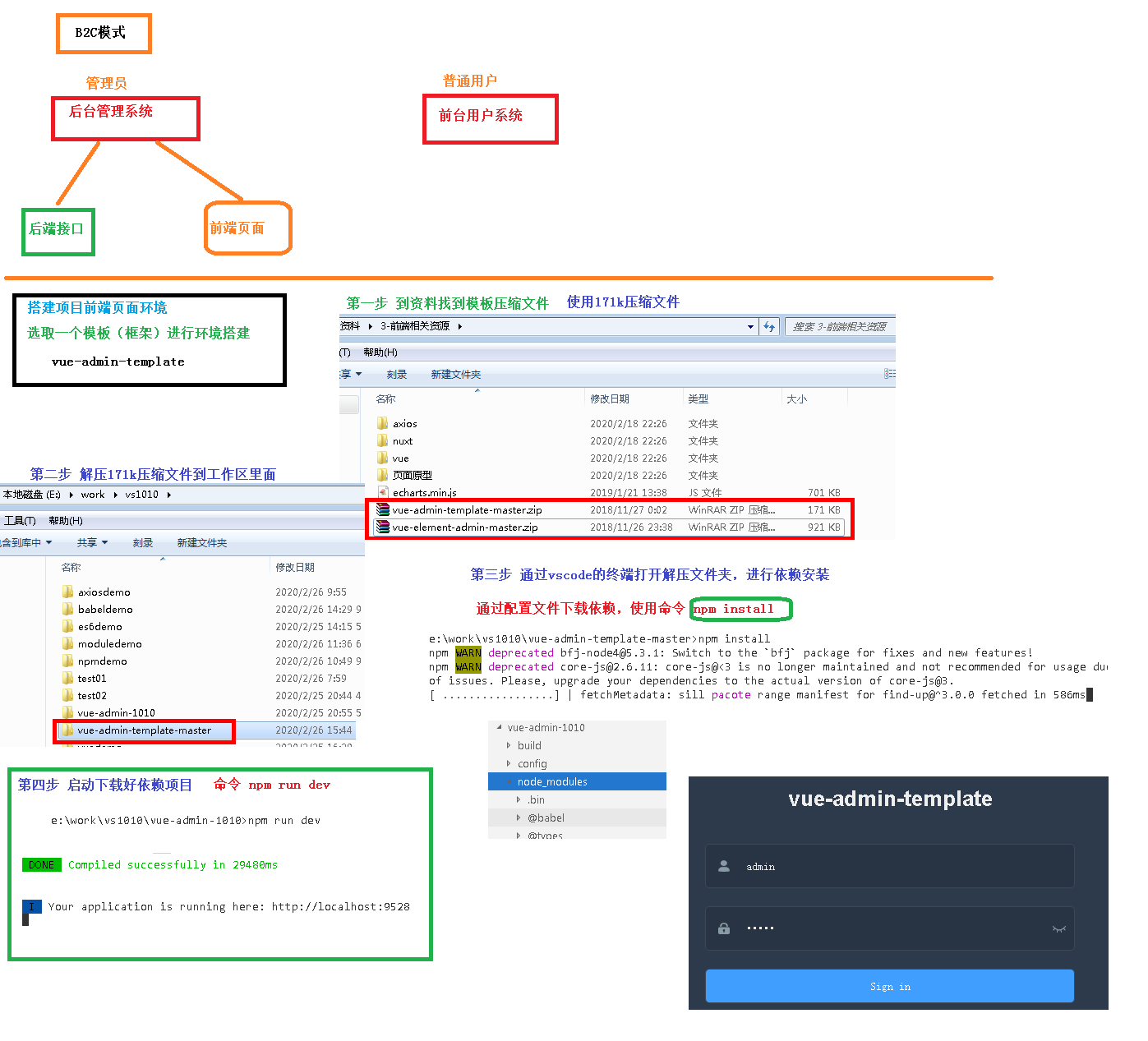
# 9-vue-element-admin



## 一、vue-element-admin

### 1、简介

而vue-element-admin是基于element-ui 的一套后台管理系统集成方案。

**功能：**[https://panjiachen.github.io/vue-element-admin-site/zh/guide/#功能](https://panjiachen.github.io/vue-element-admin-site/zh/guide/" \l "%E5%8A%9F%E8%83%BD)

**GitHub地址：**<https://github.com/PanJiaChen/vue-element-admin>

**项目在线预览：**[https://panjiachen.gitee.io/vue-element-admin](https://panjiachen.gitee.io/vue-element-admin/" \l "/login?redirect=%2Fdashboard)

