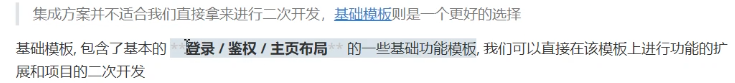
**人力资源项目总结**

**vue-element-admin**

通用的vue后台集成方案

B端，面向管理员





**搭建项目的基本准备**

本项目技术栈基于ES2015+、vue、vuex、vue-router、vue-cli、axios和element-ui

第一步：确认node环境，要依赖npm功能来管理依赖包，查看node和npm 的版本

node -v

npm -v

第二步：查看git安装版本  git --version

第三步：查看npm淘宝镜像是否安装成功

//查看镜像地址

npm config get registry

//设置淘宝镜像地址

npm config set registry https://registry.npm.taobao.org/

第四步：vscode编辑器的插件是否已装好eslint 和vetur

vetur 是基于单文件组件开发的支持插件

eslint 是基于代码校验的插件工具

第五步、git 拉取基础项目模板到hrsaas目录

git clone <https://github.com/PanJiaChen/vue-admin-template.git> hrsaas

把.git文件删除，我们要自己建立仓库 git init  ->npm i  ->npm run dev

各文件含义

**项目运行机制和代码注释**

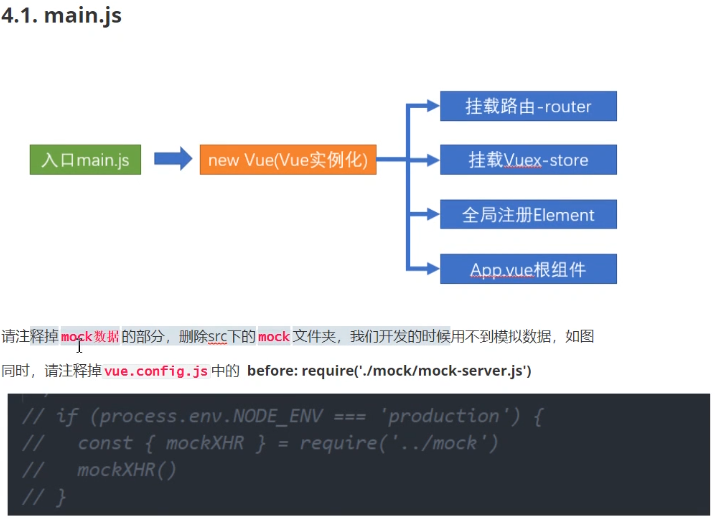




js文件，都是import导入，export default默认导出或者export 选择导出

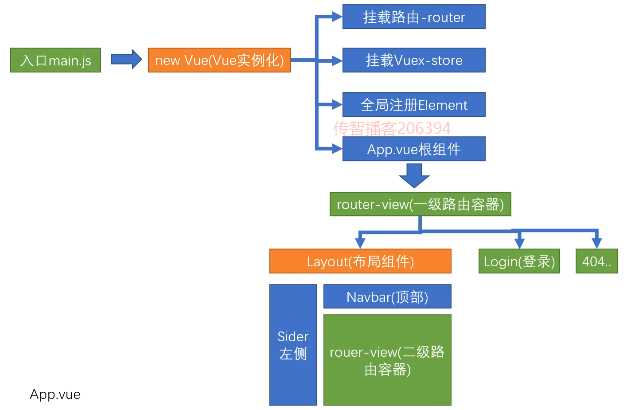
vue文件，都是template，script,style三个模块，其中script也是import导入，export default默认导出，与js文件不一样多了很多组件用的data,watch,methods等

