

第4章 路由与状态管理

学习目标：

- 理解路由在单页面工程中的作用
- 掌握可搜索下拉框、复合型输入框等ElementUI的使用，完成招聘管理功能
- 完成文章管理功能
- 理解Vuex状态管理在工程中的作用

1 路由vue-router

1.1 什么是vue-router

vue-router就是vue官方提供的一个路由框架。使用 Vue.js ，我们已经可以通过组合组件来组成应用程序，当你要把 vue-router 添加进来，我们需要做的是，将组件(components)映射到路由(routes)，然后告诉 vue-router 在哪里渲染它们。

1.2 快速入门

1.2.1 初始化工程

```
# 全局安装 vue-cli
npm install -g vue-cli
# 创建一个基于 webpack 模板的新项目
vue init webpack vue-router-demo
# 安装依赖，走你
cd vue-router-demo
npm run dev
```

1.2.2 路由定义

src/App.vue是我们的主界面，其中的 `<router-view/>` 标签用于显示各组件视图内容

src/router/index.js是定义路由的脚本 path是路径， name是名称， component是跳转的组件。

（1）我们现在定义两个页面组件，存放在src/components下

list.vue

```
<template>
  <div>
    这是一个列表
  </div>
</template>
```

about.vue

```
<template>
  <div>
    关于我们
  </div>
</template>
```

（2）定义路由

修改src/router/index.js

```
import Vue from 'vue'
import Router from 'vue-router'
import HelloWorld from '@/components/HelloWorld'
import list from '@/components/list'
import about from '@/components/about'

Vue.use(Router)

export default new Router({
  routes: [
    {
      path: '/',
      name: 'HelloWorld',
      component: HelloWorld
    },
    {
      path: '/list',
      name: 'List',
      component: list
    },
    {
      path: '/about',
      name: 'About',
      component: about
    }
  ]
})
```

(3) 放置跳转链接

修改src/app.vue ,添加链接

```
<router-link to="/" >首页</router-link>
<router-link to="/list">列表</router-link>
<router-link to="/about">关于</router-link>
```

通过router-link标签实现路由的跳转

router-link标签属性如下:

属性	类型	含义
to	string Location	表示目标路由的链接。当被点击后，内部会立刻把 <code>to</code> 的值传到 <code>router.push()</code> ，所以这个值可以是一个字符串或者是描述目标位置的对象。
replace	boolean	设置 <code>replace</code> 属性的话，当点击时，会调用 <code>router.replace()</code> 而不是 <code>router.push()</code> ，于是导航后不会留下 history 记录。
append	boolean	设置 <code>append</code> 属性后，则在当前（相对）路径前添加基路径。例如，我们从 <code>/a</code> 导航到一个相对路径 <code>b</code> ，如果没有配置 <code>append</code> ，则路径为 <code>/b</code> ，如果配了，则为 <code>/a/b</code>

测试运行看是否可以跳转页面

1.3 深入了解

1.3.1 动态路由

我们经常会遇到这样的需求，有一个新闻列表，点击某一条进入新闻详细页，我们通常是传递新闻的ID给详细页，详细页根据ID进行处理。这时我们就会用到动态路由

一个『路径参数』使用冒号 `:` 标记。当匹配到一个路由时，参数值会被设置到 `this.$route.params`

看代码实现：

在src/components下创建item.vue

```
<template>
  <div>
    详细页 {{ $route.params.id }}
  </div>
</template>
```

修改src/router/index.js，引入item组件

```
import item from '@components/item'
```

添加路由设置

```
{
  path: '/item/:id',
  name: 'Item',
  component: item
}
```

修改src/components/list.vue, 增加链接

```
<template>
  <div>
    这是一个列表
    <router-link to="/item/1">新闻1</router-link>
    <router-link to="/item/2">新闻2</router-link>
    <router-link to="/item/3">新闻3</router-link>
  </div>
</template>
```

1.3.2 嵌套路由

实际生活中的应用界面，通常由多层嵌套的组件组合而成。同样地，URL 中各段动态路径也按某种结构对应嵌套的各层组件，例如：

/about/address		/about/linkman
+-----+		+-----+
About		About
+-----+		+-----+
address	+----->	linkman
+-----+		+-----+
+-----+		+-----+

我们来看代码的实现

(1) 在src/components下创建address.vue



```
<template>
  <div>
    地址：金燕龙
  </div>
</template>
```

