路由配置：

1、安装vue-router npm install vue-router --save （对于创建项目时没有选择路由的项目需要手动安装路由）

2、创建多个页面（组件）

3、新建 routes.js路由配置文件

```
import VueRouter from 'vue-router'
import Page1 from './components/page1'
import Page2 from './components/page2'
import Page3 from './components/page3'
import Dashboard from './components/Dashboard'
```

引入组件

```
let routes = new VueRouter({
    mode:"history",
    routes:[
        //  路由定义
        {path:'/page1',component:Page1},
        {path:'/page2',component:Page2},
        //  路由嵌套
        {
            path:'/dashboard',
            component:Dashboard,
            children:[
                {path:'page1',component:Page1},
                {path:'page2',component:Page2},
                {path: 'page3/:id', props: true, component: Page3 },
            ]
        },
    ]
})
export default routes
```

path是路由地址，component是组件名

4、在main.js引入路由配置文件

```
import Vue from 'vue'
import App from './App.vue'
//  引入路由
import VueRouter from 'vue-router'
//  引入路由配置
import router from './router'

Vue.config.productionTip = false
// 开启路由插件
Vue.use(VueRouter)

new Vue({
  //注册路由
  router,
  render: h => h(App)
}).$mount('#app')
```

5、使用父组（APP.vue）件中使用<router-view></router-view>占位符，然后再引用路由导航<router-link to="/page3/aaa">login</router-link>

```
<template>
  <div id="app">
    <div>
      <router-link to="/page1">Page1</router-link>
      |
      <router-link to="/page2">Page2</router-link>
      |
      <router-link to="/page3/aaa">login</router-link>
    </div>
    <div>
      <p>在首页中</p>
    </div>
    <router-view></router-view>
  </div>
</template>
```

路由导航

占位符

history模式

默认是hash模式，url使□#后□定位路由，对seo不利，设置history，就可以使□普通的url模式

```
export default new VueRouter({
    mode:"history",
    routes:[
        {path:'/page1',component:Page1},
        {path:'/page2',component:Page2},
    ]
})
```

动态路由、参数、占位符

```
// 动态路由
{path:'/page3/:id',component:Page3},
//  在组件中获取参数
// return this.$route.params.id

// 设置props属性，获取路由的变量 就和普通的属性传递没什么区别
  { path: '/page3/:id', props: true, component: Page3 },
//  在组件中获取参数
// props:['id']
```

id就是地址占位符，也是参数可以在组件中获取

路由嵌套

```
//  路由嵌套
{
    path:'/dashboard',
    component:Dashboard,
    children:[
        {path:'page1',component:Page1},
        {path:'page2',component:Page2},
        {path: 'page3/:id', props: true, component: Page3 },
    ]
},
```

父组件地址

关键字children,父组件下的子路由

路由重定向、路由的异步组件

```
routes:[
    //访问根目录  进入页面1  重定向跳转
    { path: '/', redirect: '/dashboard/page1' },
```

关键字 redirect

```
{
    path:'/login',          访问时加载组件        异步组件
    component:()=>import('./components/login')    //懒加载
},
```

路由守卫及组件内部路由事件

定义在路由配置文件中的全局守卫

beforeEach 路由跳转进入页面前执行

afterEach 路由跳转进入页面结束后执行

```
//路由守卫
//路由跳转前执行
routes.beforeEach((to,from,next)=>{
    console.log("路由跳转前执行:",to)
    //做权限判断  如果不是login页面就可以访问
    if (to.path!='/login'){
        next()
    } else{
        //做权限判断  如果是login页面就弹出提示
       alert("你不能进登录页面哦")
    }
})
//路由跳转结束后执行
routes.afterEach((to,from)=>{
    console.log("路由跳转结束后执行:",to)
})
export default routes
```

```
// 组件内部的路由

beforeRouteEnter    // 进入组件后执行的路由

beforeRouteLeave    // 离开该组件之前执行的路由
```

```
export default {
    name: "page3",
    //组件内部的路由
    //进入组件后执行的路由
    beforeRouteEnter(to,from,next){
        console.log("进入page3了")
        next()
    },
    //离开该组件之前执行的路由
    beforeRouteLeave(to,from,next){
        console.log("要离开了")
        next()
    },
```

