目录

[一． 安装配置 2](#_Toc47449142)

[1. 安装 2](#_Toc47449143)

[2. 创建详解 2](#_Toc47449144)

[二． 基础操作 4](#_Toc47449145)

[1、 变量和事件绑定 4](#_Toc47449146)

[2、 if和for渲染 4](#_Toc47449147)

[3、 v-show，style，class 4](#_Toc47449148)

[(1).v-show与v-hide 4](#_Toc47449149)

[(2). style绑定 4](#_Toc47449150)

[(3) .class绑定 5](#_Toc47449151)

[4、 格式化数据{{数据 | 方法名}} 5](#_Toc47449152)

[5、 watch监听以及触发的运算 5](#_Toc47449153)

[6、 async/await场景 5](#_Toc47449154)

[7、 插入样式 6](#_Toc47449155)

[8、 参数的默认值 6](#_Toc47449156)

[9、 注册全局变量 6](#_Toc47449157)

[三． vue-router 6](#_Toc47449158)

[1. Vue-router导航页的路由 6](#_Toc47449159)

[2、 创建组件并向子组件传参 7](#_Toc47449160)

[3、 子组件向父组件传值 7](#_Toc47449161)

[4、 新窗口打开 8](#_Toc47449162)

[四． vuex （全局变量和全局方法） 8](#_Toc47449163)

[五． 综合实例 9](#_Toc47449164)

[1.sass的使用 9](#_Toc47449165)

[2. swiper 9](#_Toc47449166)

[3.高德地图 10](#_Toc47449167)

[(1)引入方法 10](#_Toc47449168)

[(2) 页面里边引入 10](#_Toc47449169)

[(3)使用 11](#_Toc47449170)

[4.生成二维码 11](#_Toc47449171)

[5. Axios 11](#_Toc47449172)

[安装 11](#_Toc47449173)

[main.js 11](#_Toc47449174)

[GET方法 11](#_Toc47449175)

[POST方法 12](#_Toc47449176)

[6.调取本地json 12](#_Toc47449177)

[7.使用例子效果控件vue-particles 13](#_Toc47449178)

# 安装配置

## 安装

安装node

安装脚手架

npm install -g @vue/cli

vue –version

安装之后可能会 vue 无法识别

这需要配置下环境变量

C:\Users\yarg\AppData\Roaming\npm

vue : 无法加载文件 C:\Users\xxx\AppData\Roaming\npm\vue.ps1，因为在此系统上禁止运行脚本

使用管理员权限打开Windows power shell 执行命令：set-ExecutionPolicy RemoteSigned

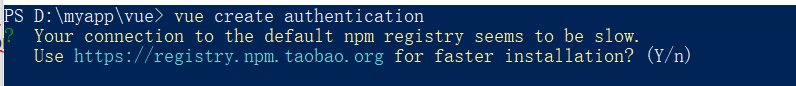
Y

创建项目

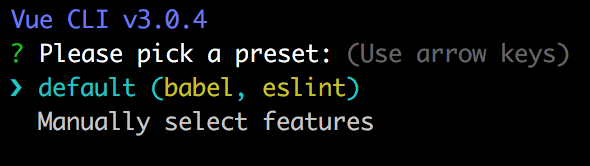
vue create [项目名]

## 创建详解

1. 有可能先询问是否使用淘宝的cnpm



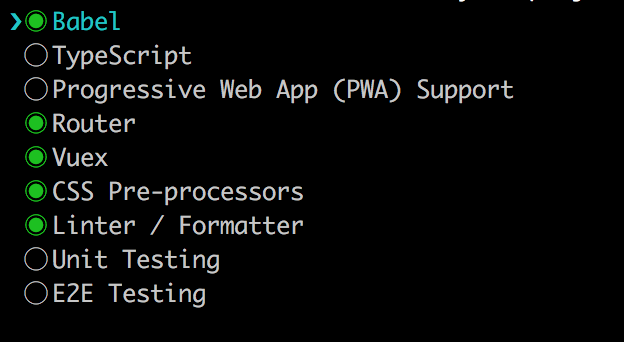
1. 选择模块



如果选择默认选项 default，将会构建一个最基本的 vue 项目（没有 vue-router 也没有 vuex）

这里推荐选择第二项 Manually select features 根据自己的需求，使用空格键选择具体的模块

2.第一项一般不选，选了第二项后，选择模块，不懂按下图的示意



（如果选了TypeScript）Use class-style component syntax?