### 2、安装

# 解压压缩包

# 进入目录  
cd vue-element-admin-master  
 # 安装依赖  
npm install  
 # 启动。执行后，浏览器自动弹出并访问http://localhost:9527/  
npm run dev

## 二、vue-admin-template

### 1、简介

vueAdmin-template是基于vue-element-admin的一套后台管理系统基础模板（最少精简版），可作为模板进行二次开发。

**GitHub地址：**<https://github.com/PanJiaChen/vue-admin-template>

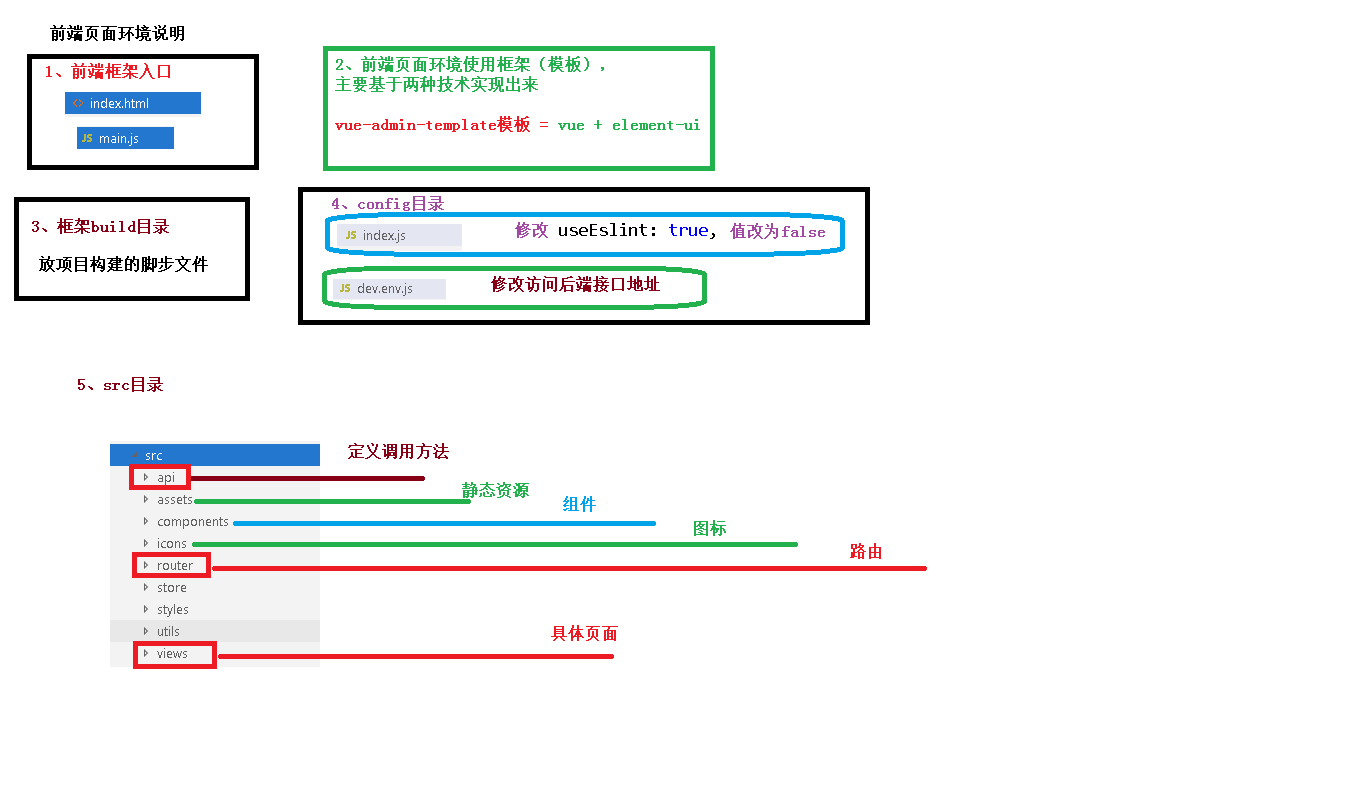
**建议：**你可以在 vue-admin-template 的基础上进行二次开发，把 vue-element-admin当做工具箱，想要什么功能或者组件就去 vue-element-admin 那里复制过来。

### 2、安装

# 解压压缩包

# 进入目录  
cd vue-admin-template-master  
# 安装依赖  
npm install  
# 启动。执行后，浏览器自动弹出并访问http://localhost:9528/  
npm run dev

# 10-后台系统前端项目创建



## 一、项目的创建和基本配置

### 1、创建项目

将vue-admin-template-master重命名为guli-admin

### 2、修改项目信息

package.json

{

"name": "guli-admin",

......

"description": "谷粒学院后台管理系统",

"author": "Helen <55317332@qq.com>",

......

}

### 3、如果需要修改端口号

config/index.js中修改

port: 9528

### 4、项目的目录结构

.

├── build // 构建脚本

├── config // 全局配置

├── node\_modules // 项目依赖模块

├── src //项目源代码

├── static // 静态资源

└── package.jspon // 项目信息和依赖配置

src

├── api // 各种接口

├── assets // 图片等资源

├── components // 各种公共组件，非公共组件在各自view下维护

├── icons //svg icon

├── router // 路由表

├── store // 存储

├── styles // 各种样式

├── utils // 公共工具，非公共工具，在各自view下维护

├── views // 各种layout

├── App.vue //\*\*\*项目顶层组件\*\*\*

├── main.js //\*\*\*项目入口文件\*\*\*

└── permission.js //认证入口

### 5、运行项目

npm run dev

## 二、登录页修改

src/views/login/index.vue（登录组件）

4行

<h3 class="title">谷粒学院后台管理系统</h3>

28行

<el-button :loading="loading" type="primary" style="width:100%;" @click.native.prevent="handleLogin">

