GERENCIAMENTO DE ESTADO COM VUEX

1. Aula 280 – Porque usar um gerenciador de Estado
2. O vuex é uma ferramento utiliza pelo Vue para intermediar as comunicações realizadas entre componentes, uma vez que quando o projeto cresce as comunicações entre componentes, filhos, netos etcc. Ficam mais difíceis, necessitando de estratégias como props, eventos personalizados($emit()), event Bus (barramento) etc.
3. O Vuex irá facilitar essas comunicações
4. Aula 281 – Baixe o projeto
5. Baixe o projeto e instale o vuex: npm i –save [vuex@3.0.1](mailto:vuex@3.0.1) -E, isso para instalar exatamente a versão 3.0.1
6. Nesse projeto teremos 4 componentes, Loja, Parametros, Carrinho, Resumo e um componente padrão Panel que será declarado globalmente e servirá de base para receber os conteúdos html de todos os componentes.
7. Componemte Panel.
8. <template>
9. <div *class*="painel">
10. <div *class*="cabecalho">
11. <span *class*="titulo">{{ titulo }}</span>
12. <span *class*="notificacao" v-if="notificacao">
13. {{ notificacao }}
14. </span>
15. </div>
16. <div *class*="conteudo">
17. <slot></slot>
18. </div>
19. </div>
20. </template>
21. <script>
22. export default {
23. props: ['titulo', 'notificacao']
24. }
25. </script>
26. <style *scoped*>
27. *.painel* {
28. flex: 1;
29. display: flex;
30. flex-direction: column;
31. margin: 5px;
32. }
33. *.cabecalho* {
34. display: flex;
35. justify-content: center;
36. }
37. *.titulo* {
38. flex: 1;
39. padding: 5px;
40. }
41. *.conteudo* {
42. padding: 25px;
43. }
44. *.notificacao* {
45. display: flex;
46. justify-content: center;
47. align-items: center;
48. font-size: 1.6rem;
49. font-weight: 600;
50. color: #FFF;
51. background-color: #e53935;
52. width: 60px;
53. }
54. *.painel*[*vermelho*] *.cabecalho* { color: #FFF; background-color: #c62828; }
55. *.painel*[*vermelho*] *.conteudo* { background-color: #f44336; }
56. *.painel*[*verde*] *.cabecalho* { color: #FFF; background-color: #2E7D32; }
57. *.painel*[*verde*] *.conteudo* { background-color: #4CAF50; }
59. *.painel*[*azul*] *.cabecalho* { color: #FFF; background-color: #1565C0; }
60. *.painel*[*azul*] *.conteudo* { background-color: #2196F3; }
61. *.painel*[*roxo*] *.cabecalho* { color: #FFF; background-color: #6A1B9A; }
62. *.painel*[*roxo*] *.conteudo* { background-color: #9C27B0; }
63. </style>
64. Componente App
65. <template>
66. <div *id*="app">
67. <h1>Exercícios Vuex</h1>
68. <div *class*="linha">
69. <Parametros />
70. <Loja />
71. </div>
72. <div *class*="linha">
73. <Carrinho />
74. <Resumo />
75. </div>
76. </div>
77. </template>
78. <script>
79. import Parametros from './components/Parametros'
80. import Loja from './components/Loja'
81. import Carrinho from './components/Carrinho'
82. import Resumo from './components/Resumo'
83. export default {
84. name: 'app',
85. components: { Parametros, Loja, Carrinho, Resumo }
86. }
87. </script>
88. <style>
89. \* {
90. font-family: 'Oswald', sans-serif;
91. }
92. body {
93. font-size: 2rem;
94. background: linear-gradient(to right, rgb(0, 0, 0), rgb(67, 67, 67));
95. }
96. *#app* {
97. *-webkit-font-smoothing*: antialiased;
98. *-moz-osx-font-smoothing*: grayscale;
99. text-align: center;
100. color: #FFF;
101. }
102. h1 {
103. font-weight: 200;
104. }
105. *.linha* {
106. display: flex;
107. }
108. button {
109. font-size: 1.8rem;
110. padding: 5px 15px;
111. border-radius: 5px;
112. color: #222;
113. background-color: #FAFAFA;
114. outline: none;
115. border: none;
116. }
117. </style>
118. Componente Parametros
119. <template>
120. <Painel *titulo*="Parâmetros" *vermelho*>
121. <div *class*="parametros">
122. <span>
123. <strong>Quantidade Padrão: </strong>