使用样式类 组件语法

( ) Unit Testing // 单元测试（unit tests）

( ) E2E Testing // e2e（end to end） 测试

3、选路由

IMG_256

选择路由模式yes是 history模式，no是hash模式，实际项目通常采用history模式

4、选样式

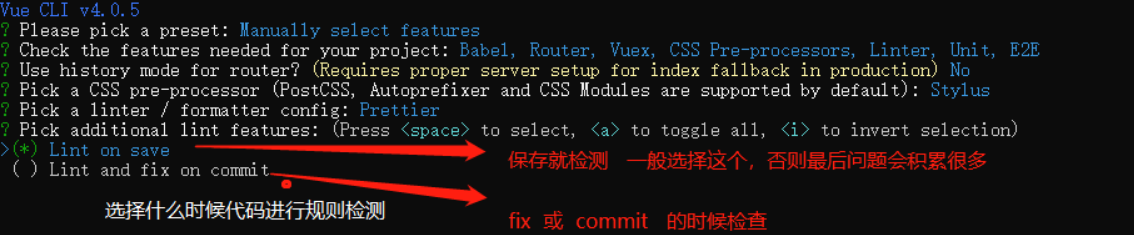
都不熟悉的话就选 sass 或者 less，因为 stylus 不支持原生 CSS 写法

下文有sass的公用样式的示例，这里的sass 需要是with node-sass

5、选择 ESLint 的校验规则，格式化的时机，和各个插件的配置项的位置，不熟悉的话就选第一个

建议使用ESLint + standard config

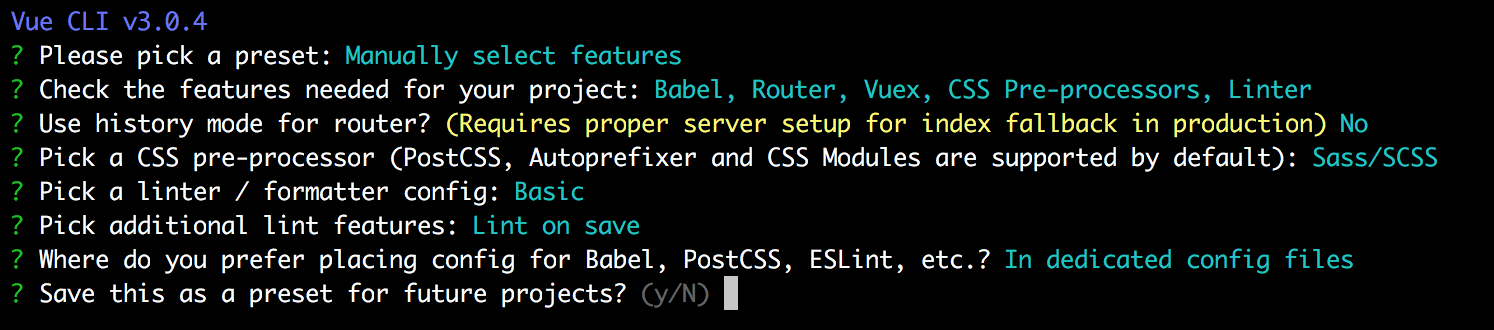
6、选择什么时候进行检测



7.选择如何存放配置



6、是否将配置项保存为预设，然后配置完成，开始生成项目



运行

npm run serve

npx vue-cli-service serve

vue serve --open

有时候会要求先安装

npm install -g @vue/cli-service-global

打包

npx vue-cli-service build

# 基础操作

## 变量和事件绑定

(1).{{变量名}}

(2).{如果在标签上,直接写变量，不需要{{}}，并且这个标签前要加: 或者v-bind:

(3).{如何需要加上其它字符，可以+’其它字符’