登录

</el-button>

## 三、页面零星修改

### 1、标题

index.html（项目的html入口）

<title>谷粒学院后台管理系统</title>

修改后热部署功能，浏览器自动刷新

### 2、国际化设置

打开 src/main.js（项目的js入口），第7行，修改语言为 zh-CN，使用中文语言环境，例如：日期时间组件

import locale from 'element-ui/lib/locale/lang/zh-CN' // lang i18n

### 3、icon

复制 favicon.ico 到根目录

### 4、导航栏文字

src/views/layout/components（当前项目的布局组件）

src/views/layout/components/Navbar.vue

13行

<el-dropdown-item>

首页

</el-dropdown-item>

17行

<span style="display:block;" @click="logout">退出</span>

### 5、面包屑文字

src/components（可以在很多项目中复用的通用组件）

src/components/Breadcrumb/index.vue

38行

meta: { title: '首页' }

## 四、Eslint语法规范型检查

### 1、ESLint简介

JavaScript 是一个动态的弱类型语言，在开发中比较容易出错。因为没有编译程序，为了寻找 JavaScript 代码错误通常需要在执行过程中不断调适。

ESLint 是一个语法规则和代码风格的检查工具，可以用来保证写出语法正确、风格统一的代码。让程序员在编码的过程中发现问题而不是在执行的过程中。

### 2、语法规则

ESLint 内置了一些规则，也可以在使用过程中自定义规则。

本项目的语法规则包括：两个字符缩进，必须使用单引号，不能使用双引号，语句后不可以写分号，代码段之间必须有一个空行等。