创建linkman.vue

```
<template>
  <div>
    联系人：小二黑
  </div>
</template>
```

(2) 修改src/router/index.js

引入linkman和address

```
import linkman from '@/components/linkman'
import address from '@/components/address'
```

配置嵌套路由：

```
{
  path: '/about',
  name: 'About',
  component: about,
  children: [
    {path: 'linkman', component: linkman},
    {path: 'address', component: address}
  ]
}
```

(3) 修改src/components/about.vue

```
<template>
  <div>
    关于我们
    <router-link to="/about/address" >地址</router-link>
    <router-link to="/about/linkman" >联系人</router-link>
    <router-view/>
  </div>
</template>
```

1.4 十次方的路由代码

我们现在通过看提供的代码来了解

(1) src/router/index.js



```
import Vue from 'vue'
import Router from 'vue-router'

Vue.use(Router)

/* Layout */
import Layout from '../views/layout/Layout'

export const constantRouterMap = [
  { path: '/login', component: () => import('@views/login/index'),
    hidden: true },
  { path: '/404', component: () => import('@views/404'), hidden: true },
  {
    path: '/',
    component: Layout,
    redirect: '/dashboard',
    name: 'Dashboard',
    hidden: true,
    children: [{
      path: 'dashboard',
      component: () => import('@views/dashboard/index')
    }]
  },
  {
    path: '/example',
    component: Layout,
    redirect: '/example/table',
    name: 'Example',
    meta: { title: 'Example', icon: 'example' },
    children: [
      {
        path: 'table',
        name: 'Table',
        component: () => import('@views/table/index'),
        meta: { title: 'Table', icon: 'table' }
      },
      {
        path: 'tree',
        name: 'Tree',

        component: () => import('@views/tree/index'),
```




```
      meta: { title: 'Tree', icon: 'tree' }
    }
  ],
},
{
  path: '/form',
  component: Layout,
  children: [
    {
      path: 'index',
      name: 'Form',
      component: () => import('@/views/form/index'),
      meta: { title: 'Form', icon: 'form' }
    }
  ]
},
{ path: '*', redirect: '/404', hidden: true }
]

export default new Router({
  // mode: 'history', //后端支持可开
  scrollBehavior: () => ({ y: 0 }),
  routes: constantRouterMap
})
```

(2) src/main.js

```
.....
import router from './router'
.....
new Vue({
  el: '#app',
  router,
  template: '<App/>',
  components: { App }
})
```

2 招聘管理

2.1 准备工作

2.1.1 代码生成

- (1) 使用《黑马程序员代码生成器》，连接数据库tensquare_recruit
- (2) 将api 与vue页面拷贝到当前工程

2.1.2 路由设置

```
{
  path: '/recruit',
  component: Layout,
  name: 'recruit',
  meta: { title: '招聘管理', icon: 'example' },
  children: [
    { path: 'enterprise', name: 'enterprise', component: () =>
import('@/views/table/enterprise'), meta: { title: '企业管理', icon:
'table' }},
    { path: 'recruit', name: 'recruit', component: () =>
import('@/views/table/recruit'), meta: { title: '招聘管理', icon: 'table'
}}
  ]
},
```

2.1.3 easyMock接口导入

将swaggerAPI文档导入到easyMock中。

2.2 企业管理

2.2.1 企业简介（文本域）

修改src/views/table/enterprise.vue

```
<el-form-item label="企业简介">
  <el-input v-model="pojo.summary" type="textarea" :rows="4"></el-input>

</el-form-item>
```

2.2.2 是否热门（开关）

修改src/views/table/enterprise.vue编辑窗口中是否热门部分

```
<el-form-item label="是否热门">
  <el-switch placeholder="是否热门" on-text="" off-text="" active-
value="1" inactive-value="0" v-model="pojo.ishot" ></el-switch>
</el-form-item>
```

2.3.3 网址输入（复合型输入框）

```
<el-input v-model="pojo.url" placeholder="请输入网址">
  <template slot="prepend">http://</template>
</el-input>
```

2.2.4 上传Logo

参见elementUI官方文档 <http://element-cn.eleme.io/#/zh-CN/component/upload>
（用户头像上传）实现Logo上传

