Vue-router && Vuex实战

课堂目标

1. vue-router基础配置
2. 路由传参
3. 子路由
4. 路由重定向
5. 路由守卫
6. vuex数据流
7. Store
8. state
9. mutation
10. action

知识要点

1. vue-router多页面
2. vuex管理数据

资源

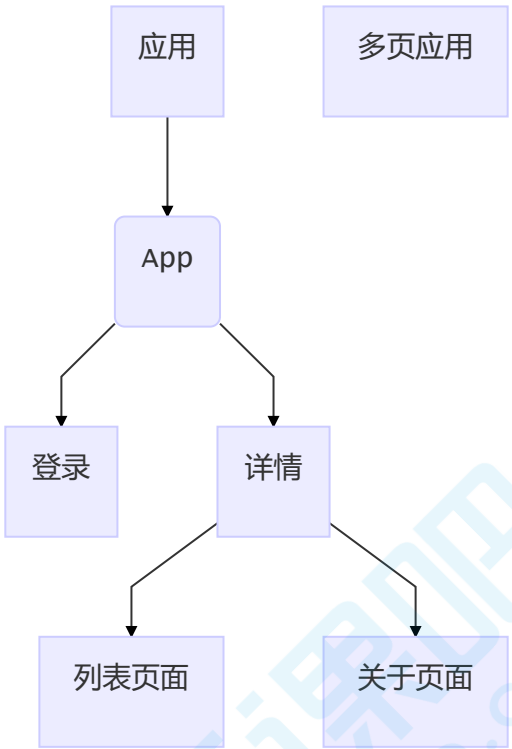
1. vue-router
2. vuex

vuex模块切分 modules 分模块(后续) getter可以理解为vuex里面的computed action 异步mutions

起步

Vue Router 是 Vue.js 官方的路由管理器。它和 Vue.js 的核心深度集成，让构建单页面应用变得易如反掌。包含的功能有：

1. 嵌套的路由/视图表
2. 模块化的、基于组件的路由配置
3. 路由参数
4. 基于 Vue.js 过渡系统的视图过渡效果
5. 细粒度的导航控制
6. 带有自动激活的 CSS class 的链接
7. HTML5 历史模式或 hash 模式



1. 新建项目 `vue create vue-router-vuex`
2. 安装vue-router `npm install vue-router -S or vue add router`
3. `npm run serve`

多页面体验

- 新建`routes.js`

新建两个测试组件

```
<template>
    <div>页面1</div>
</template>

<script>
export default {


}
</script>
```

```
<template>
    <div>页面2</div>
</template>

<script>
export default {


}
</script>
```

```
import Vue from 'vue';
import VueRouter from 'vue-router'
import Page1 from './components/Page1'
import Page2 from './components/Page2'

Vue.use(VueRouter)

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

- main.js

```
import Vue from 'vue'
import App from './App.vue'
import VueRouter from 'vue-router'
import router from './routes'
Vue.config.productionTip = false

Vue.use(VueRouter)
new Vue({
  router, //名字必须是router
  render: h => h(App)
}).$mount('#app')
```

- app.vue 使用router-view占位符

```
<template>
  <div id="app">
    <router-view></router-view>
  </div>
</template>

<script>
// import HelloWorld from './components/HelloWorld.vue'
```

```
export default {
  name: 'app',
}
</script>

<style>
</style>
```

访问 `http://localhost:8080/#/page1` 和 `http://localhost:8080/#/page2` 即可看到不同的组件内容

导航

使用App.vue中，使用router-link作为导航按钮

```
<div>
    //to 必须得有
    <router-link to="/page1">Page1</router-link>
    <router-link to="/page2">Page2</router-link>
</div>
```

history模式

默认是hash模式，url使用#后面定位路由，对seo不利，设置history，就可以使用普通的url模式

```
// routes.js

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

url就变成了 `http://localhost:8080/page1`

重定向

```
{
  path: '/',
  redirect: '/page1'
},
```

路由命名

可以给路由设置name，方便router-link 跳转

```
<router-link :to="{name:'home',params:{userId:123}}"></router-link>
```