路由的执行顺序

1. 导航被触发。
2. 调用全局的 beforeEach 守卫。
3. 在重用的组件里调用 beforeRouteUpdate 守卫。
4. 在路由配置里调用 beforeEnter。
5. 在被激活的组件里调用 beforeRouteEnter。
6. 调用全局的 beforeResolve 守卫 (2.5+)。
7. 导航被确认。
8. 调用全局的 afterEach 钩子。
9. 触发 DOM 更新。

```
beforeEach  全局的
beforeRouteEnter   组件的
beforeRouteUpdate  组件的
beforeEnter 全局的
beforeRouteEnter   组件的
beforeResolve 全局的
beforeRouteLeave   组件的
afterEach  全局的
```

Vuex数据管理

1、npm install vuex --save 安装

2、核心概念

store　对所有数据的存取改的管理中心

state　用于保存所有组件的公共数据.

mutations　对数据进行修改

3、步骤：

3.1、新建store.js

3.2、引入vuex　　import Vuex from 'vuex'

```
import Vuex from 'vuex'
//1、new Vuex就是新建一个存储实例，来管理数据
export default new Vuex.Store({
    strict:true,     // 禁用this.$store.state.count++  这种直接修改数据的方式
    //全局共享
     state:{
         counts:664
     },
})
```

3.3、new Vuex就是新建一个存储实例来管理，并创建全局共享的数据

```
//全局共享
 state:{
     counts:664
 },
```

```
export default new Vuex.Store({
    strict:true,     // 禁用this.$store.state.count++  这种直接修改数据的方式
    //全局共享
     state:{
         counts:664
     },
```

3.4、引入vue,并注册vuex

```
import Vue from 'vue'

Vue.use(Vuex)
```

```
import Vue from 'vue'
import Vuex from 'vuex'
//  2
Vue.use(Vuex)
```

3.5、在组件中基本使用展示数据

```
count(){
    //  3 展示获取数据
    return this.$store.state.counts
},
```

mutations专用于修改数据

```
mutations: {
    inc(state) {
        // state.counts   获取到全局数据的值     修改后再赋值给数据对象本身
        state.counts = (state.counts) + s
    }
},
```

在组件中

```
//  4 调用修改数据的方法,提交修改的请求方法
this.$store.commit('inc')

count(){
    // 展示获取数据   此时获取展示的数据就是修改后的数据
    return this.$store.state.counts
},
```

getters 对修改后的数据做处理

```
//  对修改后的数据做处理
getters:{
    //两个方法相同，只是写法不用
    m: state => `¥${state.counts * 10}`,
    ms(state){
        `¥${state.counts * 10}`
    }
},


money(){
    // 使用修改后处理过的数据
    return this.$store.getters.m
}
count(){
    return this.$store.state.counts
},
```

actions 异步的修改数据

```
//异步的修改数据
actions:{
    // 异步的参数是必须的只能是 store 或者 context
    itsanyss(store){
        // 调用 mutation
        setTimeout(()=>{
            store.commit('inc')   //调用执行修改数据的方法
        },3000)
    },
    itsany: context => {
        // 调用 mutation
        setTimeout(()=>{
            context.commit('inc')    //调用执行修改数据的方法
        },5000)
    },
```

```
    insany({commit}){
        setTimeout(()=>{
            commit('inc')    //调用执行修改数据的方法
        },10000)
    }
}


//使用actions的修改数据的方法
this.$store.dispatch('insany')

this.$store.dispatch('itsanyss')
展示异步修改数据的方法
count(){
    return this.$store.state.counts
},
```

在组件中map映射获取多个数据的方式 ...mapState

```
import {mapState} from 'vuex'
computed:{
    // map映射获取数据的方式
    ...mapState({
        count:state=>state.counts,

        county(state){
            state.counts
        }
    }),
```

执行顺序

1、state:{counts:664}创建数据 ---> 直接获取 state.数据名

2、mutations的修改数据 ---> this.$store.commit('inc') ---> this.$store.state.数据名

3、getters 处理修改后的数据 ---> this.$store.getters.m ---> this.$store.state.数据名

4、actions 异步的修改数据----> mutations---> this.$store.dispatch('itsanyss') ---> this.$store.state.数据名

mutations 对数据进行修改 store中的methods,

mutations对象中保存着更改数据的回调函数,

该函数名官方规定叫type, 第一个参数是state,

第二参数是payload, 也就是自定义的参数.

getters属性 理解为所有组件的computed属性, 也就是计算属性.

vuex的官方文档也是说到可以将getter理解为store的计算属性,

getters的返回值会根据它的依赖被缓存起来,

且只有当它的依赖值发生了改变才会被重新计算。

Actions actions 类似于 mutations，不同在于：

actions提交的是mutations而不是直接变更状态

actions中可以包含异步操作，mutations中绝对不允许出现异步

actions中的回调函数的第一个参数是context，是一个与store实例具有相同属性和方法的对象