一、Vuex是什么

集中式的管理组件里的数据，管理数据的运用状态，当组件里面有交互的时候，有一种规则去改变这种状态，从而以预测的方式发生变化

1、状态

组件内部状态：仅在一个组件内部使用和改变的数据（data字段）

应用级别的状态：多个组件共用的状态

2、什么情况下使用Vuex

多个视图依赖于同一个状态，（如果是同一级别的，可以不使用Vuex，可以通过父组件传到每个组件里面去）

来自不同视图的行为需要变更同一状态（一个组件有交互去改变另一个组件的数据（状态），不是直接操纵数据来改变，而是通过改变Vuex里面的数据，Vuex里是响应的，进而改变其他用到这个数据的组件）

二、兄弟组件共享数据问题 (案列：仿select)（不使用Vuex）

1、给个显示隐藏的状态

select.vue:

定义一个状态 aa: false

v-show = "aa"

2、点击显示和隐藏

selectInput.vue:

·父组件改变状态，子组件也改变状态（通信问题），自定义事件实现通信，同时实现双向数据绑定（父组件和子组件同时改变）

·自定义属性 :is-show = "aa" 传给子组件，这个时候需要加上标识 :is-show.sync = "aa" ，因为是双向数据绑定，需要同步的；

·然后在子组件绑定一个点击事件bb,bb写在methods里面，点击一下就要改变aa的值,那么只需要改变is-show的值就行。

·is-show我们怎么通知父组件去改变的呢？只需要在点击的时候触发一个事件，这个事件也是自定义事件：this.$emit("update:isShow",true)

update是vue内部去定义的，需要更新哪个，就写哪个(拿到key值就行，这里是is-show)

第二个参数是要传给他改变的值，这里是改变成一个true

·但是点击一下是显示，再点击是隐藏，所以不能写死成true或false，我们需要再多声明一个数据来改变它；

先把值拿到：props: ['isShow']

然后在计算属性里拿到初始状态：如果this.isShow是true，initShow的值也是true

computed: {

initShow(){

return this.isShow

}

}

那么在点击的时候，第二个参数我们只需要拿到initShow的值，然后取反,再传给父组件就行

!this.initShow

·这样就实现在子组件里面去改变父组件里面的状态了，这是利用双向数据绑定的特性

3、数据渲染到list里

select.vue:

·准备一套数据 listData：渲染list里的数据

·传入到 list.vue 里面：:data = "listData"

list.vue:

·接收父组件传过来的数据：props:['data']

·循环到每个 li 里面去：v-for = "item in data " ({{ item }})

4、点击list里的选项，放在input框里面

当点击每个 li 的数据时，我们就要把它放在 input 框里面，而 selectInput 和 list 是兄弟组件的关系，不是嵌套关系，那解决的思路是，点击li，改变父级的状态，再把父级的状态给到 selectInput；

select.vue:

·在父组件定义一个状态：title:""

·把 title传给 selectInput: :title = "title"

在 list 里面去改变 title , 从而改变父组件的 title 改变，那么selectInput 里面的值就改变了

selectInput.vue:

·拿到字符串：props:['isShow','title']

·在 input 里面动态的绑定 :value = "title"

select.vue:

·通过回调函数的方式（自定义事件）获取 list 里面的 li 的值

@changTitle = "titleHand"

·在methods里面定义一下这个事件titleHand：

titleHand （title） {

console.log(title)

this.title = title

}

让它触发这个 changTitle 事件，然后执行 titleHand 这个函数，这个函数接收一个title，拿到title之后就可以改变selectInput 的 title了。

list.vue:

·需要点击li，并把item.title传过来 :

@click = "getTitleHandle(item.title)"

·传过来之后，就要有个事件处理函数（methods里面）：

getTitleHandle(title){

//因为要发给父组件的，所以要触发一下父组件的自定义事件：

this.$emit('changTitle',title)

}

·最终会触发 titleHand 然后拿到值

三、使用Vuex

1、安装Vuex模块：

npm install vuex --save

作为插件使用： Vue.use(Vuex)

定义容器： new Vuex.Store()

注入根实例： { store }

在src下定义一个文件夹store,然后在里面定义一个index.js，Vuex作为插件使用

import Vue from 'vue'

import Vuex from 'vuex'