### 3、确认开启语法检查

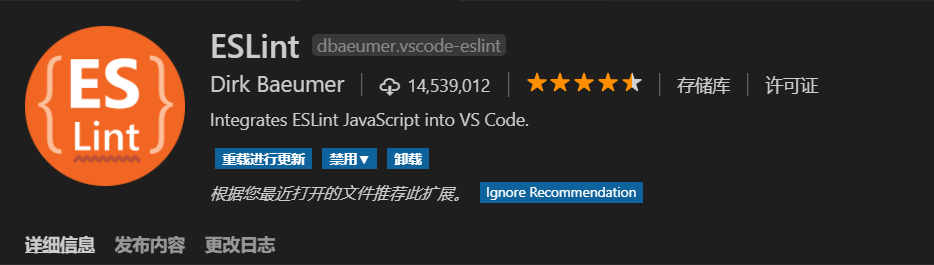
打开 config/index.js，配置是否开启语法检查

useEslint: true,

可以关闭语法检查，建议开启，养成良好的编程习惯。

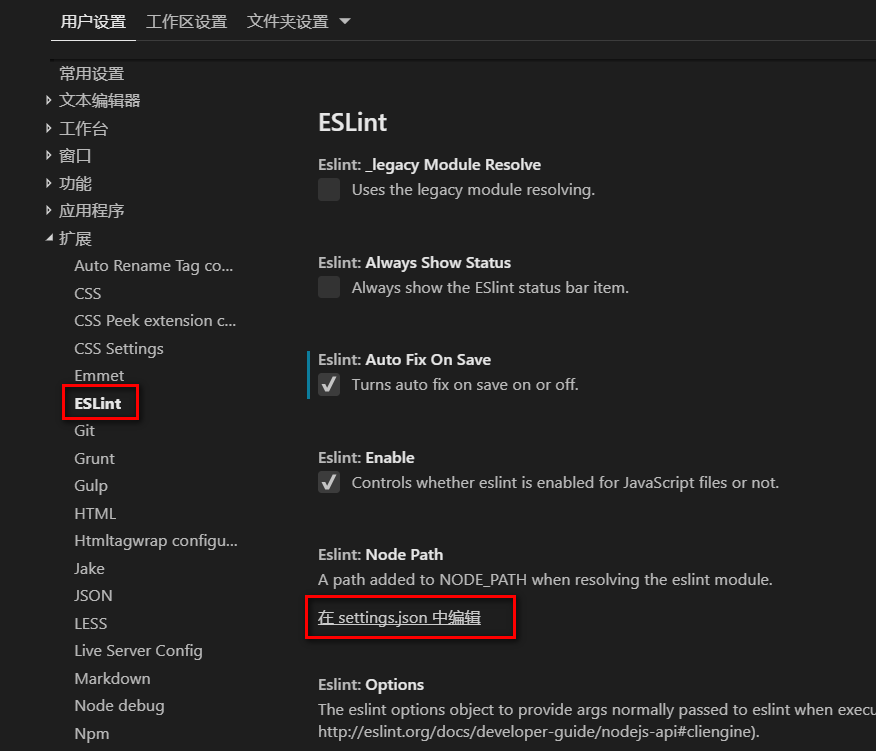
### 4、ESLint插件安装

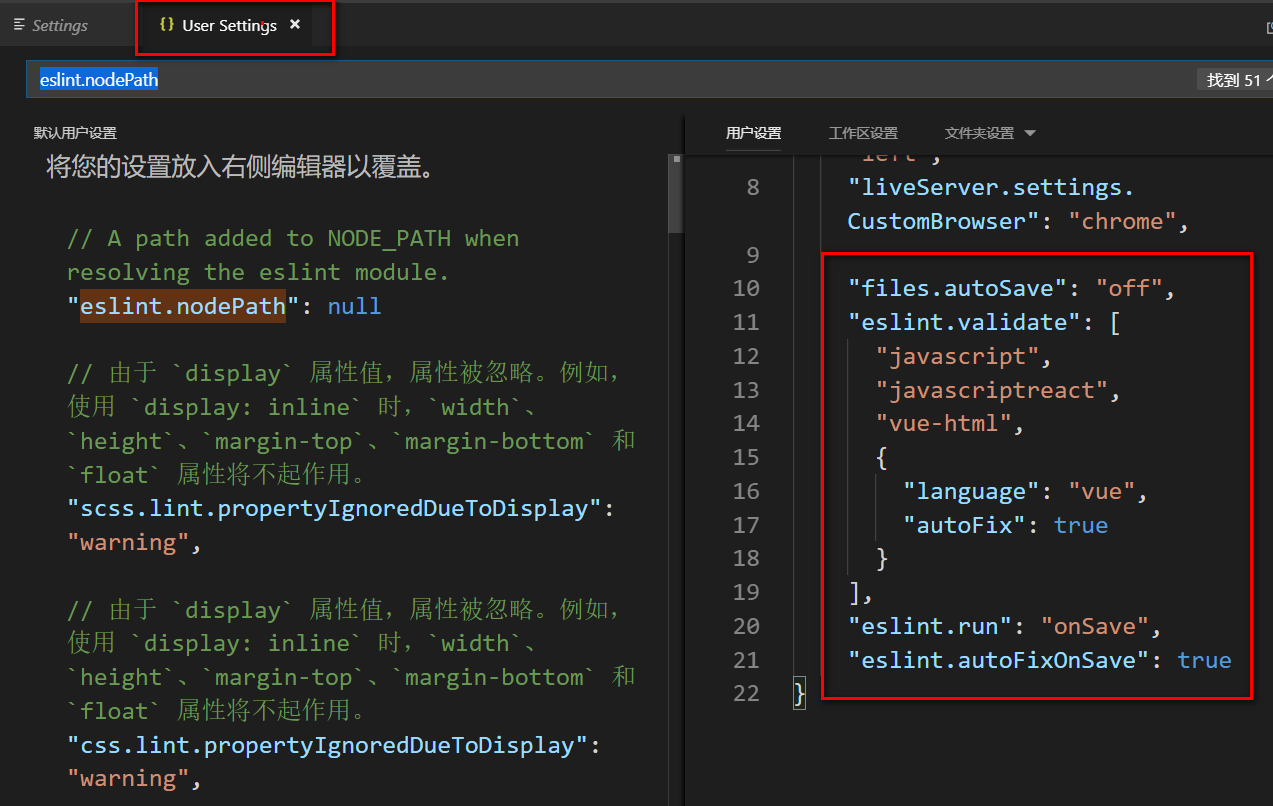
vs code的ESLint插件，能帮助我们自动整理代码格式



### 5、插件的扩展设置

选择vs code左下角的“设置”， 打开 VSCode 配置文件,添加如下配置





"files.autoSave": "off",

"eslint.validate": [

"javascript",

"javascriptreact",

"vue-html",

{

"language": "vue",

"autoFix": true

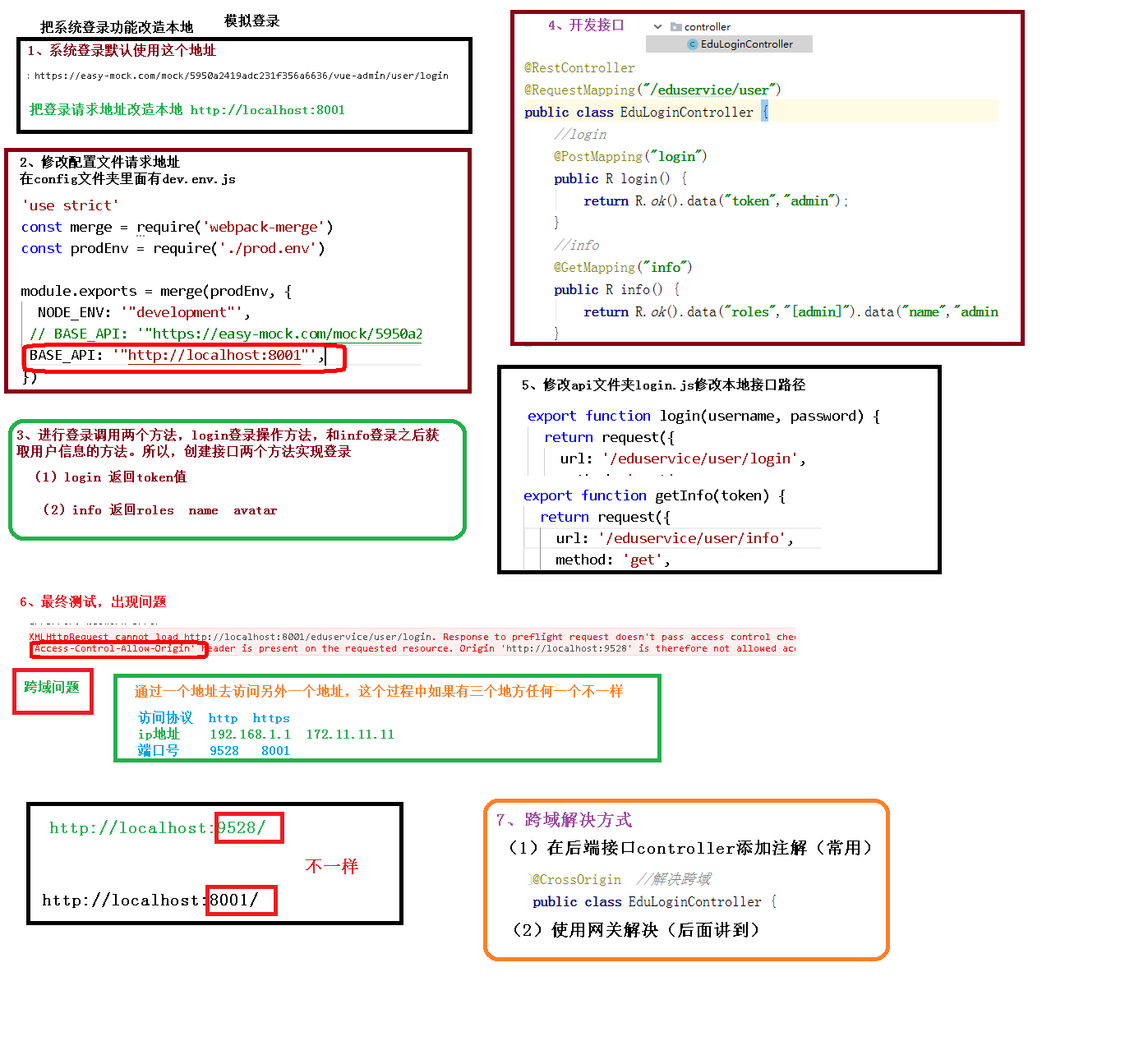
}

],

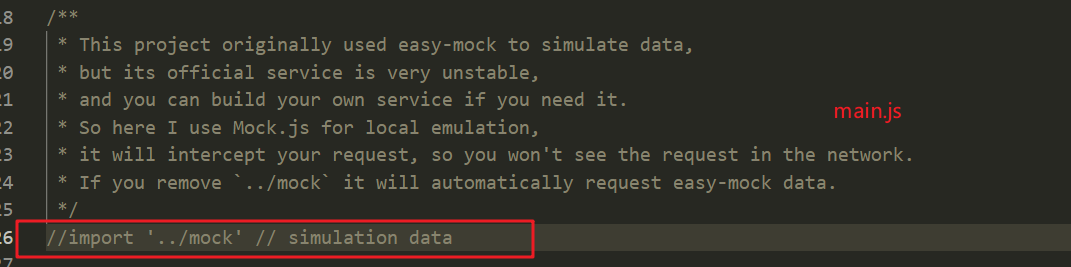
"eslint.run": "onSave",

"eslint.autoFixOnSave": true

# 项目登陆改造



注释掉框架的模拟数据



# 项目中的路由



## 一、后台系统路由实现分析

### 1、入口文件中调用路由

src/main.js

......

import router from './router' //引入路由模块

......

new Vue({

el: '#app',

router, //挂载路由

store,

render: h => h(App)

})

### 2、路由模块中定义路由

src/router/index.js

......

export const constantRouterMap = [

......