124. <input *type*="number" *v-model*="quantidade">
125. </span>
126. <span>
127. <strong>Preço Padrão: </strong>
128. <input *type*="number" *v-model*="preco">
129. </span>
130. </div>
131. </Painel>
132. </template>
133. <script>
134. export default {
135. data() {
136. return {
137. quantidade: 0,
138. preco: 0
139. }
140. }
141. }
142. </script>
143. <style>
144. *.parametros* {
145. display: flex;
146. justify-content: space-around;
147. }
148. </style>
149. Componente Loja
150. <template>
151. <Painel *titulo*="Loja Virtual" *verde*>
152. <div *class*="loja">
153. <span>Adicionar</span>
154. <input *type*="number" *v-model*.*number*="quantidade">
155. <span>itens de <strong>R$</strong></span>
156. <input *type*="number" *v-model*.*number*="preco">
157. <button @*click*="adicionar">Agora!</button>
158. </div>
159. </Painel>
160. </template>
161. <script>
162. export default {
163. data() {
164. return {
165. sequencia: 1,
166. quantidade: 1,
167. preco: 9.99,
168. }
169. },
170. methods: {
171. adicionar() {
172. const produto = {
173. id: *this*.sequencia,
174. nome: `Produto ${*this*.sequencia}`,
175. quantidade: *this*.quantidade,
176. preco: *this*.preco
177. }
178. *this*.sequencia++
179. // eslint-disable-next-line
180. console.log(produto)
181. }
182. }
183. }
184. </script>
185. <style>
186. *.loja* {
187. display: flex;
188. justify-content: center;
189. }
190. *.loja* > \* {
191. margin: 0px 10px;
192. }
193. input {
194. font-size: 2rem;
195. width: 90px;
196. }
197. </style>
198. Componente Carrinho
199. <template>
200. <Painel *titulo*="Carrinho" *azul* :*notificacao*="produtos.length">
201. <div *class*="carrinho">
202. <table>
203. <thead>
204. <tr>
205. <th>Nome</th>
206. <th>Qtde</th>
207. <th>Preço</th>
208. </tr>
209. </thead>
210. <tbody>
211. <tr v-for="produto in produtos" :*key*="produto.nome">
212. <td>{{ produto.nome }}</td>
213. <td>{{ produto.quantidade }}</td>
214. <td>{{ produto.preco | dinheiro }}</td>
215. </tr>
216. </tbody>
217. </table>
218. <hr>
219. <div *class*="total">
220. <span>Total: <strong>{{ total | dinheiro }}</strong></span>
221. </div>
222. </div>
223. </Painel>
224. </template>
225. <script>
226. export default {
227. computed: {
228. total() {
229. return *this*.produtos.map(*p* => *p*.quantidade \* *p*.preco)
230. .reduce((*total*, *atual*) => *total* + *atual*, 0)
231. }
232. },
233. data() {
234. return {
235. produtos: [
236. { id: 1, nome: 'Produto 1', quantidade: 7, preco: 14.55 },
237. { id: 2, nome: 'Produto 2', quantidade: 10, preco: 22.99 },
238. { id: 3, nome: 'Produto 3', quantidade: 1, preco: 43.18 },
239. ]
240. }
241. }
242. }
243. </script>
244. <style>
245. table {
246. width: 100%;
247. }
248. td {
249. border-top: 1px solid #EEE;
250. width: 33%;
251. }
252. hr {
253. margin-top: 30px;
254. }
255. *.total* {
256. display: flex;
257. justify-content: flex-end;
258. }
259. </style>
260. Componente Resumo
261. <template>
262. <Painel *titulo*="Resumo" *roxo*>
263. <div *class*="resumo">
264. <span>Total: <strong>{{ total | dinheiro }}</strong></span>
265. <hr>
266. <button>Finalizar!</button>
267. </div>
268. </Painel>
269. </template>
270. <script>
271. export default {
272. computed: {
273. total() {
274. return *this*.produtos.map(*p* => *p*.quantidade \* *p*.preco)
275. .reduce((*total*, *atual*) => *total* + *atual*, 0)
276. }
277. },
278. data() {
279. return {
280. produtos: [
281. { nome: 'Produto 1', quantidade: 7, preco: 14.55 },
282. { nome: 'Produto 2', quantidade: 10, preco: 22.99 },
283. { nome: 'Produto 3', quantidade: 1, preco: 43.18 },
284. ]
285. }
286. }
287. }
288. </script>
289. <style>
290. table {