**main.js**



**App.vue**

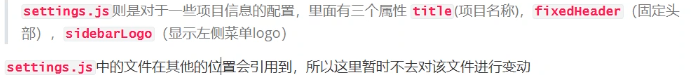
App.vue:根组件文件



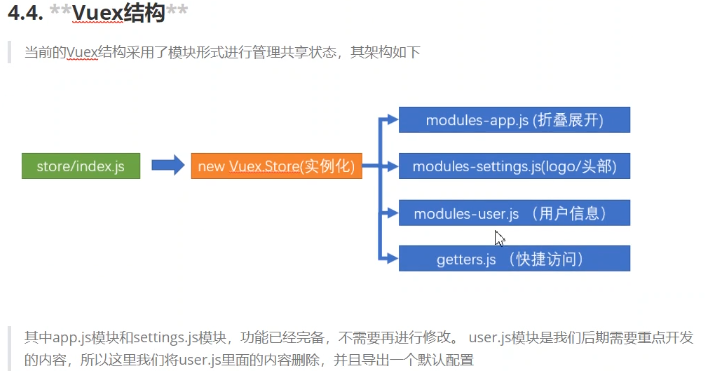
**promission**

是控制页面登录权限的文件，专门做页面的权限拦截，

**setting**



**store文件夹下的配置**

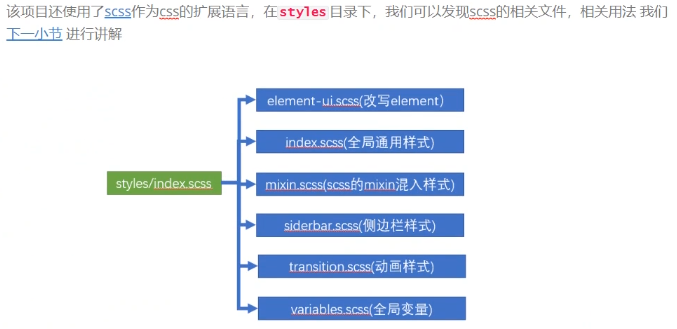


一、store文件夹是vue的vuex文件，其中，index是对外的主文件，要被引入到main.js中的

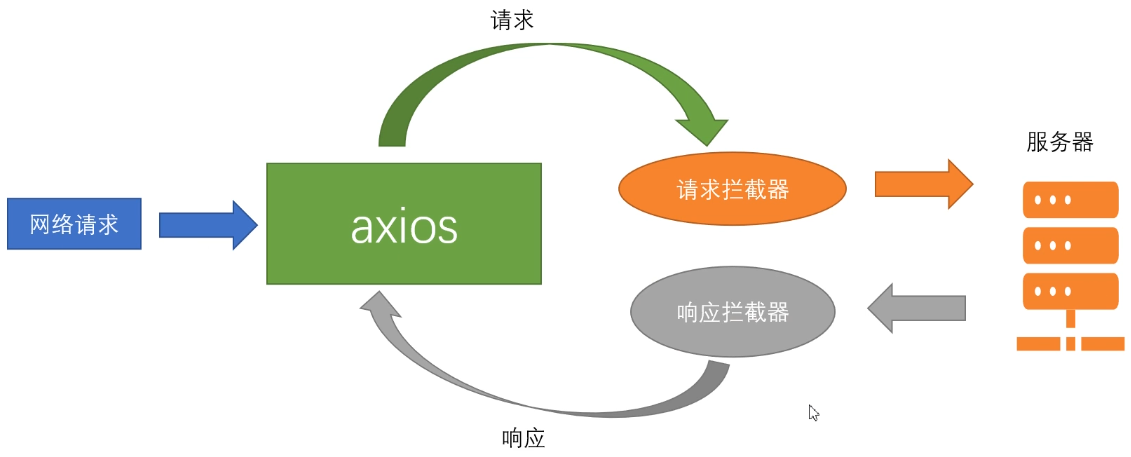
二、index文件夹内必须引入vue和vuex,再引入store文件夹下modules文件夹下的所有子模块文件，和处理快捷访问的getters文件，将他们都挂载到modules属性上

三、在store文件夹下的getters.js中设置快捷访问

**scss**



**API模块和请求封装模块的介绍**



utils工具文件夹下的request.js文件内

// create an axios instance，通过creat创建了一个新的axios实例

const service = axios.create({

baseURL: process.env.VUE\_APP\_BASE\_API, // url = base url + request url

// withCredentials: true, // send cookies when cross-domain requests

// 超时时间

timeout: 5000 // request timeout

})

为了后续更清楚的写代码，我们将原有的代码注释掉，换成如下的代码

//导入axios

import axios from 'axios'

const service = axios.create()

// request interceptor 请求拦截器

service.interceptors.request.use()