首页 —》搜索 列表页面 —> 详情页面

/index

/list?serchval=xxx

Shuma/detail/123 1

detail?id=123 2

shipin/detail/123

动态路由

路由可以携带一些参数,使用this.$router获取

```
{path:'/page3/:id',component:Page3}
```

```
<template>
    <div>详情页面</div>
</template>

<script>
export default {
    created(){
        console.log(this.$route)
    }
}
</script>
```

```
[HMR] Waiting for update signal from WDS...                    log.js?1afd:
                                                               Page3.vue?43b2
▼{name: undefined, meta: {…}, path: "/page3/react", hash: "", query: {…}, …}
    fullPath: "/page3/react"
    hash: ""
  ▶ matched: [{…}]
  ▶ meta: {}
    name: undefined
  ▶ params: {id: "react"}
    path: "/page3/react"
  ▶ query: {}
  ▶ __proto__: Object
Download the Vue Devtools extension for a better development experience:
https://github.com/vuejs/vue-devtools                          vue.runtime.esm.js?2b0e:80
```

参数属性传递

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 }
```

```
<template>
    <div>
        <p>详情页面</p>
        <div>
            哈喽啊{{id}}
        </div>
    </div>
</template>

<script>
export default {
    created(){
        console.log(this.$route)
    },
    props:['id']
}
</script>
```

嵌套路由

子路由的概念,比如页面内部的导航复用

```
export default new VueRouter({
    mode: "history",
```

```
    routes: [
        {path:'/login',component:Login},
        {
            path:'/dashboard',
            component:Dashboard,
            children:[
                { path: 'page1', component: Page1 },
                { path: 'page2', component: Page2 },
                { path: 'page3/:id', props: true, component: Page3 }
            ]
        },
    ]
})
```

app.vue

```
<template>
  <div id="app">
    <router-view></router-view>
    <hr>
    <div>开课吧还不错</div>
  </div>
</template>

<script>
// import HelloWorld from './components/HelloWorld.vue'

export default {
  name: 'app',
}
</script>

<style>
</style>
```

dashboard.vue

```
<template>
    <div>
        <div>
            <router-link to="/dashboard/page1">Page1</router-link>
            |
            <router-link to="/dashboard/page2">Page2</router-link>
            |
            <router-link to="/login">login</router-link>

        </div>
        <hr>
        <router-view></router-view>
    </div>
</template>
```

```
<script>
export default {

}
</script>
```

page1.vue

```
<template>
    <div>
        <p>页面1</p>
        <div>
            <router-link to='page3/react'>react</router-link>
            <br>
            <router-link to='page3/vue'>vue</router-link>
        </div>
    </div>
</template>

<script>
export default {

}
</script>
```

命名视图

一个组件内部有多个router-view 怎么来分配组件呢？ 比如三栏布局，顶部栏点击按钮，左侧栏的菜单变化

```
home.vue
<template>
    <div id="app">
        <router-view></router-view>

        <router-view name="a"></router-view>
    </div>

</template>



routes:[
  {
    path:'/home',
    compontents:{
      defalut: Home,
      a: List
    }
```

```
  }
]
```

导航守卫

- **全局守卫**
  - 每次路由跳转 都会被触发

```
router.beforeEach((to, from, next) => {
  //全局前置守卫,当一个导航触发时，全局前置守卫按照创建顺序调用
  // 数据校验时，非常有用 if(to.fullPath === '/home'){next('/login')}
  console.log('before Each')
  next();
})
router.beforeResolve((to, from, next) => {
  //全局解析守卫,2.5.0 新增,这和 router.beforeEach 类似，区别是在导航被确认之前，同时在所有组件内守卫和异步路由组件被解析之后，解析守卫就被调用
  console.log('before Resolve')
  next();
})
router.afterEach((to, from) => {
  //全局后置钩子
  console.log('after each')
})
```

beforeEach 所有路由跳转前执行，next同意跳转 比如login执行2秒后跳转

```
routers.beforeEach((to, from, next) => {
    console.log('beforeEach')
    console.log(to)
    if(to.path!='/login'){
        next()
    }else{
        setTimeout(()=>{
            next()
        },2000)
    }
})

routers.afterEach((to, from) => {
    console.log('afterEach')
})
```

- **路由独享的守卫**
  - 写在配置里

  你可以在路由配置上直接定义 `beforeEnter` 守卫