]

export default new Router({

......

routes: constantRouterMap

})

## 二、谷粒学院路由定义

### 1、复制icon图标

将vue-element-admin/src/icons/svg 中的图标复制到 guli-admin项目中

### 2、修改路由

修改 src/router/index.js 文件，重新定义constantRouterMap

**注意：**每个路由的name不能相同

export const constantRouterMap = [

{ path: '/login', component: () => import('@/views/login/index'), hidden: true },

{ path: '/404', component: () => import('@/views/404'), hidden: true },

// 首页

{

path: '/',

component: Layout,

redirect: '/dashboard',

name: 'Dashboard',

children: [{

path: 'dashboard',

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

meta: { title: '谷粒学院后台首页', icon: 'dashboard' }

}]

},

// 讲师管理

{

path: '/edu/teacher',

component: Layout,

redirect: '/edu/teacher/list',

name: 'Teacher',

meta: { title: '讲师管理', icon: 'peoples' },

children: [

{

path: 'list',

name: 'EduTeacherList',

component: () => import('@/views/edu/teacher/list'),

meta: { title: '讲师列表' }

},

{

path: 'create',

name: 'EduTeacherCreate',

component: () => import('@/views/edu/teacher/form'),

meta: { title: '添加讲师' }

},

{

path: 'edit/:id',

name: 'EduTeacherEdit',

component: () => import('@/views/edu/teacher/form'),

meta: { title: '编辑讲师', noCache: true },

hidden: true

}

]

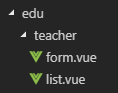
},

{ path: '\*', redirect: '/404', hidden: true }

]

### 3、创建vue组件

在src/views文件夹下创建以下文件夹和文件



### 4、form.vue

<template>

<div class="app-container">

讲师表单

</div>

</template>

### 5、list.vue

<template>

<div class="app-container">

讲师列表

</div>

</template>

# 使用nginx配置后台多服务器api

## 一、项目中的Easy Mock

config/dev.env.js 中BASE\_API 为项目的easymock地址，目前具有模拟登录、登出、获取用户信息的功能

BASE\_API: '"https://easy-mock.com/mock/5950a2419adc231f356a6636/vue-admin"',

**登录：**/user/login

**获取用户信息：**/user/info?token=admin

**登出：**/user/logout

config/dev.env.js，只有一个api地址的配置位置，而我们实际的后端有很多微服务，所以接口地址有很多，

我们可以使用nginx反向代理让不同的api路径分发到不同的api服务器中

## 二、配置nginx反向代理

### 1、安装window版的nginx

将nginx-1.12.0.zip解压到开发目录中

如：E:\development\nginx-1.12.0-guli-api

双击nginx.exe运行nginx

访问：localhost

### 2、配置nginx代理

在Nginx中配置对应的微服务服务器地址即可

注意：最好修改默认的 80端口到81

http {

server {

listen 81;

......

}，

......

server {

listen 8201;

server\_name localhost;

location ~ /edu/ {

proxy\_pass http://localhost:8101;

}

location ~ /user/ {

rewrite /(.+)$ /mock/5950a2419adc231f356a6636/vue-admin/$1 break;

proxy\_pass https://www.easy-mock.com;

}

}

}

### 3、重启nginx

nginx -s reload

### 4、测试

访问讲师列表接口：<http://localhost:8201/admin/edu/teacher>

访问获取用户信息接口：<http://localhost:8201/user/info?token=admin>

## 三、配置开发环境

### 1、修改config/dev.env.js

BASE\_API: '"http://127.0.0.1:8201"'

### 2、重启前端程序

修改配置文件后，需要手动重启前端程序

# 讲师管理列表组件



## 分页列表



### 1、定义api

创建文件 src/api/edu/teacher.js

import request from '@/utils/request'

*const* api\_name = '/admin/edu/teacher'

export default {

  getPageList(*page*, *limit*, *searchObj*) {

    return request({

      url: `${api\_name}/${*page*}/${*limit*}`,

      method: 'get',

      data: *searchObj*

    })

  }

}

### 2、初始化vue组件

src/views/edu/teacher/list.vue

<template>

  <div class="app-container">

    讲师列表

  </div>

</template>

<script>

import teacher from '@/api/edu/teacher'

export default {

  data() {// 定义数据

      return {}

  },

  created() { // 当页面加载时获取数据

    this.fetchData()

  },

  methods: {

    fetchData() { // 调用api层获取数据库中的数据

      console.log('加载列表')

    }

  }

}

</script>

### 3、定义data

  data() {

    return {

      listLoading: true, // 是否显示loading信息

      list: null, // 数据列表

      total: 0, // 总记录数

      page: 1, // 页码

      limit: 10, // 每页记录数

      searchObj: {}// 查询条件

    }

  },

### 4、定义methods

  methods: {

    fetchData(*page* = 1) { // 调用api层获取数据库中的数据

      console.log('加载列表')

      this.page = *page*

      this.listLoading = true

      teacher.getPageList(this.page, this.limit, this.searchObj).then(*response* *=>* {

        // debugger 设置断点调试

        if (*response*.success === true) {

          this.list = *response*.data.rows

          this.total = *response*.data.total

        }

        this.listLoading = false

      })

    }

  }

### 5、表格渲染

<!-- 表格 -->

    <el-table

      v-loading="listLoading"

      :data="list"

      element-loading-text="数据加载中"

      border

      fit

      highlight-current-row>

      <el-table-column

        label="序号"

        width="70"

        align="center">

        <template slot-scope="scope">

          {{ (page - 1) \* limit + scope.$index + 1 }}

        </template>

      </el-table-column>

      <el-table-column prop="name" label="名称" width="80" />

      <el-table-column label="头衔" width="80">

        <template slot-scope="scope">

          {{ scope.row.level===1?'高级讲师':'首席讲师' }}

        </template>

      </el-table-column>

      <el-table-column prop="intro" label="资历" />

      <el-table-column prop="gmtCreate" label="添加时间" width="160"/>

      <el-table-column prop="sort" label="排序" width="60" />

      <el-table-column label="操作" width="200" align="center">

        <template slot-scope="scope">

          <router-link :to="'/edu/teacher/edit/'+scope.row.id">

            <el-button type="primary" size="mini" icon="el-icon-edit">修改</el-button>

          </router-link>

          <el-button type="danger" size="mini" icon="el-icon-delete" @click="removeDataById(scope.row.id)">删除</el-button>

        </template>

      </el-table-column>

    </el-table>

### 6、分页组件

  <!-- 分页 -->

    <el-pagination

      :current-page="page"

      :page-size="limit"

      :total="total"

      style="padding: 30px 0; text-align: center;"

      layout="total, prev, pager, next, jumper"