（1）页面添加上传组件

```
<el-upload
  class="avatar-uploader"
  action="https://jsonplaceholder.typicode.com/posts/"
  :show-file-list="false"
  :on-success="handleAvatarSuccess"
  :before-upload="beforeAvatarUpload">
  
  <i v-else class="el-icon-plus avatar-uploader-icon"></i>
</el-upload>
```

action用于定义提交的服务器地址

show-file-list 是否显示已上传文件列表

before-upload 在上传之前被调用，用于判断图片类型和大小

on-success 在上传成功之后被调用，用于获取服务器上的文件名



(2) 添加样式:

```
<style>
.avatar-uploader .el-upload {
  border: 1px dashed #d9d9d9;
  border-radius: 6px;
  cursor: pointer;
  position: relative;
  overflow: hidden;
}
.avatar-uploader .el-upload:hover {
  border-color: #409EFF;
}
.avatar-uploader-icon {
  font-size: 28px;
  color: #8c939d;
  width: 100px;
  height: 50px;
  line-height: 50px;
  text-align: center;
}
.avatar {
  width: 100px;
  height: 50px;
  display: block;
}
</style>
```

(3) 代码:

data添加属性

```
data() {
  return {
    .....
    imageUrl: ''
  }
}
```

methods增加方法

```
handleAvatarSuccess(res, file) {
  this.imageUrl = URL.createObjectURL(file.raw);
  this.pojo.logo= this.imageUrl
},
beforeAvatarUpload(file) {
  const isJPG = file.type === 'image/jpeg';
  const isLt2M = file.size / 1024 / 1024 < 2;
  if (!isJPG) {
    this.$message.error('上传头像图片只能是 JPG 格式!');
  }
  if (!isLt2M) {
    this.$message.error('上传头像图片大小不能超过 2MB!');
  }
  return isJPG && isLt2M;
}
```

2.3 招聘管理

2.3.1 任职方式(单选按钮)

修改src/views/table/recruit.vue

```
<el-form-item label="任职方式">
  <el-radio v-model="pojo.type" label="1">全职</el-radio>
  <el-radio v-model="pojo.type" label="2">兼职</el-radio>
</el-form-item>
```

2.3.2 选择企业(可搜索下拉选择框)

(1) 修改src/views/table/recruit.vue 增加变量--企业列表

```
enterpriseList: []
```

(2) 修改created()

```
created() {  
  this.fetchData()  
  enterprise.getList().then(response => { // 企业列表  
    if (response.flag === true) {  
      this.enterpriseList = response.data  
    }  
  })  
},
```

(3) 修改弹出窗口部分，将文本框替换为下拉框

```
<el-form-item label="企业ID">  
  <el-select v-model="pojo.eid" filterable placeholder="请选择">  
    <el-option  
      v-for="item in enterpriseList"  
      :key="item.id"  
      :label="item.name"  
      :value="item.id">  
    </el-option>  
  </el-select>  
</el-form-item>
```

2.3.3 删除创建日期

创建日期是在后端自动生成的，所以要在弹出窗口中删除控件

2.3.4 状态（开关）

修改src/views/table/recruit.vue

```
<el-form-item label="状态">  
  <el-switch placeholder="是否热门" on-text="" off-text=""  
    active-value="1" inactive-value="0" v-model="pojo.state" ></el-switch>  
</el-form-item>
```

3 文章管理

3.1 准备工作

3.1.1 代码生成

- (1) 使用《黑马程序员代码生成器》，连接数据库tensquare_article
- (2) 将api与vue页面拷贝到当前工程

3.1.2 路由设置

```
{
  path: '/article',
  component: Layout,
  name: 'article',
  meta: { title: '文章管理', icon: 'example' },
  children: [
    { path: 'channel', name: 'channel', component: () =>
import('@/views/table/channel'), meta: { title: '频道管理', icon: 'table'
}},
    { path: 'column', name: 'column', component: () =>
import('@/views/table/column'), meta: { title: '专栏审核', icon: 'table'
}},
    { path: 'article', name: 'article', component: () =>
import('@/views/table/article'), meta: { title: '文章审核', icon: 'table'
}}
  ]
}
```

3.1.3 easyMock接口导入

将swaggerAPI文档导入到easyMock中。

3.2 频道管理

修改频道状态为开关，代码略

3.3 专栏审核

3.3.1 修改easyMock数据

URL: article/column/search/{page}/{size}

```
{
  "code": 20000,
  "flag": true,
  "message": "@string",
  "data": {
    "total": "@integer(60, 100)",
    "rows|10": [{
      "id": "@string",
      "name": "@cword(10,20)",
      "summary": "@cword(30,50)",
      "userid": "@string",
      "createtime": "@string",
      "checktime": "@string",
      "state|1": ['0', '1']
    }]
  }
}
```