```
const router = new VueRouter({
  routes: [
    {
      path: '/foo',
      component: Foo,
      //在进入这个路由之前调用
      beforeEnter: (to, from, next) => {
        // ...
      }
    }
  ]
})
```

- ###组件内的守卫

  最后，你可以在路由组件内直接定义以下路由导航守卫

```
const Foo = {
  template: `...`,
  beforeRouteEnter (to, from, next) {
    // 在渲染该组件的对应路由被 confirm 前调用
    // 不！能！获取组件实例 `this`
    // 因为当守卫执行前，组件实例还没被创建
  },
  beforeRouteUpdate (to, from, next) {
    // 在当前路由改变，但是该组件被复用时调用
    // 举例来说，对于一个带有动态参数的路径 /foo/:id，在 /foo/1 和 /foo/2 之间跳转的时候，
    // 由于会渲染同样的 Foo 组件，因此组件实例会被复用。而这个钩子就会在这个情况下被调用。
    // 可以访问组件实例 `this`
  },
  beforeRouteLeave (to, from, next) {
    // 导航离开该组件的对应路由时调用
    // 可以访问组件实例 `this`
    // 通常用来禁止用户在还未保存修改前突然离开。该导航可以通过 next(false) 来取消
  }
}
```

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。

异步组件

路由懒加载 vue中配合webpack 非常简单

```
{
    path:'/login',
    component:()=>import('./components/Login')
}
```

Vuex数据管理

有几个不相关的组件，想共享一些数据怎么办呢?

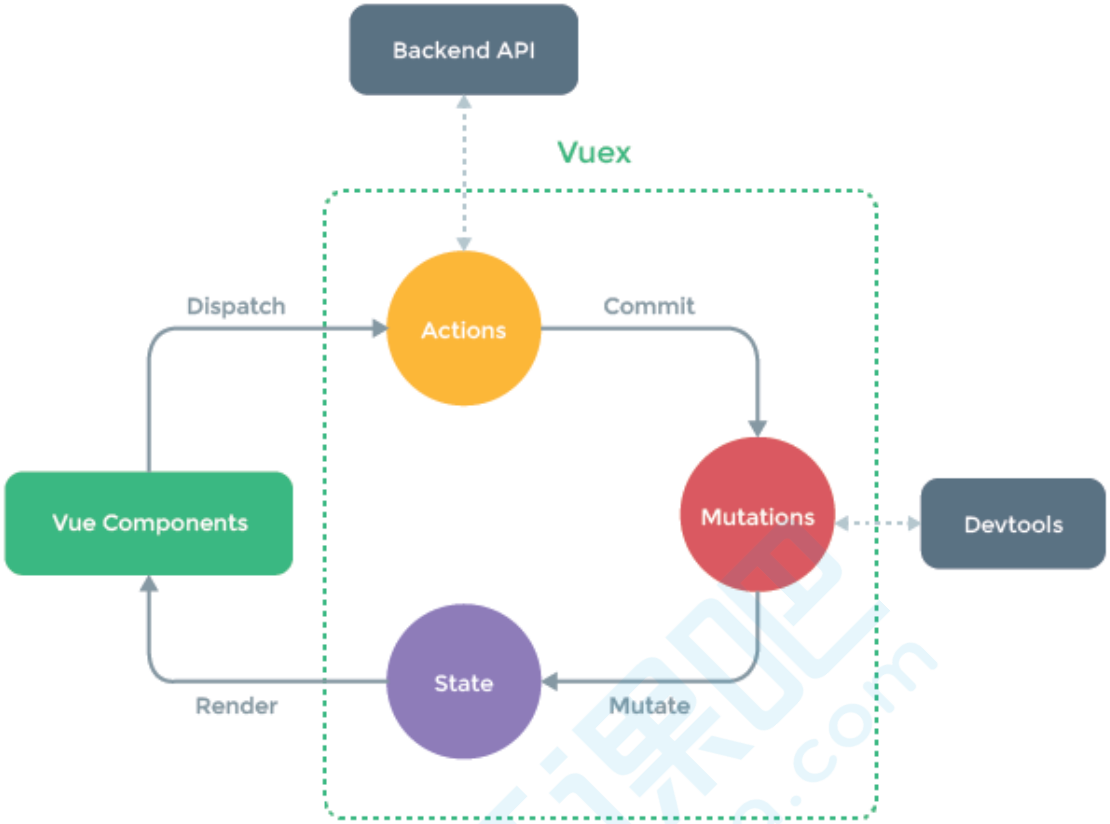
`npm install vuex --save` 安装

Vue add vuex

核心概念

- store
- state //数据中心
- mutations //操作数据
- Actions //什么时候触发操作，执行动作，改变数据

状态管理模式



store

新建store.js