      @current-change="fetchData"

    />

### 7、顶部查询表单

**注意：**

element-ui的 date-picker组件默认绑定的时间值是默认世界标准时间，和中国时间差8小时

设置 value-format="yyyy-MM-dd HH:mm:ss" 改变绑定的值

   <!--查询表单-->

    <el-form :inline="true" class="demo-form-inline">

      <el-form-item>

        <el-input v-model="searchObj.name" placeholder="讲师名" />

      </el-form-item>

      <el-form-item>

        <el-select v-model="searchObj.level" clearable placeholder="讲师头衔">

          <el-option :value="1" label="高级讲师" />

          <el-option :value="2" label="首席讲师" />

        </el-select>

      </el-form-item>

      <el-form-item label="添加时间">

        <el-date-picker

          v-model="searchObj.begin"

          type="datetime"

          placeholder="选择开始时间"

          value-format="yyyy-MM-dd HH:mm:ss"

          default-time="00:00:00"

        />

      </el-form-item>

      <el-form-item>

        <el-date-picker

          v-model="searchObj.end"

          type="datetime"

          placeholder="选择截止时间"

          value-format="yyyy-MM-dd HH:mm:ss"

          default-time="00:00:00"

        />

      </el-form-item>

      <el-button type="primary" icon="el-icon-search" @click="fetchData()"

        >查询</el-button

      >

      <el-button type="default" @click="resetData()">清空</el-button>

    </el-form>

清空方法

resetData() {

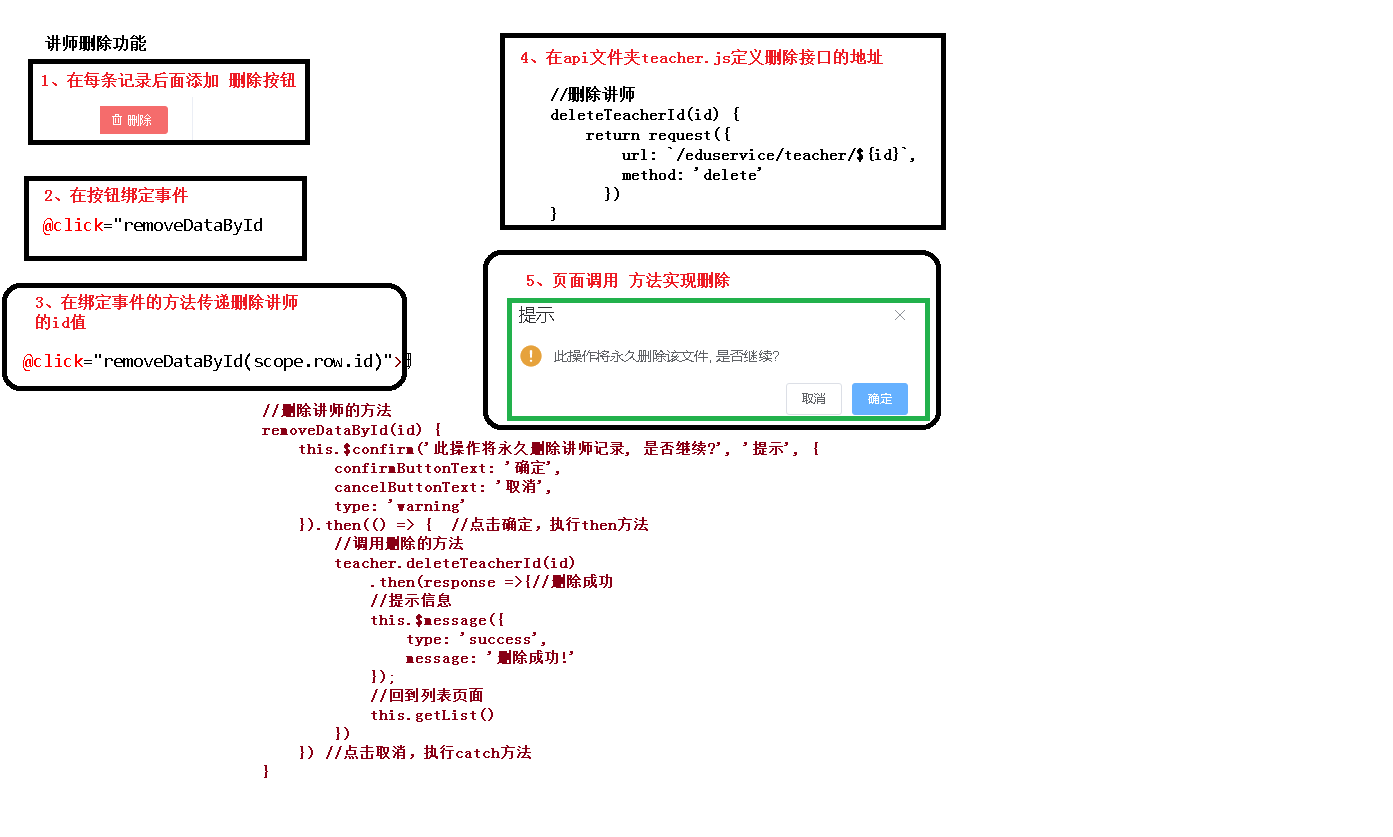
    this.searchObj = {}

    this.fetchData()

}

### 8、测试

## 删除



### 1、定义api

src/api/edu/teacher.js

    removeById(*teacherId*) {

        return request({

            url: `${api\_name}/${*teacherId*}`,

            method: 'delete'

        })

    }

### 2、定义methods

src/views/edu/teacher/list.vue

使用MessageBox 弹框组件

removeDataById(id) {

// debugger

// console.log(memberId)

this.$confirm('此操作将永久删除该记录, 是否继续?', '提示', {

confirmButtonText: '确定',

cancelButtonText: '取消',

type: 'warning'

}).then(() => {

return teacher.removeById(id)

}).then(() => {

this.fetchData()

this.$message({

type: 'success',

message: '删除成功!'