3.3.2 待审核专栏列表

修改src/table/column.vue ,修改data变量的值

```
searchMap: {state:'0'},
```

这样在查询时就会携带状态为0的条件。

3.3.3 专栏审核

(1) 修改src/api/column.js ,新增专栏审核方法

```
examine(id){
  return request({
    url: `/${group_name}/${api_name}/examine/${id}`,
    method: 'put'
  })
}
```

(2) 增加方法定义



```
handleExamine(id){
    this.$confirm('确定要审核此纪录吗?', '提示', {
        confirmButtonText: '确定',
        cancelButtonText: '取消',
        type: 'warning'
    }).then(() => {
        columnApi.examine(id).then(response => {
            this.$message({ message: response.message, type: (response.flag
? 'success' : 'error') })
            if (response.flag) {
                this.fetchData() // 刷新数据
            }
        })
    })
}
```

(3) 审核按钮

```
<el-button @click="handleExamine(scope.row.id)" type="text" size="small">
审核</el-button>
```

3.4 文章审核

3.4.1 修改easyMock接口

URL: /article/article/search/{page}/{size}

```
{
  "code": 20000,
  "flag": true,
  "message": "@string",
  "data": {
    "total": "@integer(60, 100)",
    "rows|10": [{
      "id": "@string",
      "columnid": "@string",
      "userid": "@string",
      "title": "@cword(20,30)",
      "content": "@string",
      "image": "@string",
      "createtime": "@string",
      "updatetime": "@string",
      "ispublic": "@string",
      "istop": "@string",
      "visits": "@string",
      "thumbup": "@string",
      "comment": "@string",
      "state|1": ['1', '0'],
      "channelid": "@string",
      "url": "@string",
      "type": "@string"
    }]
  }
}
```

3.4.2 待审核文章列表

修改src/table/article.vue ,修改data变量的值

```
searchMap: {state:'0'},
```

对查询表单进行精简



```
<!-- 查询表单 -->
<el-form :inline="true" class="demo-form-inline">
  <el-form-item label="标题">
<el-input v-model="searchMap.title" placeholder="标题"></el-input></el-
form-item>
    <el-form-item label="文章正文">
<el-input v-model="searchMap.content" placeholder="文章正文"></el-input>
</el-form-item>
    <el-button type="primary" @click="fetchData()">查询</el-button>
    <el-button type="primary" icon="el-icon-circle-plus"
@click="handleEdit('')">新增</el-button>
  </el-form>
```

对表格列进行精简

```
<el-table-column prop="id" label="ID" width="80"></el-table-column>
  <el-table-column prop="columnid" label="专栏ID" width="80"></el-
table-column>
  <el-table-column prop="userid" label="用户ID" width="80"></el-
table-column>
  <el-table-column prop="title" label="标题" width="80"></el-
table-column>
  <el-table-column prop="image" label="文章封面" width="80"></el-
table-column>
  <el-table-column prop="createtime" label="发表日期" width="80">
</el-table-column>
  <el-table-column prop="ispublic" label="是否公开" width="80">
</el-table-column>
  <el-table-column prop="istop" label="是否置顶" width="80"></el-
table-column>
  <el-table-column prop="state" label="审核状态" width="80"></el-
table-column>
  <el-table-column prop="channelid" label="所属频道" width="80">
</el-table-column>
  <el-table-column prop="url" label="URL" width="80"></el-table-
column>
  <el-table-column prop="type" label="类型" width="80"></el-table-
column>
```

删除“新增”按钮

3.4.3 文章详情窗口

点击“详情”按钮打开窗口，显示标题和正文 v-html用于显示富文本内容。

```
<!--编辑窗口-->
<el-dialog title="详情" :visible.sync="dialogFormVisible" >
  {{pojo.title}}
  <hr>
  <div v-html='pojo.content'></div>
</el-dialog>
```

3.4.4 文章审核与删除

(1) 修改src/api/article.js，增加文章审核的方法

```
examine(id){
  return request({
    url: `/${group_name}/${api_name}/examine/${id}`,
    method: 'put'
  })
}
```

(2) 修改src/views/table/article.vue，增加方法



```
handleExamine(id){
  this.$confirm('确定要审核此纪录吗?', '提示', {
    confirmButtonText: '确定',
    cancelButtonText: '取消',
    type: 'warning'
  }).then(() => {
    articleApi.examine(id).then(response => {
      this.$message({ message: response.message, type: (response.flag
? 'success' : 'error') })
      if (response.flag) {
        this.fetchData() // 刷新数据
      }
      this.dialogFormVisible = false
    })
  })
}
```