// response interceptor 响应拦截器

service.interceptors.response.use()

export default service

**API模块的单独封装**

api文件是放所有的请求

每一个接口都单独封装了一个方法，这样做的好处是，任何位置需要请求的话，可以直接引用我们导出来的请求方法

为了后续更好的开发，我们可以先将user.js的内容清空，后续再进行更正。<https://github.com/slimbaby/hr.git>

**图片和样式文件的放置**

图片文件夹放到src->assets文件夹内

全局样式文件放到

src->style文件夹中

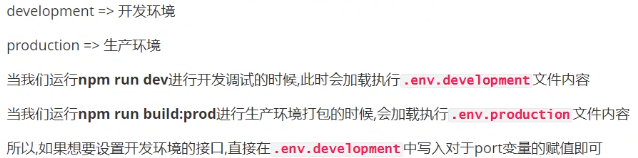
在index.scss文件中导入全局样式

// 引入公共样式

@import './common.scss';

**设置固定的本地访问端口和网站名称**

vue.config.js就是vue项目相关的编译，配置，打包，启动服务相关的配置文件，相当于改良版的webpack



在开发环境文件中增加如下代码

#定义服务启动的端口号

port = 8888

修改项目名称：在setting.js文件夹中直接修改title

**做login登录页面**

一、修改背景图片：注意，如果要在样式表style中使用@别名的时候，需要在@前面加上一个~符号

二、表单验证的错误信息样式修改,注意有两个下划线

.el-form-item\_\_error {

color: #fff;

}

**表单校验：**

trigger校验的触发方式，blur/change

**回车按键修饰符**

@keyup.enter属于按键修饰符，如果我们想监听在按回车键的时候触发，可以如下编写

@keyup.enter.native="handleLogin"

@keyup.enter.native表示监听组件的原生事件，比如keyup就是input的原生事件，就是智能操作原生DOM元素，由el-ui封装的组件标签不能直接监听这种事件，要加一个native。。。。el-button除外

**跨域问题**

生产环境的跨域问题

预生产环境的跨域问题

开发环境的跨域问题...

测试环境的跨域问题...

目前解决：开发环境的跨域问题：也就是在vue-cli脚手架环境下开发启动服务时，我们访问接口所遇到的跨域问题。vue-cli为我们在本地开起来一个服务，可以通过这个服务帮我们代理请求，解决跨域问题

这就是**vue-cli配置webpack的反向代理**

只有前端访问另一个域的后端资源有跨域问题，后端访问后端没有跨域问题



Vue-cli的配置文件即vue.config.js这里有我们需要的代理选项

devServer:{}







// 登录接口单独封装

export function login(data) {

// 返回一个promise对象

return request({

url:'/sys/login',

method:'post',

data

})

}

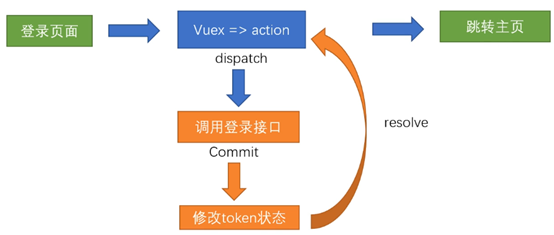
我的理解是在api接口文件中把所有的接口进行一个封装，便于后面接口的重复调用

**配置token**

传统登录页面的登录示意图



Vuex的用户登录示意图



实现store/modeules/user.js的基本设置

/ 从头开始搭建

// 降低耦合性，在外部定义三个变量

// 放置状态

const state = {}

const mutations = {}

const actions = {}

export default {

namespaced: true,

state,

mutations,

actions

}

**注意**：因为我们调用的是vuex中子模块的action，该模块我们进行了namespaced: true,  所以我们引用action时需要带上user/

this['user/login'](this.loginForm)

要用这种方法调用，不然会报错

在state中设置token的共享状态

token: null // 设置token为共享状态

我们发现在utils文件夹下的auth.js文件给我们配置了设置，获取，移除token的方法

设置  const TokenKey = 'hrsaas-ihrm-token'  //设置一个独一无二的key

在store/modeules/user.js中引入这三个token方法：import { getToken,setToken,removeToken } from '@/utils/auth'