291. width: 100%;
292. }
293. td {
294. border-top: 1px solid #EEE;
295. width: 33%;
296. }
297. </style>
298. C
299. Como podermos observar nos componentes Resumo e Carrinho, temos uma duplicação da propriedade computada e dos dados em data().
300. Para resolver isso iremos criar uma pasta dentro de src chamada store e um arquivo chamado store.js
301. Dentro de store.js iremos importar o Vue e o Vuex
302. E iremos registrar o vues dentro do vue.
303. Vamos exportar uma função construtora do Vuex, chamada Store, ela terá um parâmetro chamado state:{} que é o estado central gerenciado pelo Vuex.
304. Como temos uma lista que irá mudar em nossos componentes, iremos inserir essa lista em nosso estado central da aplicação
305. import Vue from 'vue'
306. import Vuex from 'vuex'
307. Vue.use(Vuex)
308. export default **new** Vuex.Store({
309. state:{
310. produtos:[
311. { id: 1, nome: 'Produto 1', quantidade: 7, preco: 14.55 },
312. { id: 2, nome: 'Produto 2', quantidade: 10, preco: 22.99 },
313. { id: 3, nome: 'Produto 3', quantidade: 1, preco: 43.18 },
314. ]
315. }
316. })
317. Feito isso vamos importar e registrar nosso arquivo store.js no arquivo main.js.
318. import Vue from 'vue'
319. import App from './App.vue'
320. import store from './store/store'
321. Vue.config.productionTip = false
322. import Painel from './components/Painel'
323. Vue.component('Painel', Painel)
324. Vue.filter('dinheiro', *valor* => {
325. return `R$ ${parseFloat(*valor*).toFixed(2)}`.replace('.', ',')
326. })
327. **new** Vue({
328. store,
329. render: *h* => h(App),
330. }).$mount('#app')
331. Agora vamos alterar o componente Carrinho apagando o campo data() e criando uma propriedade computada que irá trazer os dados que estão no state de store.
332. export default {
333. computed: {
334. total() {
335. return *this*.produtos.map(*p* => *p*.quantidade \* *p*.preco)
336. .reduce((*total*, *atual*) => *total* + *atual*, 0)
337. },
338. produtos(){
339. return *this*.$store.state.produtos
340. }
341. }
342. }
343. No componente Loja vamos alterar o método adicionar() para incluirmos produtos no array do arquivo store.js.
344. adicionar() {
345. *this*.sequencia = *this*.$store.state.produtos.length
346. *this*.sequencia++
348. const produto = {
349. id: *this*.sequencia,
350. nome: `Produto ${*this*.sequencia}`,
351. quantidade: *this*.quantidade,
352. preco: *this*.preco
353. }
355. *this*.$store.state.produtos.push(produto)
356. // eslint-disable-next-line
357. // console.log(produto)
358. }
359. }
360. **Aula 283 – Porque estado centralizado sozinho não resolve**
361. Vemos que no carrinho e no resumo existe uma propriedade computada que está se repetindo, vamos resolver esse problema, centralizando essa propriedade.
362. **Aula 284 – Entendendo Getters**
363. É uma função que lê uma informação e retorna um resultado.
364. **Aula 285 – Usando Getters**
365. Vamos criar nosso Getter dentro de store.js
366. Iremos utilizar o atributo getters: iremos criar uma função chamada valorTotal que irá receber o state e em cima do state iremos pegar os produtos, fazer um map para multiplicar a quantidade de produto pelo preço e um reduce que irá retornar a soma dos valores de map.
367. getters: {
368. valorTotal(*state*){
369. return *state*.produtos.map(*p* => *p*.quantidade \* *p*.preco)
370. .reduce((*total*, *atual*)=> *total* + *atual*, 0)
371. }
372. }
373. No componente resumo agora não precisamos mais da lista de produtos para realizar o calculo total, uma vez que ele já é passado pelo getters do store.
374. export default {
375. computed: {
376. total() {
377. return *this*.$store.getters.valorTotal
378. }
379. }
380. }

