day09笔记

课程回顾

- 路由导航守卫
  - 全局导航守卫
    - 前置钩子函数 beforeEach((to,from,next)=>{})
    - 后置钩子函数 afterEach((to,from)=>{})
  - 路由独享守卫

    beforeEnter(to,from,next){

    }
  - 组件内部的守卫
    - 进入组件之前的守卫

    beforeRouteEnter(to,from,next){}
    - 组件内部的更新守卫

    beforeRouteUpdate(to,from,next){}
    - 离开组件的守卫

    beforeRouteLeave(to,from,next){}
- 路由懒加载

```
const 变量 = ()=>import('组件路径')
const 变量 = ()=>Promise.resolve(import('组件路径'))

{
    component:()=>import('组件路径') || 变量
}
```

- 组件通信（组件传值）
  - 父子组件
  - 子父组件通信
  - 非父子组件
- 跨域

```
'/api':{
    target:'//要解决的目标地址',
    changeOrigin:true,//是否跨域
    pathRewrite:{
        '^/api':'//要解决的目标地址'
    }
}
```

问题：

一、export 和 export default 区别

```
ES6重要的导入导出

export 导出方式
一、export 变量 || export 函数
页面中可以有多个export

引入的时候 用import {变量} from '封装好的模块路径'
|| import {变量,变量,变量} from '封装好的模块路径'

二、 默认导出 export default 变量 || export
default 函数 || export default {}
一个模块中只有一个默认导出，你只能导出一次
引入的时候  import 变量 from '模块的路径'
```

二、全局导航守卫怎么去拦截注册？？？

```
不会有这种需求！！！
一、全局导航拦截登录，这种逻辑会用于后台管理项目（内部项目）。根本不会有注册
二、如果是电商的移动端，那么你不会去全局拦截登录，而是全局拦截一下必要的页面，比如购物车。拦截是否去购物车，拦截是否登录
```

---

vuex状态管理！！！

vuex 和路由一样是vue的核心插件.

vuex借鉴的是React中的状态管理思想 （Flux 和 Redux）

vuex的官网：

```
https://vuex.vuejs.org/zh/
```

vuex的概念

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的使用场景

大型的单页应用项目
多个组件共享一个数据的时候

vuex的下载

```
npm install(i) vuex

版本号：  + vuex@3.5.1
```

vuex定义与使用

创建store文件夹=>创建index.js

```
//引入vue核心库
import Vue from 'vue'
//引入vuex的核心库
import Vuex from 'vuex'
//引入封装好的接口文件
import {getBanner} from '../util/axios'
//调用Vuex
Vue.use(Vuex)
//vuex是一个对象，不能直接实例化，要实例化Store
//console.log(new Vuex.Store(),'你是谁')

export default new Vuex.Store({
```

```
    state:{//唯一的数据源
    },
    mutations:{//修改state的唯一方式，且它只能是同步方法
    },
    getters:{//它类似于计算属性computed
        //它一般作为视图和state之间的中间层存在
    },
    actions:{
        //大量的异步操作
    }
})
```

vuex的核心属性

一、state

- 概念

唯一数据源，用来管理数据

- 应用

```
赋值：
state:{
    KEY:'VALUE'
}


取值
this.$store.state.KEY
```

- 辅助性函数

```
//引入
import {mapState} from 'vuex'

//计算属性
computed:{
    ...mapState(['state值'])
    或者
    ...mapState({
        key:state=>state.name
    })
}
```

二、mutations

- 概念

修改state的唯一方式，它只操作同步方法，它被commit触发

- 应用

```
一、未传送参数状态
组件触发：
方法名(){
    this.$store.commit('type名称')
}

vuex应用
mutations:{
    type名称(state){
        //修改state即可
    }
```

```
}
==========================================
=
二、传送参数状态
组件触发:
方法名(){
    this.$store.commit('type名称',参数)
}

vuex应用
mutations:{
    type名称(state,payload){
        //修改state即可
        //payload 参数名
    }
}
```

- 辅助性函数

```
//视图
//CHANGE_NAME 必须和mutations里面定义的一致
<button @click="CHANGE_NAME">点我修改</button>
//引入
import {mapMutations} from 'vuex'

//应用
    methods:{
        ...mapMutations(['CHANGE_NAME']),
    }
```

三、getters

- 概念

它类似于vue中computed计算属性，它可以缓存，它作为视图层和状态层中的中间层存在,让state渲染更加的方便简单

- 应用

```
取值:
{{$store.getters.属性名称}}

vuex的应用
getters:{
    属性名称(state){
        return state.属性名
    }
}
```

- 辅助性函数

```
//引入
import {mapGetters} from 'vuex'

//计算属性
computed:{
    ...mapGetters(['getter值'])
    或者
    ...mapGetters({
        key:gettter=>gettter.name
    })
}
```