(3) 新增审核和删除按钮

```
<el-button type="success" @click="handleExamine(pojo.id)" >审核通过</el-
button>
<el-button type="danger" @click="handleDelete(pojo.id)" >删除</el-button>
<el-button @click="dialogFormVisible = false">关闭</el-button>
```

(4) 删除方法添加代码

```
this.dialogFormVisible = false // 隐藏窗口
```

4 状态管理Vuex

我们经过测试会发现，用户登陆后可以访问其它页面的资源。未登录或退出登录后，再次访问资源会跳回到登陆页，这是如何实现的呢？长话短说，这是通过一种叫Vuex的技术来实现的。

4.1 Vuex简介

官方的解释：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。

快速理解：每个组件都有它自己数据属性，封装在data()中，每个组件之间data是完全隔离的，是私有的。如果我们需要各个组件都能访问到数据数据，或是需要各个组件之间能互相交换数据，这就需要一个单独存储的区域存放公共属性。这就是状态管理所要解决的问题。

4.2 快速入门

4.2.1 工程搭建

```
# 创建一个基于 webpack 模板的新项目
vue init webpack vuexdemo
# 安装依赖，走你
cd vuexdemo
cnpm install --save vuex
npm run dev
```

4.2.2 读取状态值

每一个 Vuex 应用的核心就是 store（仓库）。“store”基本上就是一个容器，它包含着你的应用中大部分的状态 (state)。

实现步骤：

（1）在src下创建store，store下创建index.js

```
import Vue from 'vue'
import Vuex from 'vuex'
Vue.use(Vuex)
const store = new Vuex.Store({
  state: {
    count: 0
  }
})
export default store
```



(2) 修改main.js，引入和装载store

```
import Vue from 'vue'
import App from './App'
import router from './router'
import store from './store'
Vue.config.productionTip = false
new Vue({
  el: '#app',
  router,
  store,
  components: { App },
  template: '<App/>'
})
```

(3) 修改components\HelloWorld.vue

```
<template>
  <div>
    {{ $store.state.count }}
    <button @click="showCount">测试</button>
  </div>
</template>
<script>
export default {
  methods: {
    showCount() {
      console.log(this.$store.state.count)
    }
  }
}
</script>
```

4.2.3 改变状态值

你不能直接改变 store 中的状态。改变 store 中的状态的唯一途径就是显式地提交 **(commit) mutation**。这样使得我们可以方便地跟踪每一个状态的变化，从而让我们能够实现一些工具帮助我们更好地了解我们的应用。

(1) 修改store/index.js,增加mutation定义

```
import Vue from 'vue'
import Vuex from 'vuex'
Vue.use(Vuex)
const store=new Vuex.Store({
  state: {
    count: 0
  },
  mutations: {
    increment(state) {
      state.count++
    }
  }
})
export default store
```

(2) 修改components\HelloWorld.vue，调用mutation

```
<template>
  <div>
    {{ $store.state.count }}
    <button @click="addCount">测试</button>
  </div>
</template>
<script>
export default {
  methods:{
    addCount(){
      this.$store.commit('increment')
    }
  }
}
</script>
```

测试：运行工程，点击测试按钮，我们会看到控制台和页面输出递增的数字

4.2.4 状态值共享测试

如果是另外一个页面，能否读取到刚才我在HelloWorld中操作的状态值呢？我们接下来就做一个测试

(1) 在components下创建show.vue

```
<template>
  <div>
    show: {{$store.state.count}}
  </div>
</template>
```

(2) 修改路由设置 router/index.js

```
import Vue from 'vue'
import Router from 'vue-router'
import HelloWorld from '@/components/HelloWorld'
import Show from '@/components/Show'
Vue.use(Router)
export default new Router({
  routes: [
    {
      path: '/',
      name: 'HelloWorld',
      component: HelloWorld
    },
    {
      path: '/show',
      name: 'Show',
      component: Show
    }
  ]
})
```

测试：在HelloWorld页面点击按钮使状态值增长，然后再进入show页面查看状态值

4.2.5 提交载荷

所谓载荷（payload）就是 向 store.commit 传入额外的参数。

(1) 修改store下的index.js



```
.....  
mutations: {  
  increment (state,x) {  
    state.count += x  
  }  
}  
.....
```

(2) 修改HelloWorld.vue

```
.....  
addCount(){  
  this.$store.commit('increment',10)  
  console.log(this.$store.state.count)  
}  
.....
```