例如：<div :id="url+'321'"></div>

(4).如果需要输出带html标签的内容

<div v-html="变量名"></div>

(5).点击事件

v-on:click=”方法名”

@click=”方法名”

methods:{

方法名(){

}

}

## if和for渲染

(1).if渲染

v-if

v-else-if

v-else

上边三个是平级div中的，如果是多层if，写在div中

1. .for渲染

v-for="(item,index) in 变量名" :key="index"

## v-show，style，class

### (1).v-show与v-hide

这两个为vue写法，也是判断里边的内容是否为真

### (2). style绑定

不带变量style正常

带的化写成对象形式 :style="{'color':'#ccc','样式名':'样式值'}"

### .class绑定

不带变量class正常

带的化写成对象形式，并且样式值为bool :class="{'样式名':true}"

## 格式化数据{{数据 | 方法名}}

filters: {

方法名(time) {

var a = time.$date.substr(0, 10);

return a;

}

}

## watch监听以及触发的运算

Watch 与computed都是data型的一级对象

watch:{

name:function (newval,oldval){

console.log("检测到变化，新值是："+newval+"，老值是："+oldval)

}

},

Computed:{

msg1:function(){

return ‘我发生了改变’+this.msg }｝

这种写法msg一发生变化msg1也跟着变化了

变量发生变化触发的运算，也会导致关联的元素重新渲染，监听的是一个变量

## async/await场景

这是一个用同步的思维来解决异步问题的方案，当前端接口调用需要等到接口返回值以后渲染页面时。

    async的用法，它作为一个关键字放到函数前面，用于表示函数是一个异步函数，因为async就是异步的意思， 异步函数也就意味着该函数的执行不会阻塞后面代码的执行，async 函数返回的是一个promise 对象。

   await的含义为等待。意思就是代码需要等待await后面的函数运行完并且有了返回结果之后，才继续执行下面的代码。这正是同步的效果

async listAreaAPIFn (id) {  
 await listAreaAPI(id).then(res => {

})  
}

## 插入样式

let aMapScript = document.createElement('myAMap')

aMapScript.setAttribute('src', 'https://webapi.amap.com/maps?v=1.4.14&key=f93f00a27151d556a1140c574910e5eb')

document.head.appendChild(aMapScript)

## 参数的默认值

方法（参数=’默认值’）｛｝

## 注册全局变量

import 类库名 from '类库名'  
Vue.prototype.$类库名 = 类库名

后边使用 this.$类库名

const [name, age] = res

# vue-router

## Vue-router导航页的路由

新建一个组件，以.vue为后缀

<template>

<div>我是一个新组件</div>

</template>

<script>

export default {

name: '组件名',

components: {}

}

</script>

<style scoped lang="scss"> //scoped 是只有这个页面读取到这些样式 scss是sass

</style>

如果后边想作为一个导航栏上的页面需加入router.js

,{

path: '/组件名',

name: '组件名',

component: () => import('./views/组件名.vue')

}

使用

<router-link to="/info">info</router-link>

前提是有<router-view/>或<router-view> </router-view>

## 创建组件并向子组件传参

子组件定义变量

props: {

msg: String

}

父组件传参

<template><div>

<HelloWorld msg="Welcome to Your Vue.js App"/> //调取组件并传参

//直接传变量的方法 :msg=变量

</div></template>

<script>

// @ is an alias to /src

import HelloWorld from '@/components/HelloWorld.vue' //引入组件

export default {

components: {

HelloWorld //定义组件

}

}

</script>

## 子组件向父组件传值

子组件

在一个方法内直接写

this.$emit("事件名","参数值")

this.$emit("ListenToChildEvent","这是啥啊")

父组件

在调用这个组件的时候，添加@+事件名=”本地方法”

这个本地方法获取到的返回值就是传的参数

<HelloWorld msg="Welcome to Your Vue.js App" @ListenToChildEvent="bb"/>

bb:function(data){

console.log(data)

}

## 新窗口打开

let routeUrl = this.$router.resolve({  
 path: 'print',  
 name: 'print'  
})  
window.open(routeUrl.href, '\_blank')

