

Vue-router & Vuex实战

Vue-router & Vuex实战

课堂目标

知识要点

资源

起步

多页面体验

导航

history模式

动态路由

单文件组件

参数属性传递

路由嵌套

重定向

路由守卫

组件内部生命周期

异步组件

Vuex数据管理

store

Mutation

getters

Action

mapState

mapActions

mapMutations

回顾

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

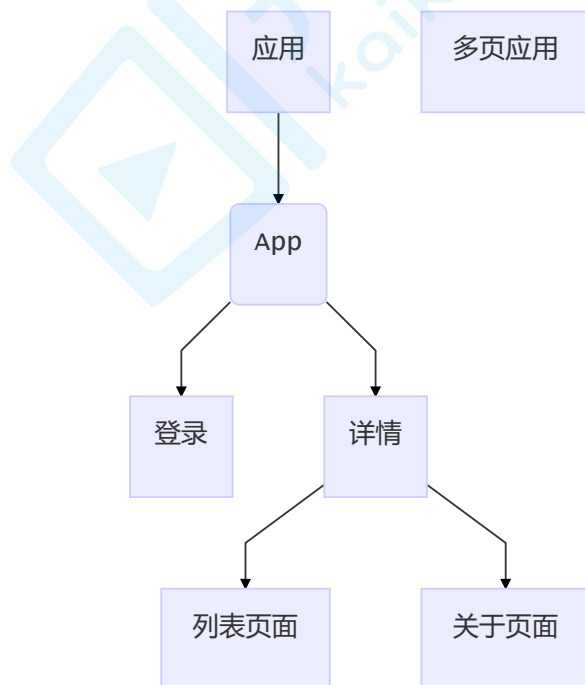
1. [vue-router](#)
2. [vuex](#)

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mutations

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

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,
  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>
  <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`

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>
```

 image-20181228184008105

单文件组件

.vue 是vue单文件组件，一个文件就是一个组件，由template，script和style三个标签构成，分别是html，js和css的内容部分,修改App.vue文件内容

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

<script>
export default {
  created(){
    console.log(this.$route)
  },
  computed:{
    name(){
      return this.$route.params.id
    }
  }
}
</script>
```

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>
```

```

    </div>
  </template>

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

```

路由嵌套

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>
```

重定向

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

路由守卫

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

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

组件内部生命周期

Page3.vue

```
export default {  
  props: ["id"],  
  beforeRouteEnter(to, from, next) {  
    console.log("page3路由进入前");  
    next();  
  },  
  beforeRouteUpdate(to, from, next) {  
    console.log("page3路由, 但是参数变了");  
  }  
}
```

```
    next();
  },
  beforeRouteLeave(to, from, next) {
    console.log("page3路由离开前");
    next();
  }
};
```

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

`npm install vuex --save` 安装

核心概念

- store
- state
- mutations

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

 image-20181228191938737

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  
})
```

mapMutations

同理可得

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

```
this.increment()
```

Vue-router && Vuex实战

课堂目标

知识要点

资源

起步

多页面体验

导航

history模式

动态路由

单文件组件

参数属性传递

路由嵌套

重定向

路由守卫

组件内部生命周期

异步组件

Vuex数据管理

store

Mutation

getters

Action

mapState

mapActions

mapMutations

回顾

回顾