4.2.6 Action

Action 类似于 mutation，不同在于：

- Action 提交的是 mutation，而不是直接变更状态。
- Action 可以包含任意异步操作。

我们现在使用 Action 来封装increment

(1) 修改store/index.js

```
const store = new Vuex.Store({  
  .....  
  actions: {  
    increment (context){  
      context.commit('increment',10)  
    }  
  }  
})
```

(2) 修改show.vue



```
<template>
  <div>
    show: {{$store.state.count}}

    <button @click="addCount">测试</button>
  </div>
</template>
<script>
export default {
  methods:{
    addCount(){
      this.$store.dispatch('increment')
      console.log(this.$store.state.count)
    }
  }
}
```

我们使用dispatch来调用action，Action也同样支持载荷

4.2.7 派生属性Getter

有时候我们需要从store中的state中派生出一些状态，例如我们在上例代码的基础上，我们增加一个叫remark的属性，如果count属性值小于50则remark为加油，大于等于50小于100则remark为你真棒，大于100则remark的值为你是大神。这时我们就需要用到getter为我们解决。

(1) 修改store/index.js，增加getters定义



```
const store = new Vuex.Store({
  .....
  getters: {
    remark(state){
      if(state.count<50){
        return '加油'
      }else if( state.count<100){
        return '你真棒'
      }else{
        return '你是大神'
      }
    }
  }
  .....
})
```

Getter 接受 state 作为其第一个参数，也可以接受其他 getter 作为第二个参数

(2) 修改HelloWorld.vue 显示派生属性的值

```
{{ $store.getters.remark }}
```

4.3 模块化

4.3.1 Module

由于使用单一状态树，应用的所有状态会集中到一个比较大的对象。当应用变得非常复杂时，store 对象就有可能变得相当臃肿。

为了解决以上问题，Vuex 允许我们将 store 分割成模块（**module**）。每个模块拥有自己的 state、mutation、action、getter、甚至是嵌套子模块——从上至下进行同样方式的分割。参见以下代码模型



```
const moduleA = {
  state: { ... },
  mutations: { ... },
  actions: { ... },
  getters: { ... }
}

const moduleB = {
  state: { ... },
  mutations: { ... },
  actions: { ... }
}

const store = new Vuex.Store({
  modules: {
    a: moduleA,
    b: moduleB
  }
})

store.state.a // -> moduleA 的状态
store.state.b // -> moduleB 的状态
```

我们现在就对工程按模块化进行改造

(1) 修改store/index.js



```
import Vue from 'vue'
import Vuex from 'vuex'
Vue.use(Vuex)
const moduleA = {
  state: {
    count: 0
  },
  getters: {
    remark(state){
      if(state.count<50){
        return '加油'
      }else if( state.count<100){
        return '你真棒'
      }else{
        return '你是大神'
      }
    }
  },
  mutations: {
    increment (state,x) {
      state.count += x
    }
  },
  actions: {
    increment (context){
      context.commit('increment',10)
    }
  }
}
const store = new Vuex.Store({
  modules: {
    a:moduleA
  }
})
export default store
```

(2) 修改HelloWorld.vue和show.vue

```
{{ $store.state.a.count }}
```

4.3.2 标准工程结构

如果所有的状态都写在一个js中，这个js必定会很臃肿，所以Vuex建议你按以下代码结构来构建工程

```
|— index.html
|— main.js
|— api
|   |— ... # 抽取API请求
|— components
|   |— App.vue
|   |— ...
|— store
|   |— index.js          # 我们组装模块并导出 store 的地方
|   |— getters.js
|   |— modules
|       |— a.js         # A模块
|       |— b.js         # B模块
```

我们现在就按照上面的结构，重新整理以下我们的代码：

(1) store下创建modules文件夹，文件夹下创建a.js

```
export default {
  state: {
    count: 0
  },
  mutations: {
    increment (state,x) {
      state.count += x
    }
  },
  actions: {
    increment (context){
      context.commit('increment',10)
    }
  }
}
```

(2) store下创建getters.js



```
export default {
  remark: state => {
    if(state.a.count<50){
      return '加油'
    }else if( state.a.count<100){
      return '你真棒'
    }else{
      return '你是大神'
    }
  },
  count: state=> state.a.count
}
```

(3) 修改store/index.js

```
import Vue from 'vue'
import Vuex from 'vuex'
import a from './modules/a'
import getters from './getters'
Vue.use(Vuex)

const store = new Vuex.Store({
  getters,
  modules: {
    a
  }
})

export default store
```