Vue.use(Vuex)

let store = new Vuex.Store() //定义容器

new Vuex.Store()可以传入一个对象，在对象里面可以写上一些参数，想定义一些状态的话，它给我们提供了一个 state ，state对应的是一个对象，也就是我们所需要的状态数据都放在 state 对象里面就可以了。比如定义一个count：

let store = new Vuex.Store({

state: {

count: 100

}

})

export default store

然后在main.js里, store 注入到 new Vue() 中去，之后就可以用Vuex去管理整个目录的状态了

四、Vuex核心概念

·store：类似容器，包含应用的大部分状态

一个页面只能有一个store

状态存储是响应式的

不能直接改变store中的状态，唯一途径显示地提交mutation

·state：包含所有应用级别状态的对象

·getters：在组件内部获取 store 中的状态的函数

·mutations：唯一修改修改状态事件回调函数

·actions：包含异步操作、提交mutation改变状态

·modules：将 store 分割成不同的模块

五、案例（简易的加法计算器）

运用Vuex的状态去管理这个案例，当在页面中操作加减，对数字进行改变。首先要把状态count的值渲染到页面，然后点击加减的时候去操作状态值就行

html：

<input type="button" value="-" />

<span>0</span>

<input type="button" value="+" />

1、因为是状态改变的时候，span里面的值也改变，所以我们需要写在计算属性里面：

computed: {

num () {

return this.$store.state.count

}

}

this.$store.state.count是从容器里面获取的状态值，然后渲染到span标签里面就行了

2、点击加减的时候改变容器里面的状态值，然后界面要重新渲染

Vuex定义的规则：你想改变容器里面的状态值，就必须要通过一种方式去改变

·如果要修改一个状态，必须要显示的提交一个mutation才行，所以我们需要在容器里面去定义一个mutations，mutations是一个对象，对象里面有N多个函数，函数的作用就是当你触发这个函数的时候，就去改变这个状态的值，这个函数接收一个参数state，用来改变state里面的值

let store = new Vuex.Store({

state: {

count: 100

},

mutations: {

addIncrement(state){

state.count += 1

}

}

})

·然后在组件里面去定义一个事件处理函数addHandle，用来触发addIncrement函数：点击按钮的是触发

<input type="button" value="+" @click = "addHandle" />

在methods中定义函数：

methods:{

addHandle(){

//点击的时候需要改变状态，改变状态时需要提交mutation

//怎么提交？Vuex里有个方法提交：

// $store.commit('addIncrement')

//当点的时候，就把addIncrement提交了，类似于事件对应事件处理函数一样，addIncrement是自定义的事件名字，addHandle函数就是事件处理函数。当我们需要改变状态的时候，就去触发这个事件对应的函数就行。

this.$store.commit('addIncrement')

}

}

·

3、点击提交的时候，我们也可以提交一个参数，通常是作为第二个参数传过去的，比如：5 (实参)

this.$store.commit('addIncrement', 5 )

在接收的时候，自定义事件也接收这个参数，形参为n：

addIncrement(state, n ){

state.count += n

}

如果传入N多个参数的话，可以传入一个对象过去：

this.$store.commit('addIncrement', { n:5 })

而接收的也是一个对象：

addIncrement(state, payload ){

state.count += payload.n

}

4、上面是运用触发一个事件的方式，后面写上我们的参数；实际上还有另一种方式完成，可以使用对象来提交：

this.$store.commit({ })

传入对象的方式，需要告诉他你要传入哪个mutation：type:'addIncrement',下面也是写上参数：n:10

this.$store.commit({

type: 'addIncrement',

n: 10

})

在容器里面接收的时候，第一个参数还是我们的状态，第二个参数就是对象：

addIncrement(state, payload ){

state.count += payload.n

}

5、异步改变状态：点击之后向后端发送请求，然后再回头改变状态

模拟一下：点击之后做个延迟

addIncrement(state, payload ){

setTimeout(()=>{

state.count += payload.n

},1000)

}

这是会出现一个问题：在页面上点击的按钮的时候，页面的值100增加了5，等于105，但是在vue调试工具里面，这条状态的值还是上一次的值；当在页面上快速的点击很多次，页面的值变了很多次，但是在vue调试工具里面，这条状态的值还是100。那么这种状况是怎么回事?