// 降低耦合性，在外部定义三个变量

const state = {

token: getToken() // 设置token为共享状态，一开始缓存有就是有，没有就是空

}

const mutations = {

// 缓存token改变的时候需要同步到vuex里，将数据设置给vuex

setToken(state, token) {

state.token = token

// 当token值变化的时候，同步给缓存存起来

setToken(token)

},

// 移除token同步到缓存

removeToken(state) {

state.token = null

removeToken()

}

}

**封装登录的action**

登录action要做的事情，调用登录接口，成功后设置token到vuex,失败则返回失败

// 在这里调用接口

const actions = {

async login(context, data) {

// 调用login的api的接口

const { data: result } = await login(data)

if (result.success) {

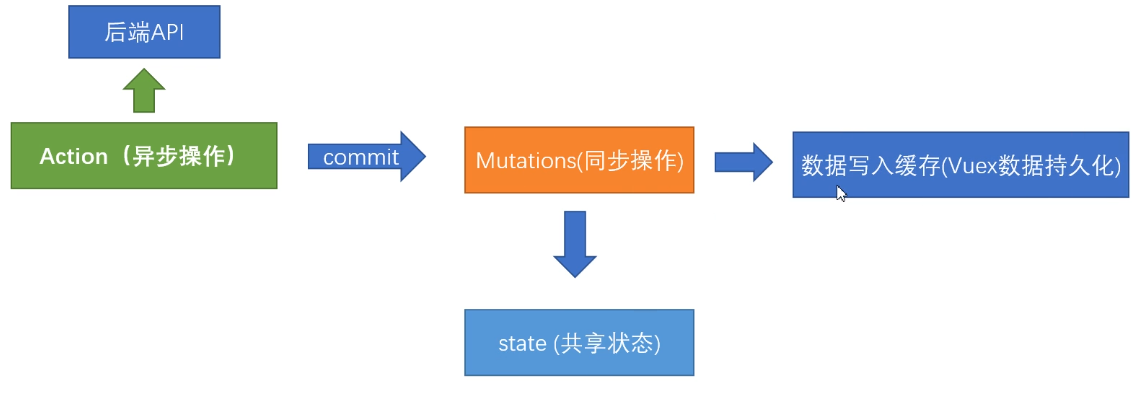
context.commit(setToken, 'result.data ')

}

}

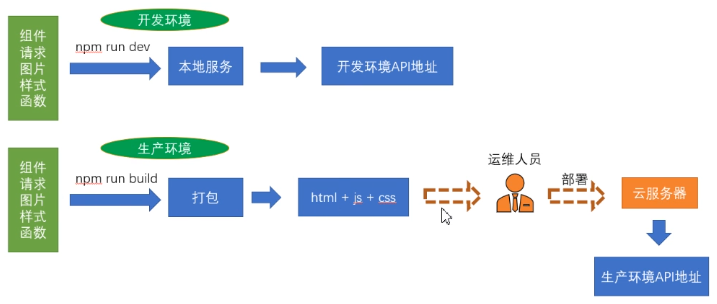
}

完成token数据持久化图像总结



**设置request中环境变量和异常的处理**

**区分axios在不同环境中的请求基础地址**





**axios设置根目录路径**

在request文件中,设置基础地址，并设置响应时间

const service = axios.creat({ baseURL: process.env.VUE\_APP\_BASE\_API, time:5000})

**处理axios的响应拦截器**

就是给所有响应回来的数据先进行一系列的操作，过滤掉失败的响应和结构数据

// response interceptor 响应拦截器

service.interceptors.response.use(response => {

// asiox默认加了一层data

const { success, message, data } = response.data

// 要根据success的成功与否解决下面的操作

if (success) {

return data

} else {

// 业务已经错误了，就不能进then了，要进catch

Message.error(message) // 提示错误消息

// 没有错误对象，new一个错误对象

return Promise.reject(new Error(message))