(4) 修改HelloWorld.vue


```
<template>
  <div>
    {{ $store.getters.count }}    {{ $store.getters.remark }}
    <button @click="addCount">测试</button>
  </div>
</template>

<script>
export default {
  methods: {
    addCount() {
      this.$store.commit('increment', 10)
      console.log(this.$store.getters.count)
    }
  }
}
</script>
```

4.4 十次方后台登陆（课下阅读）

脚手架已经实现了登陆部分的代码，只需学员课下阅读，不需要编写，理解实现思路即可。

4.4.1 登陆

（1）src/api下创建login.js



```
import request from '@/utils/request'

export function login(username, password) {
  return request({
    url: '/user/login',
    method: 'post',
    data: {
      username,
      password
    }
  })
}
```

(2) src下建立store文件夹，store下创建modules，modules下创建user.js



```
import { login, logout, getInfo } from '@api/login'
import { getToken, setToken, removeToken } from '@utils/auth'

const user = {
  state: {
    token: getToken(),
    name: '',
    avatar: '',
    roles: []
  },
  mutations: {
    SET_TOKEN: (state, token) => {
      state.token = token
    },
    SET_NAME: (state, name) => {
      state.name = name
    },
    SET_AVATAR: (state, avatar) => {
      state.avatar = avatar
    },
    SET_ROLES: (state, roles) => {
      state.roles = roles
    }
  },
  actions: {
    // 登录
    Login({ commit }, userInfo) {
      const username = userInfo.username.trim()
      return new Promise((resolve, reject) => {
        login(username, userInfo.password).then(response => {
          const data = response.data
          setToken(data.token)
          commit('SET_TOKEN', data.token)
          resolve()
        }).catch(error => {
          reject(error)
        })
      })
    }
  }
}
```



```
    }  
  }  
}  
export default user
```

(3) store下创建getters.js

```
const getters = {  
  token: state => state.user.token,  
  avatar: state => state.user.avatar,  
  name: state => state.user.name,  
  roles: state => state.user.roles  
}  
export default getters
```

(4) store下创建index.js

```
import Vue from 'vue'  
import Vuex from 'vuex'  
import user from './modules/user'  
import getters from './getters'  
  
Vue.use(Vuex)  
  
const store = new Vuex.Store({  
  modules: {  
    user  
  },  
  getters  
})  
  
export default store
```

(5) 修改src下的main.js，引入store

```
import store from './store'

.....
new Vue({
  el: '#app',
  router,
  store,
  template: '<App/>',
  components: { App }
})
```

(6) 构建登陆页面.在src/views/login/index.vue



```
<template>
  <div class="login-container">
    <el-form autoComplete="on" :model="loginForm" :rules="loginRules"
ref="loginForm" label-position="left" label-width="0px"
    class="card-box login-form">
      <h3 class="title">十次方管理后台</h3>
      <el-form-item prop="username">
        <span class="svg-container svg-container_login">
          <svg-icon icon-class="user" />
        </span>
        <el-input name="username" type="text" v-
model="loginForm.username" autoComplete="on" placeholder="username" />
      </el-form-item>
      <el-form-item prop="password">
        <span class="svg-container">
          <svg-icon icon-class="password"></svg-icon>
        </span>
        <el-input name="password" :type="pwdType"
@keyup.enter.native="handleLogin" v-model="loginForm.password"
autoComplete="on"
          placeholder="password"></el-input>
        <span class="show-pwd" @click="showPwd"><svg-icon icon-
class="eye" /></span>
      </el-form-item>
      <el-form-item>
        <el-button type="primary" style="width:100%;" :loading="loading"
@click.native.prevent="handleLogin">
          Sign in
        </el-button>
      </el-form-item>
      <div class="tips">
        <span style="margin-right:20px;">username: admin</span>
        <span>password: admin</span>
      </div>
    </el-form>
  </div>
</template>
<script>
import { isValidUsername } from '@/utils/validate'

export default {
```



```
name: 'login',
data() {
  const validateUsername = (rule, value, callback) => {
    if (!isValidUsername(value)) {
      callback(new Error('请输入正确的用户名'))
    } else {
      callback()
    }
  }
  const validatePass = (rule, value, callback) => {
    if (value.length < 5) {
      callback(new Error('密码不能小于5位'))
    } else {
      callback()
    }
  }
  return {
    loginForm: {
      username: 'admin',
      password: 'admin'
    },
    loginRules: {
      username: [{ required: true, trigger: 'blur', validator:
validateUsername }],
      password: [{ required: true, trigger: 'blur', validator:
validatePass }]
    },
    loading: false,
    pwdType: 'password'
  },
  methods: {
    showPwd() {
      if (this.pwdType === 'password') {
        this.pwdType = ''
      } else {
        this.pwdType = 'password'
      }
    },
    handleLogin() {
      this.$refs.loginForm.validate(valid => {
```

```
    if (valid) {
      this.loading = true
      this.$store.dispatch('Login', this.loginForm).then(() => {
        this.loading = false
        this.$router.push({ path: '/' })
      }).catch(() => {
        this.loading = false
      })
    } else {
      console.log('error submit!!')
      return false
    }
  })
}
}
</script>
....样式略
```