这种问题其实是：每点击一下，就要通过mutation去改变状态，在Vuex里面有一个原则，当提交一个mutation的时候必须是同步的，那就意味着，无论你做任何操作，只要你提交一个mutation，状态就要立马发生变化，在vue调试工具里面就会有一条状态发生的记录，通过记录就可以清晰的看到状态是怎么变的；还有个好处就是，做一个操作的时候，提交一个mutation，我们可以预测操作完之后的值应该发生什么变化，这样的话，我们的数据就可以预测了。

但是如果把 mutation 写成一个异步的方式，那么点击的时候，状态就不可控了，本来是点一次变化一次，但是点了N多次，不是立马就改变的，所以它记录的还是之前的状态，那么这样的话，我们在调试工具里面做时间旅行的时候就很不方便，状态就会变的很混乱，并且不可预测

但是我们点击的时候有必须要执行一个异步操作，我们应该怎么写呢？ vuex已经想好了，mutation必须同步的改变状态的，如果你有一些异步操作的话，就必须写在 actions 里面。actions 包含了一个异步操作，当异步操作完成了之后，再去提交一个 mutation 去改变状态

actions里面也是一些函数，函数里面要进行异步操作（定时器模拟），异步操作之后就要改变状态（提交一个mutation），怎么提交mutation呢？前面只要拿到commit（）方法就可以了，那么怎么拿到commit方法的呢？在actions里面，如果你触发这个 actions 的话，它会接收到一个参数 context ， 这个参数不是当前实例，它是一个对象，不过在对象里面包含了和实例一样的方法，其中一个方法就是commit（）,commit里面提交一个mutation 为 addIncrement，还有一个参数n为5；那么现在点击按钮的时候需要提交一个异步操作的函数 addAction，而不是之前提交的 mutation

异步操作：

let store = new Vuex.Store({

state: {

count: 100

},

mutations: {

addIncrement(state){

state.count += 1

}

}，

actions: {

addAction (context){

setTimeout(()=>{

//改变状态，提交mutations

context.commit('addIncrement', { n:5 })

},1000)

}

}

})

点击按钮的时候触发一个 action，

this.$store.dispatch('addAction')

dispatch()方法里面写上一个action异步操作函数的名字

6、actions里异步操作函数的参数 context

context是一个对象，不是当前实例，这个对象里包含了当前实例里的方法；context里面都有哪些东西呢？

console.log(context) // commit、dispatch、getters、rootGetters、rootState、state

其中，dispatch 是另一个 action ，当发送一个Ajax，拿到结果之后再去发送第二个Ajax；那么怎么触发第二个ajax呢？有个 dispatch() 方法，里面写上要出发的第二个异步操作函数的名字就行：dispatch('textAction')

actions: {

addAction (context){

console.log(context)

setTimeout(()=>{

//改变状态，提交mutations

context.dispatch('textAction')

},1000)

},

textAction(){

console.log( '我被触发了！' )

}

}

这种方式在第二个异步请求的时候也可以传一个参数，但是在接收这个参数的时候，第一个参数还是一样的，是 context ，第二个才是通过触发 action 接收到的参数 obj ；

actions: {

addAction (context){

setTimeout(()=>{

//改变状态，提交mutations

context.dispatch('textAction',{ test: '测试'})

},1000)

},

textAction( context, obj ){

console.log( obj ) // test: "测试"

}

}

也可以写成对象的形式，type:textAction

actions: {

addAction (context){

setTimeout(()=>{

context.dispatch({

type: 'textAction',

test: '测试'

})

},1000)

},

textAction( context, obj ){

console.log( obj ) // test: "测试"

}

}

7、在一个页面里面渲染多次 num

html：

<input type="button" value="-" />

<span>{{ num }}</span>

<input type="button" value="+" />

<p>{{ num }}</p>

因为两个num应用的是同一个状态（同一个计算属性），当点击的时候，他们两个都会发生改变。现在的需求修改一下：基数为100，当点击的时候，第一个num不断增加，而第二个num增加到120的时候就不再增加了。

注意，这里用的同一个状态，一个往前加，一个不继续往前加；Vuex 给我们提供了一个方式 ：getters对象（和computed性质一样，也是计算属性），对象里面也是定义一些属性，如下的 filterCount ,属性对应的是函数，属性的值是函数执行后的返回值；当函数触发的时候，它会去接收一个 state，通过state可以访问到 count ，如果state.count>=120 ? 120 :state.count,那么第二个num用的就不是state.count了，而是getters里的filterCount；那么怎么获取到filterCount？在组件的计算属性里面单独去定义一个num2，返回值为this.$store.getters.filterCount,渲染在标签里就用num2 ：<p>{{ num2 }}</p>

let store = new Vuex.Store({

state: {

count: 100

},

getters: {

filterCount(state){

return state.count >= 120 ? 120 : state.count;

}

},

mutations: {

addIncrement(state){

state.count += 1

}

}，

actions: {

addAction (context){

setTimeout(()=>{

context.dispatch({

type: 'textAction',

test: '测试'

})

},1000)

},

textAction( context, obj ){

console.log( obj ) // test: "测试"