Aula 286 – Mapeando getters para propriedade.

1. Temos outra forma de acessarmos os getters que é através da função mapGetters.
2. Para exemplificar vamos comentar a propriedade computada no componente Resumo.
3. Vamos importar a função mapGetters e dentro da propriedade computada, nós iremos chamar a função valorTotal que está dentro de um getters no store.js
4. export default {
5. // computed: {
6. //     // total() {
7. //     //     return this.$store.getters.valorTotal
8. //     // }
9. // }
10. computed: mapGetters(['valorTotal'])
11. }
12. Mas dessa forma, temos que altear a varial total do template por ‘valorTotal’
13. <span>Total: <strong>{{ valoTotal | dinheiro }}</strong></span>
14. Para não termos que realizar essa alteração, podemos modificar a propriedade computada incluindo um atributo que será chamado total.
15. computed: mapGetters ({
16. total: 'valorTotal'
17. })

**Aula 288 – Entendendo Mutations**

1. Mutations são métodos similares ao getters porém são utilizados para alterar o state do store.js.

**Aula 289 – Usando Mutations**

1. No arquivo store.js vamos utilizar a propriedade mutations: incluindo um método chamado adicionarProduto() que terá o parâmetro state onde estão os dados que serão alterados e outro parâmetro será a informação que será alterada, nesse caso produto.
2. mutations: {
3. adicionarProdutos(*state*, *produto*){
4. *state*.produtos.push(*produto*)
5. }
6. }
7. Para podermos utilizar a mutation devemos utilizar o método de $store chamado commit, passando para ele a mutation e a variável que será alterada.
8. adicionar() {
9. *this*.sequencia = *this*.$store.state.produtos.length
10. *this*.sequencia++
12. const produto = {
13. id: *this*.sequencia,
14. nome: `Produto ${*this*.sequencia}`,
15. quantidade: *this*.quantidade,
16. preco: *this*.preco
17. }
19. // this.$store.state.produtos.push(produto)
20. // eslint-disable-next-line
21. // console.log(produto)
22. *this*.$store.commit('adicionarProduto', produto)
23. }
24. }
25. Outra forma de utilizarmos mutation é importar para o componente a função mapMutations do vuex e fazermos um spreding de mapMutations passando o método definido em mutations do store, desta forma podemos acessar o método diretamente.
26. import { mapMutations } from 'vuex'
27. export default {
28. data() {
29. return {
30. sequencia: 1,
31. quantidade: 1,
32. preco: 9.99,
33. }
34. },
35. methods: {
36. ...mapMutations(['adicionarProduto']),
37. adicionar() {
38. *this*.sequencia = *this*.$store.state.produtos.length
39. *this*.sequencia++
41. const produto = {
42. id: *this*.sequencia,
43. nome: `Produto ${*this*.sequencia}`,
44. quantidade: *this*.quantidade,
45. preco: *this*.preco
46. }
48. // this.$store.state.produtos.push(produto)
49. // eslint-disable-next-line
50. // console.log(produto)
51. //    this.$store.commit('adicionarProduto', produto)
52. *this*.adicionarProduto(produto)
53. }
54. }
55. A vantagem de utilizar o mutation é que se necessário poderemos ter alguma lógica antes de realizar a alteração ou inclusão do produto.

**Aula 291 – Como Actions Complementam as Mutation**

1. A multation tem como finalidade realizar alterações no state, o action é utilizada para registrar regras ou uma lógica de acesso ou permissão de alteração no state do store. Dentro das actions chamamos uma ou mais mutations,

**Aula 292 – Usando Actions**

1. No store.js vamos incluir um novo atributo actions:{}
2. Vamos criar o método adicionarProduto que receberá um contexto e o objeto ou variável que será alterada no state.
3. Vamos inserir dentro de adicionarProduto um setTimeout(()=>{},time)
4. actions:{
5. adicionarProduto(*context*, *produto*){
6. setTimeout(() => {
7. *context*.commit('adicionarProduto', *produto*)
8. }, 1000);
9. }
10. }
11. No componente Loja vamos alterar o mapMutations para mapActions.
12. methods: {
13. // ...mapMutations(['adicionarProduto']),
14. ...mapActions(['adicionarProduto']),
15. adicionar() {
16. *this*.sequencia = *this*.$store.state.produtos.length
17. *this*.sequencia++
19. const produto = {
20. id: *this*.sequencia,
21. nome: `Produto ${*this*.sequencia}`,
22. quantidade: *this*.quantidade,
23. preco: *this*.preco
24. }
26. // this.$store.state.produtos.push(produto)
27. // eslint-disable-next-line
28. // console.log(produto)
29. //    this.$store.commit('adicionarProduto', produto)
30. *this*.adicionarProduto(produto)
31. }
32. }