页面，你存储在 *Vuex* 中的 *store* 里的数据就丢失了。

引入vuex-persist 插件，它就是为 Vuex 持久化存储而生的一个插件。不需要你手动存取 storage ，而是直接将状态保存至 cookie 或者 localStorage 中。

安装：

```
npm install --save vuex-persist

or
yarn add vuex-persist
```

引入：

先创建一个对象并进行配置：

```

import VuexPersistence from 'vuex-persist'
const vuexLocal = new VuexPersistence({
    storage: window.localStorage
})
```

引入进**vuex**插件：

```
const store = new Vuex.Store({
  state: { ... },
  mutations: { ... },
  actions: { ... },
  plugins: [vuexLocal.plugin]
})
```

通过以上设置，在图3中各个页面之间跳转，如果刷新某个视图，数据并不会丢失，依然存在，并且不需要在每个 mutations 中手动存取 storage 。

vuex-persist 的详细属性：

属性	类型	描述
key	string	将状态存储在存储中的键。默认: 'vuex'

属性	类型	描述
storage	Storage (Web API)	可传localStorage, sessionStorage, localforage 或者你自定义的存储对象. 接口必须要有get和set. 默认是: **window.localStorage**
saveState	function (key, state[, storage])	如果不使用存储，这个自定义函数将保存状态保存为持久性。
restoreState	function (key[, storage]) => state	如果不使用存储，这个自定义函数处理从存储中检索状态
reducer	function (state) => object	将状态减少到只需要保存的值。默认情况下，保存整个状态。
filter	function (mutation) => boolean	突变筛选。看mutation.type并返回true，只有那些你想坚持写被触发。所有突变的默认返回值为true。
modules	string[]	要持久化的模块列表。

15、Vuex的Module

Vuex使用单一状态树，应用程序的所有状态都包含在一个大的对象中。当应用变得复杂时，store对象就会变得非常的臃肿。

为了解决这个问题，Vuex允许讲store划分为多个模块，每个模块都可以包含自己的state,mutations,actions，getters以及嵌套的子模块，例如：

```
const moduleA = {
  state: () => ({ ... }),
  mutations: { ... },
```

```
  actions: { ... },
  getters: { ... }
}

const moduleB = {
  state: () => ({ ... }),
  mutations: { ... },
  actions: { ... }
}

const store = new Vuex.Store({
  modules: {
    a: moduleA,
    b: moduleB
  }
})

store.state.a // -> moduleA 的状态
store.state.b // -> moduleB 的状态
```

模块的局部状态

对于模块内部的 mutation 和 getter，接收的第一个参数是模块的局部状态对象。

```
const moduleA = {
  state: () => ({
    count: 0
  }),
  mutations: {
    increment (state) {
```

```
      // 这里的 `state` 对象是模块的局部状态
      state.count++
    }
  },

  getters: {
    doubleCount (state) {
      return state.count * 2
    }
  }
}
```

同样，对于模块内部的 action，局部状态通过 `context.state` 暴露出来，根节点状态则为 `context.rootState`：

```
const moduleA = {
  // ...
  actions: {
    incrementIfOddOnRootSum ({ state, commit, rootState }) {
      if ((state.count + rootState.count) % 2 === 1) {
        commit('increment')
      }
    }
  }
}
```

对于模块内部的 getter，根节点状态会作为第三个参数暴露出来：

```
const moduleA = {
  // ...
  getters: {
    sumWithRootCount (state, getters, rootState) {
      return state.count + rootState.count
    }
  }
}
```

详见查看视频