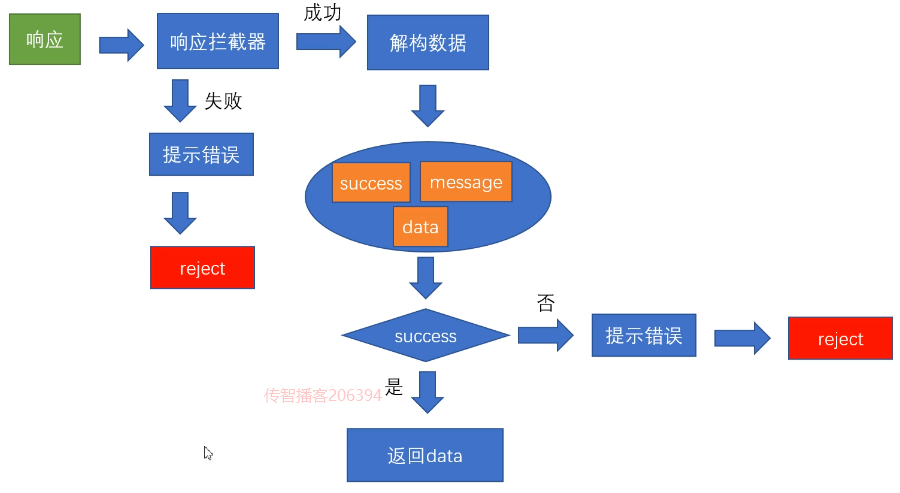
}

}, error => {

Message.error(error.message) // 提示错误信息

return Promise.reject(error) // 返回执行错误，让当前的执行链跳出成功，直接进入.catch

})



**ref**

ref可以获取一个元素的dom对象，当ref作用到组件上的时候，可以获取该组件的实例 this

组件快捷方式传入的

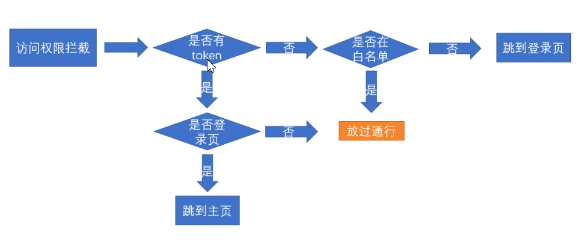
...mapActions(['user/login']),

this['user/login'](this.loginForm)

要用这种方法调用，不然会报错

**权限拦截**

前置守卫，后置守卫，进度条，白名单



// 权限拦截在这里拦截

import router from '@/router'

// 引入store实例

import store from '@/store'

// 引入进度条

import nprogress from 'nprogress'

import 'nprogress/nprogress.css'

// 定义一个白名单

const whiteList = ['/login', '/404']

// 前置守卫

router.beforeEach((to, from, next) => {

// 开启进度条

nprogress.start()

if (store.getters.token) {

if (to.path === '/login') {

// 如果访问的是登录页，跳转到登录页

next('/')

} else {

next()

}

} else {

if (whiteList.indexOf(to.path) > -1) {

next()

} else {

next('/login')

}

}

nprogress.done()// 解决手动切换地址时，进度条不关闭的事件

})

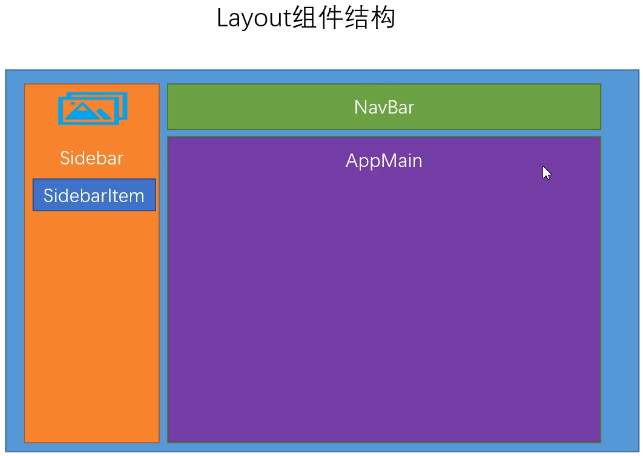
// 后置守卫

router.afterEach(() => {

nprogress.done()

})

**导航结构**



**统一注入token src/utils/request.js,为有权限的请求注入token**

service.interceptors.request.use(config => {

// 在这个位置需要统一的去注入token

if (store.getters.token) {

// 如果token存在 注入token

config.headers['Authorization'] = `Bearer ${store.getters.token}`

}

return config // 必须返回配置!!!!!!!