})

}).catch((response) => { // 失败

if (response === 'cancel') {

this.$message({

type: 'info',

message: '已取消删除'

})

} else {

this.$message({

type: 'error',

message: '删除失败'

})

}

})

}

# 14-讲师管理表单组件

## 一、新增



### 1、定义api

 src/api/edu/teacher.js

 save(teacher) {

return request({

url: api\_name,

method: 'post',

data: teacher

})

}

### 2、初始化组件

src/views/edu/teacher/form.vue

html

 <template>

  <div class="app-container">

    <el-form label-width="120px">

      <el-form-item label="讲师名称">

        <el-input v-model="teacher.name"/>

      </el-form-item>

      <el-form-item label="讲师排序">

        <el-input-number v-model="teacher.sort" controls-position="right" min="0"/>

      </el-form-item>

      <el-form-item label="讲师头衔">

        <el-select v-model="teacher.level" clearable placeholder="请选择">

          <!--

            数据类型一定要和取出的json中的一致，否则没法回填

            因此，这里value使用动态绑定的值，保证其数据类型是number

          -->

          <el-option :value="1" label="高级讲师"/>

          <el-option :value="2" label="首席讲师"/>

        </el-select>

      </el-form-item>

      <el-form-item label="讲师资历">

        <el-input v-model="teacher.career"/>

      </el-form-item>

      <el-form-item label="讲师简介">

        <el-input v-model="teacher.intro" :rows="10" type="textarea"/>

      </el-form-item>

      <!-- 讲师头像：TODO -->

      <el-form-item>

        <el-button :disabled="saveBtnDisabled" type="primary" @click="saveOrUpdate">保存</el-button>

      </el-form-item>

    </el-form>

  </div>

</template>

<!-- js-->

<script>

export default {

  data() {

    return {

      teacher: {

        name: '',

        sort: 0,

        level: 1,

        career: '',

        intro: '',

        avatar: ''

      },

      saveBtnDisabled: false // 保存按钮是否禁用,

    }

  },

  methods: {

    saveOrUpdate() {

      this.saveBtnDisabled = true

      this.saveData()

    },

    // 保存

    saveData() {

    }

  }

}

</script>

### 3、实现新增功能

引入teacher api模块

import teacher from '@/api/edu/teacher'

完善save方法

// 保存

saveData() {

teacher.save(this.teacher).then(response => {

return this.$message({

type: 'success',

message: '保存成功!'

})

}).then(resposne => {

this.$router.push({ path: '/edu/teacher' })

}).catch((response) => {

// console.log(response)

this.$message({

type: 'error',

message: '保存失败'

})

})

}

## 二、回显

### 1、定义api

 src/api/edu/teacher.js

getById(id) {

return request({

url: `${api\_name}/${id}`,

method: 'get'

})

}

### 2、组件中调用api

methods中定义fetchDataById

// 根据id查询记录

fetchDataById(id) {

teacher.getById(id).then(response => {

this.teacher = response.data.item

}).catch((response) => {

this.$message({

type: 'error',

message: '获取数据失败'

})

})

}

### 3、页面渲染前调用fetchDataById

created() {

console.log('created')

if (this.$route.params && this.$route.params.id) {

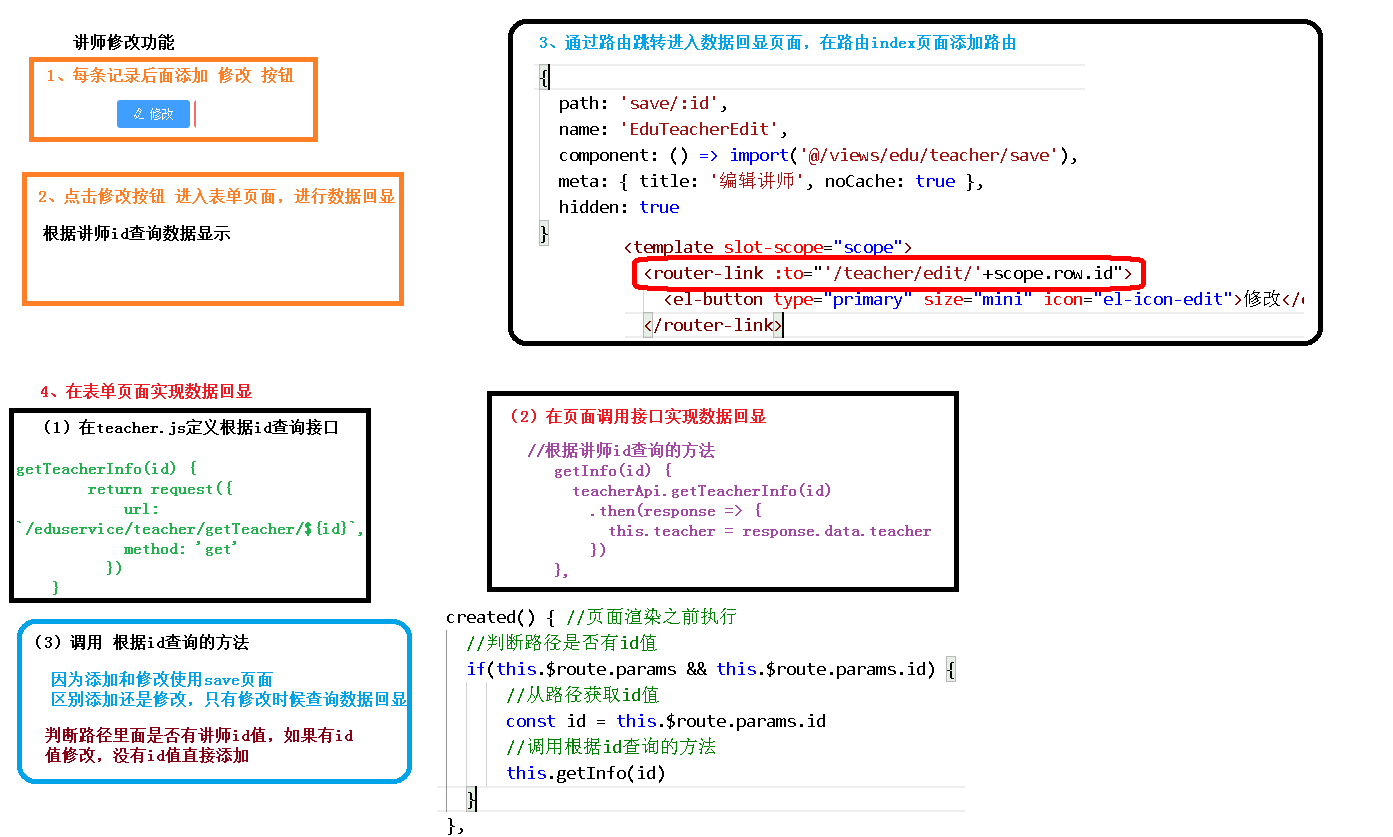
const id = this.$route.params.id

this.fetchDataById(id)