四、actions

- 概念
  - action 它类似mutation，但是要想修改state属性还是要去操作mutation
  - 它是异步操作，当然也可以操作同步
- 应用

```
一、未传送参数状态
组件触发：
方法名(){
    this.$store.dispatch('type名称')
}

vuex应用
actions:{
    type名称(context){
        //context 上下文
        context.commit('mutation方法')
    }
    或者代码优化
    type名称({commit}){
       commit('mutation方法')
    }
}
==========================================
=
二、传送参数状态
组件触发：
方法名(){
    this.$store.dispatch('type名称',参数)
}

vuex应用
```

```
actions:{
    type名称(context,参数){
        //context 上下文
        context.commit('mutation方法',参数)
    }
    或者代码优化
    type名称({commit},参数){
       commit('mutation方法',参数)
    }
}
```

- 辅助性函数

```
//视图
//changeNum 必须和actions里面定义的一致
未传参
<button  @click='changeNum'>我是action修改
</button>
传参
<button  @click='changeNum(6)'>我是action修改
</button>
//引入
import {mapActions} from 'vuex'

//应用
    methods:{
        ...mapActions(['changeNum']),
    }
```

五、modules

- 概念

防止state过于复杂冗余，我们可以把状态层切割成不同模块

- 应用

```
//定义一个模块
const moduleA = {
    state:{
        name:'我是模块A'
    },
    getters:{},
    mutations:{},
    actions:{}
}

const moduleB = {
    state:{
        name:'我是模块B'
    },
    getters:{},
    mutations:{},
    actions:{}
}


    modules:{
        moduleA,
        moduleB
    }
```

拆分状态层

一、拆分state文件

```
export default {
    //定义N条数据
    name:'stateName'
}
```

二、拆分getters

```
//导出getter
export default {

}
```

三、拆分mutations

```
//导出mutation
export default {

}
```

四、拆分actions

```
//导出mutation
export default {

}
```

五、拆分模块

创建一个modules文件夹，那你根据你的产品需求再创建不同的文件和文件夹

modules=>home=>home.js

```
//定义state
const state = {
    name:'home的名字'
}

//定义一个 getters
const  getters = {
    getStateName(state){
        return state.name
    }
}

//定义mutations
const mutations = {

}

//定义一个actions
const actions={

}

//导出一个默认的对象
export default {
    state,
    getters,
    mutations,
    actions,
    //命名空间
    namespaced:true
}
```

modules=>rank=>rank.js

```
//定义state
const state = {
    name:'rank的名字'
}

//定义一个 getters
const  getters = {
    getStateName(state){
        return state.name
    }
}

//定义mutations
const mutations = {

}

//定义一个actions
const actions={

}

//导出一个默认的对象
export default {
    state,
    getters,
    mutations,
    actions,
    //命名空间
    namespaced:true
}
```

六、全部引入到store=>index.js

```
//引入核心库
import Vue from 'vue'

//引入vuex
import Vuex from 'vuex'

//调用当前的插件
Vue.use(Vuex)


//引入封装好的模块
import home from './modules/home/home'
import rank from './modules/rank/rank'

//引入state
import state from './state'
import getters from './getters'
import actions from './actions'
import mutations from './mutations'

//导出
export default new Vuex.Store({
    state,
    getters,
    actions,
    mutations,
    modules:{
        home,
        rank
    }
})
```

七、视图组件应用

```
<template>
    <div>
        <h1>我是home</h1>
        <!-- <h2>接收数据---{{getStateName}}
</h2> -->
        <h2>接收数据---{{homeName}}</h2>
    </div>
</template>

<script>
import {mapGetters} from 'vuex'
export default {
    data() {
        return {

        };
    },
    mounted(){
        //页面一加载，打印仓库
        console.log(this.$store,'仓库')
    },
    computed:{
       // ...mapGetters(['getStateName'])
        ...mapGetters({
            homeName:'home/getStateName'
        })
    }
};
</script>

<style  lang="" scoped>

</style>
```

vuex和localStorage的区别

相同点：
都有可以管理数据，都可以进行组件之间的传值
不同点：
localStorage，是永久性存储，页面一刷新，它的值依然存在，除非手动删除。localStorage的值不刷新不会更新。
vuex一刷新，恢复到初始值状态，如果一定要让数据存储，你可以结合localStorage去存储，还有可以调取接口获取数据。
vuex的值一旦改变，相关依赖组件也会跟着变化

UI框架

一、pc端的UI框架之ElementUI

官网地址：

```
https://element.eleme.cn/#/zh-CN
```

下载命令

```
npm install(i) element-ui
版本号：+ element-ui@2.14.0
```

在脚手架中调用element-ui

main.js

```
//引入element-ui
import elementUI from 'element-ui'
//引入element-ui的css文件
import 'element-ui/lib/theme-chalk/index.css'
//调用element
Vue.use(elementUI)
```

二、移动端的框架

①mintUI(公司用的很少)

②vantUI(主流的移动端框架)

作业：

一、整理今天笔记

二、今天的案例敲3遍

三、添写每日监测

四、看UI框架

错误集锦

一、unknown mutation type: aa

意思：①你就没有定义aa的mutation ②你的aa可能写错了

二、 unknown action type: aa

意思：①你就没有定义aa的actions ②你的aa可能写错了

三、mapGetters: mapper parameter must be either an Array or an Object

要求我们在调取辅助性函数的时候，必须以数组或者对象的形式