4.4.2 获取用户登陆信息

(1) 修改src/api/login.js

```
export function getInfo(token) {
  return request({
    url: '/user/info',
    method: 'get',
    params: { token }
  })
}
```

(2) 修改src/store/modules/user.js,增加action方法



```
// 获取用户信息
GetInfo({ commit, state }) {
  return new Promise((resolve, reject) => {
    getInfo(state.token).then(response => {
      const data = response.data
      commit('SET_ROLES', data.roles)
      commit('SET_NAME', data.name)
      commit('SET_AVATAR', data.avatar)
      resolve(response)
    }).catch(error => {
      reject(error)
    })
  })
},
```

(3) 在src下创建permission.js，实现用户信息的拉取



```
import router from './router'
import store from './store'
import NProgress from 'nprogress' // Progress 进度条
import 'nprogress/nprogress.css' // Progress 进度条样式
import { Message } from 'element-ui'
import { getToken } from '@/utils/auth' // 验权

const whiteList = ['/login'] // 不重定向白名单
router.beforeEach((to, from, next) => {
  NProgress.start()
  if (getToken()) {
    if (to.path === '/login') {
      next({ path: '/' })
    } else {
      if (store.getters.roles.length === 0) {
        store.dispatch('GetInfo').then(res => { // 拉取用户信息
          next()
        }).catch(() => {
          store.dispatch('FedLogOut').then(() => {
            Message.error('验证失败,请重新登录')
            next({ path: '/login' })
          })
        })
      } else {
        next()
      }
    }
  } else {
    if (whiteList.indexOf(to.path) !== -1) {
      next()
    } else {
      next('/login')
      NProgress.done()
    }
  }
})

router.afterEach(() => {
  NProgress.done() // 结束Progress

})
```

(4) 在顶部导航栏中实现头像的读取。

修改src\views\layout\components\Navbar.vue

```
<script>
import { mapGetters } from 'vuex'
import Breadcrumb from '@/components/Breadcrumb'
import Hamburger from '@/components/Hamburger'

export default {
  components: {
    Breadcrumb,
    Hamburger
  },
  computed: {
    ...mapGetters([
      'sidebar',
      'avatar'
    ])
  },
  methods: {
    toggleSideBar() {
      this.$store.dispatch('ToggleSideBar')
    },
    logout() {
      this.$store.dispatch('LogOut').then(() => {
        location.reload() // 为了重新实例化vue-router对象 避免bug
      })
    }
  }
}
</script>
```

读取头像

```
<div class="avatar-wrapper">
  
  <i class="el-icon-caret-bottom"></i>
</div>
```

4.4.3退出登录

(1) 修改src/api/login.js

```
export function logout() {  
  return request({  
    url: '/user/logout',  
    method: 'post'  
  })  
}
```

(2) 修改src/store/modules/user.js,增加action方法

```
// 登出  
LogOut({ commit, state }) {  
  return new Promise((resolve, reject) => {  
    logout(state.token).then(() => {  
      commit('SET_TOKEN', '')  
      commit('SET_ROLES', [])  
      removeToken()  
      resolve()  
    }).catch(error => {  
      reject(error)  
    })  
  })  
},
```

(3) 在顶部导航栏中实现退出登录

```
logout() {  
  this.$store.dispatch('LogOut').then(() => {  
    location.reload() // 为了重新实例化vue-router对象 避免bug  
  })  
}
```

```
<el-dropdown-item divided>  
  <span @click="logout" style="display:block;">退出登录</span>  
</el-dropdown-item>
```



黑马程序员
www.itheima.com

传智播客旗下
高端IT教育品牌

改变中国IT教育，我们正在行动