}, error => {

return Promise.reject(error)

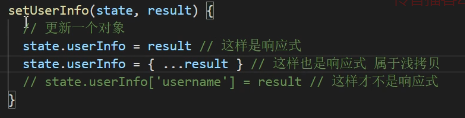
})

**以用户数据为例存入vuex中**

1、在用户api文件夹中封装请求用户信息的请求

2、在store文件夹下的用户组件中封装action的异步处理函数（先导入封装的请求）

3、通过控制mutation中的同步方法，在state中新增userInfo属性，先设定为空对象（为了确保响应式的数据）



**通过自定义指令的形式解决异常图片的处理**

注册自定义指令

Vue.directive('指令名称', {

// 会在当前指令作用的dom元素 插入之后执行！！！

// options 里面是指令的表达式

inserted: function (dom,options) {

}

})

自定义指令可以采用统一的文件来管理  **src/directives/index.js**,这个文件负责管理所有的自定义指令

首先定义第一个自定义指令  **v-imagerror**

export const imagerror = {

// 指令对象 会在当前的dom元素插入到节点之后执行

inserted(dom, options) {

// options是 指令中的变量的解释 其中有一个属性叫做 value

// dom 表示当前指令作用的dom对象

// dom认为此时就是图片

// 当图片有地址 但是地址没有加载成功的时候 会报错 会触发图片的一个事件 => onerror

dom.onerror = function() {

// 当图片出现异常的时候 会将指令配置的默认图片设置为该图片的内容

// dom可以注册error事件

dom.src = options.value // 这里不能写死

}

}

}

**在main.js完成自定义指令全局注册（循环注册）**

import \* as directives from '@/directives'

// 注册自定义指令

// 遍历所有的导出的指令对象 完成自定义全局注册

Object.keys(directives).forEach(key => {

// 注册自定义指令

Vue.directive(key, directives[key])

})

针对上面的引入语法 \*\*`import \* as 变量`\*\* 得到的是一个对象\*\*`{ 变量1：对象1，变量2： 对象2 ... }`\*\*, 所以可以采用对象遍历的方法进行处理

指令注册成功，可以在\*\*`navbar.vue`\*\*中直接使用了

注册完就可以直接使用

<img v-imageerror="defaultImg" :src="staffPhoto" class="user-avatar">

data() {

return {

defaultImg: require('@/assets/common/head.jpg')