```
import Vuex from 'vuex'

export default new Vuex.Store({
    state:{
        count:0
    }
})
```

入口

```
import Vue from 'vue'
import App from './App.vue'
import VueRouter from 'vue-router'
import Vuex from 'vuex'
import router from './routes'
import store from './store'
Vue.config.productionTip = false
```

```
Vue.use(VueRouter)
Vue.use(Vuex)

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

Page1.vue

```
export default {
  computed: {
    count() {
      return this.$store.state.count;
    }
  }
};
```

Mutation

直接修改state的数据也可以，但是建议使用单向数据流的模式，使用Mutation来修改数据 直接改 VS 数据中心

```
created(){
    setInterval(()=>{
        this.$store.state.count++
    },500)
},
```

使用strict配置，可以禁用这种模式

```
(found in <Root>)
▶ Error: [vuex] Do not mutate vuex store state outside mutation handlers.    vue.runtime.esm.js?2b0e:1819
    at assert (vuex.esm.js?2f62:97)
    at Vue.store._vm.$watch.deep (vuex.esm.js?2f62:746)
    at Watcher.run (vue.runtime.esm.js?2b0e:3345)
    at Watcher.update (vue.runtime.esm.js?2b0e:3319)
    at Dep.notify (vue.runtime.esm.js?2b0e:712)
    at Object.reactiveSetter [as count] (vue.runtime.esm.js?2b0e:1037)
    at eval (Page1.vue?a479:19)
▶ [Vue warn]: Error in callback for watcher "function () {
return this._data.$$state }": "Error: [vuex] Do not mutate vuex store state outside mutation handlers."    vue.runtime.esm.js?2b0e:601

(found in <Root>)
▶ Error: [vuex] Do not mutate vuex store state outside mutation handlers.    vue.runtime.esm.js?2b0e:1819
    at assert (vuex.esm.js?2f62:97)
```

使用commit调用Mutation来修改数据

```
import Vue from 'vue'
import Vuex from 'vuex'

Vue.use(Vuex)

export default new Vuex.Store({
    state: {
        count: 0
    },
    mutations: {
        increment(state) {
            state.count++
        }
    },
    strict: true
})
```

```
export default {
    created(){
        setInterval(()=>{
            this.$store.commit('increment')
        },500)
    },
  computed: {
    count() {
      return this.$store.state.count;
    }
  }
};
```

getters

有时候我们需要从 store 中的 state 中派生出一些状态，我们可以理解为vuex中数据的computed功能

store.js

```
getters:{
    money: state => `￥${state.count*1000}`
},
```

page1.vue

```
computed: {
    money() {
        return this.$store.getters.money;
    }
}
```

Action

Mutation必须是同步的，Action是异步的Mutation

store.js

```
actions: {
    incrementAsync({ commit }) {
        setTimeout(() => {
            commit('increment')
        }, 1000)
    }
},

this.$store.dispatch('incrementAsync')
```

传递参数

```
this.$store.dispatch('incrementAsync', {
    amount: 10
})

actions: {
    incrementAsync(store,args) {
        setTimeout(() => {
            store.commit('incrementNum',args)
        }, 1000)
    }
},

incrementNum(state,args) {
    state.count += args.amount
}
```

mapState

更方便的使用api，当一个组件需要获取多个状态时候，将这些状态都声明为计算属性会有些重复和冗余。为了解决这个问题，我们可以使用 mapState 辅助函数帮助我们生成计算属性，让你少按几次键

```
...mapState({
    count:state=>state.count
}),
```

mapActions

方便快捷的使用action

```
methods:{
    ...mapActions(['incrementAsync']),
    ...mapMutations(['increment']),
},
```

this.$store.dispatch可以变为

```
this.incrementAsync({
    amount: 10
})
```

mapMutions

同理可得

```
...mapMutations(['increment'])
this.increment()
```

回顾