# vuex （全局变量和全局方法）

感觉更像一个全局变量和全局方法

Store.js详解

import Vue from 'vue'

import Vuex from 'vuex'

Vue.use(Vuex)

export default new Vuex.Store({

state: { //state里边传变量

count:0

},

mutations: { //mutations里边传方法

increase(){ //不传参数的写法

this.state.count++;

},

redate(state, v){ //传参数的写法，第一个变量为它自己，过来的不管是一个还是多个参数，都传在第二个参数中

this.state.count=v.v1+v.v2;

}

},

actions: {

}

})

Vue调取

引入 import store from ['@/store.js'](mailto:'@/store.js')

加入export default

export default {

store,

}

调取变量 store.state.变量名

不带参数调取方法 store.commit("方法名")

带参数调取方法 store.commit("方法名",参数) 最多只有一个参数，这个参数传基础数据类

# 综合实例

## 1.sass的使用

vue.config.js（没有就新建）

module.exports = {

    css: {

        // 启用 CSS modules

        modules: false,

        // 是否使用css分离插件

        extract: true,

        // 开启 CSS source maps，一般不建议开启

sourceMap: false,

        // css预设器配置项

        loaderOptions: {

            sass: {

                //设置css中引用文件的路径，引入通用使用的scss文件（如包含的@mixin）

                data: `

                @import '@/assets/scss/css.scss';

                `

            }

        }

    },

｝

后边就可以在所有页面使用css.scss里边的变量及样式

vue.config.js 里边内容一改或者css.scss文件名一改，就得重新运行项目

## swiper

安装swiper

npm install swiper --save-dev

上边一个不对，是安装最新版，可最新版本有问题

npm i swiper@3.4.2 -S

引入

import Swiper from "swiper"

import 'swiper/dist/css/swiper.min.css';

mounted方法中初始化swiper

mounted(){

new Swiper('.swiper-container', {

// // 如果需要分页器

pagination: {

el: '.swiper-pagination',

},

// // 如果需要前进后退按钮

// navigation: {

// nextEl: '.swiper-button-next',

// prevEl: '.swiper-button-prev',

// },

// // 如果需要滚动条

// scrollbar: {

// el: '.swiper-scrollbar',

// },

autoplay:true,

loop : true,

autoplayDisableOnInteraction: true, // 用户操作swiper之后，是否禁止autoplay，默认为true

})

},

## 3.高德地图

vue-amap（不好用）

npm install -s vue-amap

https://elemefe.github.io/vue-amap/#/

### (1)引入方法

Index.html

<script type="text/javascript" src="https://webapi.amap.com/maps?v=1.4.14&key=f93f00a27151d556a1140c574910e5eb"></script>

vue.config.js

    configureWebpack: {

        externals: {

            'AMap': 'AMap',

        }

    },

使用页

import AMap from 'AMap'

## 页面里边引入

### (3)使用

## 4.生成二维码

添加插件

npm install vue-qr --save

import VueQr from 'vue-qr'

components:{

VueQr

}

<vue-qr text="二维码显示的内容" :size="200" :margin="0"></vue-qr>

## Axios

### 安装

npm install --save axios

main.js

import Axios from 'axios'

import qs from 'qs'

Vue.prototype.$Axios = Axios

Vue.prototype.$qs=qs

## GET方法

this.$Axios.get('http://hzz.test.lonhcloud.cn/lhHzzTtcServerlet/commonServerlet?prj=lhHzzTtcManager&bean=TtcManagerSS&method=findTasktargetbPageListByOrder&1=1&2=12&3=-1&4=13871&5=1,2,3&6=&7=-1&8=&9=').then(res => {

console.log(res.data)

}, res => {

console.log(res)

console.info('调用失败');

})

### POST方法

this.$Axios

.post(

"http://hzz.test.lonhcloud.cn/lhHzzTtcServerlet/commonServerlet",

this.$qs.stringify({

prj: "lhHzzTtcManager",

bean: "TtcManagerSS",

method: "findTasktargetbPageListByOrder",

"1": 1,

"2": 12,

"3": -1,

"4": 13871,

"5": "1,2,3",

"6": "",

"7": -1,

"8": "",

"9": ""