}

}

## 更新





### 1、定义api

updateById(teacher) {

return request({

url: `${api\_name}/${teacher.id}`,

method: 'put',

data: teacher

})

}

### 2、组件中调用api

methods中定义updateData

// 根据id更新记录

updateData() {

this.saveBtnDisabled = true

teacher.updateById(this.teacher).then(response => {

return this.$message({

type: 'success',

message: '修改成功!'

})

}).then(resposne => {

this.$router.push({ path: '/edu/teacher' })

}).catch((response) => {

// console.log(response)

this.$message({

type: 'error',

message: '保存失败'

})

})

}

### 3、完善saveOrUpdate方法

saveOrUpdate() {

this.saveBtnDisabled = true

if (!this.teacher.id) {

this.saveData()

} else {

this.updateData()

}

}

### 四、存在问题

vue-router导航切换 时，如果两个路由都渲染同个组件，组件会重（chong）用,

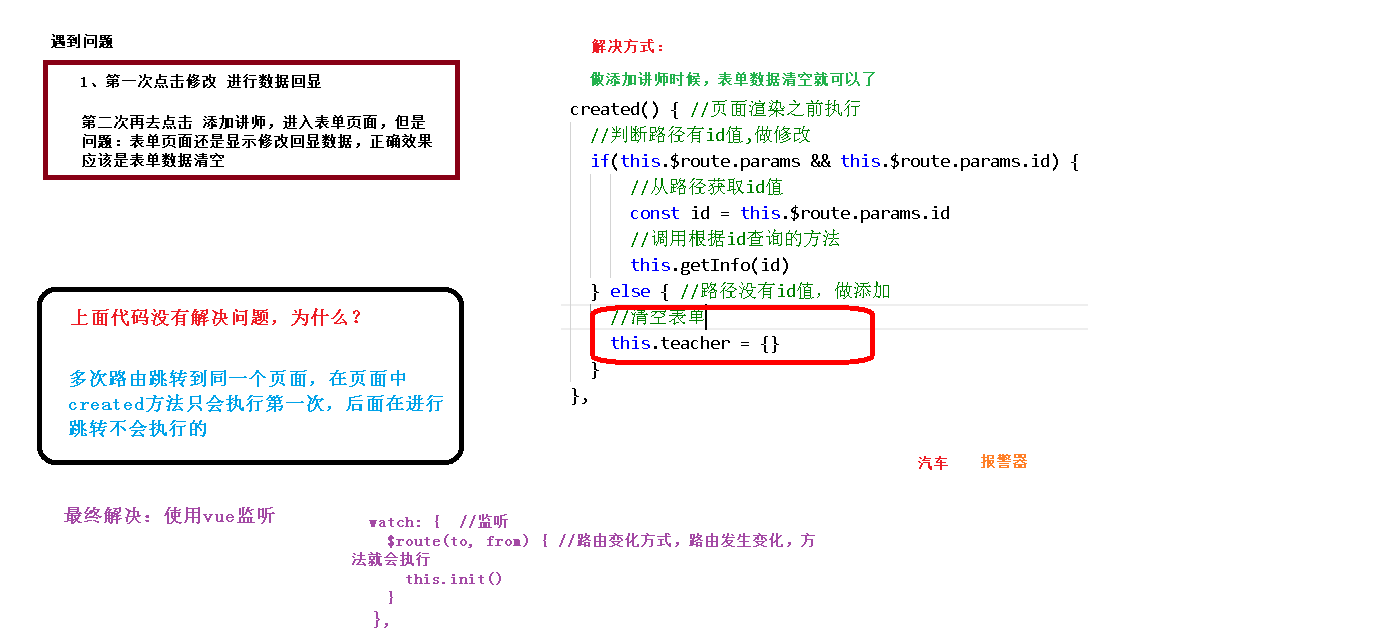
组件的生命周期钩子（created）不会再被调用, 使得组件的一些数据无法根据 path的改变得到更新

因此：

1、我们可以在watch中监听路由的变化，当路由变化时，重新调用created中的内容

2、在init方法中我们判断路由的变化，如果是修改路由，则从api获取表单数据，

      如果是新增路由，则重新初始化表单数据



<script>

import teacher from '@/api/edu/teacher'

const defaultForm = {

name: '',

sort: 0,

level: '',

career: '',

intro: '',

avatar: ''

}

export default {

data() {

return {

teacher: defaultForm,

saveBtnDisabled: false // 保存按钮是否禁用,

}

},

watch: {

$route(to, from) {

console.log('watch $route')

this.init()

}

},

created() {

console.log('created')

this.init()

},

methods: {

init() {

if (this.$route.params && this.$route.params.id) {

const id = this.$route.params.id

this.fetchDataById(id)

} else {

// 使用对象拓展运算符，拷贝对象，而不是引用，

// 否则新增一条记录后，defaultForm就变成了之前新增的teacher的值

this.teacher = { ...defaultForm }

}

},

......

}

}

</script>

# 15-后端业务的灵活优化

## 一、返回操作是否成功

### 1、删除业务逻辑

TeacherServiceImpl

@Override

public boolean removeById(Serializable id) {

Integer result = baseMapper.deleteById(id);

return null != result && result > 0;

}

TeacherAdminController

@ApiOperation(value = "根据ID删除讲师")

@DeleteMapping("{id}")

public R removeById(

@ApiParam(name = "id", value = "讲师ID", required = true)

@PathVariable String id){

boolean result = teacherService.removeById(id);

if(result){

return R.ok();

}else{

return R.error().message("删除失败");

}

}

### 2、前端

在catch中处理错误信息