}

}

})

Increment.vue :

computed: {

num () {

return this.$store.state.count

},

num2 () {

return this.$store.getters.filterCount

}

}

<p>{{ num2 }}</p>

六、Vue的调试工具：chrom插件

七、辅助函数mapState、mapActions、mapGetters、mapMutations

在Vuex里面使用辅助函数，可以极大的简化我们的操作。这几个辅助函数只要掌握一个，其他都是相同的。

1、在哪个组件里使用，就要在哪个组建立把这些方法从vuex里给引入进去：vuex给我们提供对象，我们需要拿到对象的方法：

import { mapState } from 'vuex'

{ mapState } 这里是用的ES6的结构赋值

·此时，我们便可以在这个组件里用这个方法了。mapState 是状态的一个映射，它给我们提供了一个函数；

·上面的列子的 num 我们是用 this.$store.state.count 获取的，现在我们用 mapState 获取；

★用mapState去Vuex中去取值：

a、mapState 调用完之后可以接一个对象，而computed 对应的也是一个对象，所以可以这样写：

computed: mapState({

num: 这里是对应的值

})

这里定义的key值是num，num对应的值的写法有几个：

·第一种：可以写成一个键值函数，它可以接受Vuex里面的状态作为参数，后面需要哪个状态就写哪个状态：

computed: mapState({

num: state => state.count

})

·第二种：num对应的值，可以直接写成state里的key值：

computed: mapState({

num: 'count'

})

·第三种：num对应的值，可以写成一个函数，函数接收state作为参数

computed: mapState({

num: function （state）{

return state.count

}

})

这里面还可以对state状态进一步处理，比如我们再加个100：

computed: mapState({

num: function （state）{

return state.count + 100

}

})

·第四种：在Vuex里面我们定义的是count，如果组件里我们就想用这个名字，而不想另外定义一个num了，我们就不用写的这么复杂了：

computed: mapState({

count: 'count'

})

这个时候我们渲染的就不是num，而是count了：<span>{{ count }}</span>

b、mapState除了可以接收一个对象，还可以接收一个数组,数组里面可以写上state里面的key值；因为key值定义在computed里面，正好和state状态的名字是一样的；需要几个值就取几个值，这样就比之前用 this.$store.key值 要简化很多：

computed: mapState(['count',''])

★如果组件里面需要有一个自定义的计算属性，而我们又用的是mapState定义的计算属性computed，那么我们就不能去自定义需要的属性了，那么我们该如何写呢？

我们不能直接去调用napState，让它返回值给计算属性computed了；我们还是按照以前的写法，计算属性对应的是一个对象，对象里面一部分值要从state里面去取，另一部分是自定义的（abc）；

123不是从Vuex里面去取的，而是从自定义abc的里面去取；count是用 mapState 从Vuex里面去取，因为mapState返回的是一个对象，所以我们可以把返回的对象里的属性扩展到当前的computed里面，这里要用到扩展运算符：... (三个点)。 mapState()函数里面的写法就是上面所说的几种写法。

computed: {

abc(){

return 123

},

...mapState(['count'])

}

2、当我们从getters里面去取值，比如之前的num2，我们也可以使用辅助函数 mapGetters ：

import { mapState,mapGetters } from 'vuex'

使用的时候，我们依然是用在计算属性身上，运用扩展运算符 ；

key值定为num2，num2对应的值是一个字符串(和之前的语法一样，可以写成数组、函数等)，这个字符串是就是我们在Vuex里面定义的getters的名字 filterCount ：

computed: {

abc(){

return 123

},

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

...mapGetters({

num2: 'filterCount'

}),

}

3、异步请求的时候，我们使用dispatch()去调用异步操作函数 addAction ，如果触发的个数少还可以一个个的写，如果触发N多个，那么就很不方便，所以我们可以用 mapActions 去简化操作：

引入辅助函数：import { mapState,mapGetters,mapActions } from 'vuex'

辅助函数使用：

依然用的扩展运算符，调用mapActions，里面传入一个对象，对象里面的key值就是我们调用的点击事件的函数名字：addHanlde ，对应的值是我们需要去触发的Vuex里面actions的一个key名，如 addAction：

methods: {

...mapActions({

addHanlde: 'addAction'

})

}

4、当commit要提交的数据里面除了要提交的执行函数，我们还需要传参的时候，我们用 mapMutations 去操作：

引入辅助函数：import { mapState,mapGetters,mapActions,mapMutations } from 'vuex'