//为什么不直接写路径，因为打包之后所有的静态资源文件会被放到固定的一个地方，为了不让路径发生问题，所以我们需要用require包裹一下，把它转换成一个变量

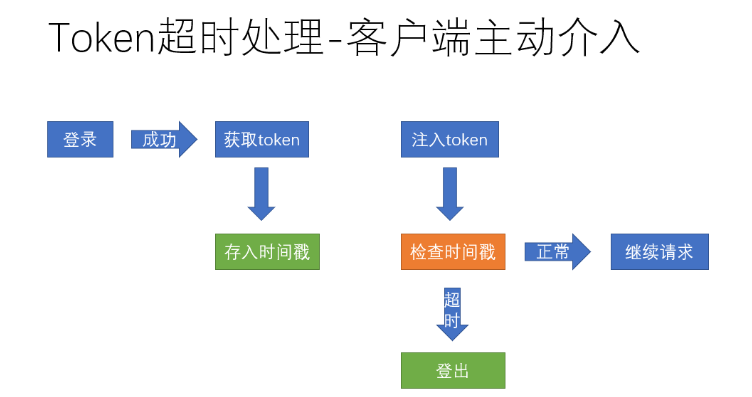
}

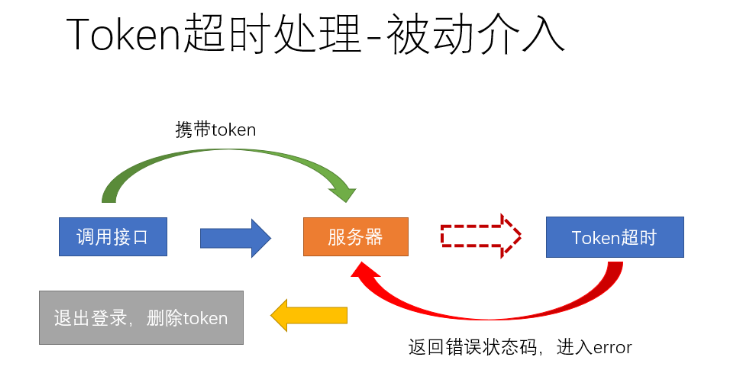
},

**登出功能**

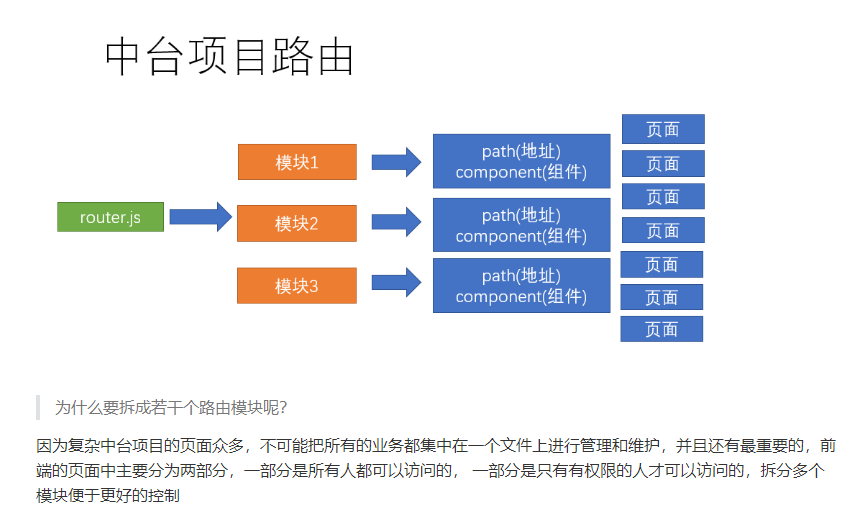


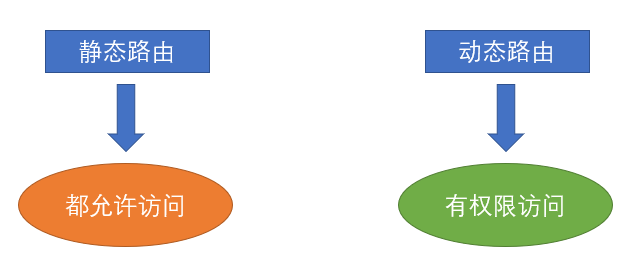
**token失效的主动介入和被动介入**





**路由页面整理**





**组织架构模块**

封装树形结构组件，并配置接口调用

**组件封装复用，但是又有个性定制**

组件复用的时候，如果大体相同，可以封装相同组件，但是某些模块稍微有些许区别，比如下拉菜单的数量不同，需要新定义

**封装弹窗组件**

！！在弹窗组件中配置显示与隐藏的变量，传给父组件，树形组件中改变父组件的值，就这么一个小小的功能要涉及三个文件。

下拉菜单分别有不同的点击事件，可以给下拉框添加

<el-dropdown @command="operateDepts">

<el-dropdown-item command="add">添加子部门</el-dropdown-item>

<el-dropdown-item command="edit">修改子部门</el-dropdown-item>

<el-dropdown-item command="del">删除子部门</el-dropdown-item>

**完成新增部门的规则校验**

自定义校验规则；{ validator: checkNameRepeat, trigger: 'blur' }

const checkNameRepeat=(rule,value,callback)=>{

value:该表单输入的值

callback:回调函数

}

**确认按钮按下时进行表单校验**

this.$refs.deptForm.validate(isOK => {

if (isOK) {// 表示可以提交了}})

**利用sync修饰符关闭新增弹层**

子向父传值前加一个update：,并把需要更新的参数带上

this.$emit('update:showDialog', false)

父在接收的时候.sync

<add-dept :show-dialog.sync="showDialog" />

**关闭对话框弹层**

第一，对话框右上角关闭按钮添加关闭事件，事件中定义关闭对话框并清空表单数据。@close="closeDia"

第二，取消按钮绑定和关闭按钮同样的点击事件

第三，因为@close关闭事件为整个对话框关闭时触发，确认按钮不需要额外绑定取消和清空事件。

父组件调用子组件的方法