辅助函数使用：

依然用的扩展运算符，调用mapMutations，里面传入一个对象，对象里面的key值就是我们调用的点击事件的函数名字：deHanlde ，对应的值是我们需要去触发的Vuex里面mutations的一个key名，如 deIncrement；

这里需要的传参不能写在methods里面，这样写是错误的：【deHanlde: 'deIncrement(参数)'】，因为这里的值是代表事件的名字，如果在后面加上括号，它会认为是一个完整的名字 deIncrement(参数)，而不是函数的执行。

这里的触发函数 deIncrement 赋值给了deHanlde，所以在我们点击的时候就可以执行 deHanlde()，那么我们就可以对这个函数进行传参了：@click = 'deHanlde({ deNum:5 })'

Vuex里：

mutations: {

deIncrement(state, payload){

state.count -= payload.deNum

}

}

组件里：

methods: {

...mapMutations({

deHanlde: 'deIncrement'

})

}

组件里的html：

<input type="button" value="+" @click="addHanle({ deNum:5 })" />

如果遇到 action 或者 mutation 需要传值的时候，如果用的是map的方式去写，传值的时候要在点击时候的函数里面去传，就可以传到 action 或者 mutation 里面去了

八、案列：仿select（使用Vuex）

1、在Vuex里面定义一个状态title，input组件可以访问到，list组件也可以访问到，点击list的时候，把list的内容拿过来，然后改变一下状态，状态是响应的，它就会重新去渲染input组件

2、异步请求的时候，我们需要一个Ajax请求接口，接口是后端给的地址；我们可以用开源的数据来模拟（比如：easy-mock）获的一个接口：https://easy-mock.com/mock/5a3a14090bd9de68557f35d6/example/listTest；easy-mock给我们提供了服务，直接可以跨域请求；

3、数据有了，我们需要一个发送Ajax请求的模块axios，这模块我们也需要安装一下：

npm run axios

★list.vue:

数据是在list里面去用的，所以我们需要在list里面去获取；

在一个生命周期结束之后去获取：

created(){

//获取数据

this.$store.dispatch('getListAction')

}

这是它的一个生命周期钩子函数，就是一个vue实例被生成后调用这个函数。一个vue实例被生成后还要绑定到某个html元素上，之后还要进行编译，然后再插入到document中。每一个阶段都会有一个钩子函数，方便开发者在不同阶段处理不同逻辑。

一般可以在created函数中调用ajax获取页面初始化所需的数据。

getListAction 是异步操作函数，需要在Vuex里面去写函数内容

★index.js：

import axios from 'axios'

axios模块里面有个方法 get()(这里要看发送的是什么请求，如果是get请求就用get方法，如果是其他的请求就用对应的方法)；然后把接口地址放在get()方法里：axios.get('https://easy-mock.com/mock/5a3a14090bd9de68557f35d6/example/listTest

')；

axios 支持 promised ，所以在请求完，调用get方法之后，我们就可以使用 .then() 方法，里面有个回调函数，如果成功，就可以拿到data；

一般我们还需要监控请求失败的情况，使用catch()方法来捕获到它的错误，catch()方法里面也是一个回调函数，拿到错误信息 error

在actions里面去发送ajax请求；

state:{

listData: []

},

mutations: {

changList(state,list){

state.listData = list

}

},

actions:{

getListAction({commit}){

//发送请求,

axios.get('https://easy-mock.com/mock/5a3a14090bd9de68557f35d6/example/listTest')

.then((data) => {

//console.log(data)

//这里拿到的数据有可能在其他地方使用，所以需要自定义一个状态 listData:[] 来达到公用的效果；当发送ajax请求之后，listData的值就要改成 data.data，要修改状态的值，就要提交一个mutation，所以在mutation里定义一个方法：changList

commit('changList',data.data) //拿到数据后，提交mutation，改变状态

})

.catch((error) => {

console.log(error)

})

}

}

★list.vue:

异步操作函数 getListAction 写好之后，需要在 list 里的 created 去触发（这里就回到了上面list.vue里面所写的代码）

下一步就需要在 list.vue 的计算属性里面拿到数据：

computed:{

data(){

return this.$store.state.listData

}

}

然后渲染在html标签里

八、划分模块module

select 模块：

let selectModule = {

state: {

listData: [],

title: ''

},

mutations: {

listDataChange (state, list) {

state.listData = list

},

titleChange (state, title) {

state.title = title

}

},

actions: {

listAction ({commit}) {

//异步请求

}

}

}

放入容器里面：

let store = new Vuex.Store({

modules: {

selectModule

}

})

组件里获取数据：

this.$store.state.selectModule.listData