}),

{

headers: {

"Content-Type": "application/x-www-form-urlencoded;charset=UTF-8"

},

}

)

.then(

function(res) {

console.log("控制台打印错误返回的内容11111111");

console.log(res.data);

//控制台打印请求成功时返回的数据

//bind(this)可以不用

}.bind(this)

)

.catch(

function(err) {

if (err.response) {

console.log("控制台打印错误返回的内容");

console.log(err.response);

//控制台打印错误返回的内容

}

//bind(this)可以不用

}.bind(this)

);

## 6.调取本地json

建json文件在本地public下

.get(“xx.json”

## 7.使用例子效果控件[vue-particles](https://www.cnblogs.com/ldlx-mars/p/8012142.html)

<https://vue-particles.netlify.app/>

1.下载依赖

npm install vue-particles --save-dev

2.main.js引入

import Vue from 'vue'

import VueParticles from 'vue-particles'

Vue.use(VueParticles)

3.直接使用

<vue-particles

color="#dedede"

:particleOpacity="0.7"

:particlesNumber="80"

shapeType="circle"

:particleSize="4"

linesColor="#dedede"

:linesWidth="1"

:lineLinked="true"

:lineOpacity="0.4"

:linesDistance="150"

:moveSpeed="3"

:hoverEffect="true"

hoverMode="grab"

:clickEffect="true"

clickMode="push"

>

</vue-particles>

4.属性含义

color: String类型。默认'#dedede'。粒子颜色。

particleOpacity: Number类型。默认0.7。粒子透明度。

particlesNumber: Number类型。默认80。粒子数量。

shapeType: String类型。默认'circle'。可用的粒子外观类型有："circle","edge","triangle", "polygon","star"。

particleSize: Number类型。默认80。单个粒子大小。

linesColor: String类型。默认'#dedede'。线条颜色。

linesWidth: Number类型。默认1。线条宽度。

lineLinked: 布尔类型。默认true。连接线是否可用。

lineOpacity: Number类型。默认0.4。线条透明度。

linesDistance: Number类型。默认150。线条距离。

moveSpeed: Number类型。默认3。粒子运动速度。

hoverEffect: 布尔类型。默认true。是否有hover特效。

hoverMode: String类型。默认true。可用的hover模式有: "grab", "repulse", "bubble"。

clickEffect: 布尔类型。默认true。是否有click特效。

clickMode: String类型。默认true。可用的click模式有: "push", "remove", "repulse", "bubble"

[IMG_259](https://www.cnblogs.com/ldlx-mars/p/javascript:void(0);)

1. 如果要换背景可以直接给#particles-js这个id添加背景图片和颜色即可，这个id是默认的

## Mockjs

1. 引入资源

"lodash": "^4.17.15",  
"lodash.clonedeep": "^4.5.0",  
"lodash.get": "^4.4.2",  
"lodash.pick": "^4.4.0",  
"mockjs": "^1.1.0",  
"mockjs2": "^1.0.8",

mockjs 和 mockjs2二者选一个

Lodash 是用来筛选数据的

1. main.js中引入所有mock相关连文件

假设在src 下新建mock文件夹，并且有index.js为主线

import './mock'

1. mock/index.js

import { isIE } from '@/utils/util'  
// 判断环境不是 prod 或者 preview 是 true 时，加载 mock 服务  
if (process.env.NODE\_ENV !== 'production' || process.env.VUE\_APP\_PREVIEW === 'true') {  
 if (isIE()) {  
 console.error('[antd-pro] ERROR: `mockjs` NOT SUPPORT `IE` PLEASE DO NOT USE IN `production` ENV.')  
 }  
 // 使用同步加载依赖  
 // 防止 vuex 中的 GetInfo 早于 mock 运行，导致无法 mock 请求返回结果  
 console.log('[antd-pro] mock mounting')  
 const Mock = require('mockjs')  
 // require('./services/auth')  
 require('./services/user')  
 require('./services/manage')  
 require('./services/menu')  
 require('./services/depart')  
 require('./services/area')  
 require('./services/dic')  
 require('./services/role')  
 require('./services/menuPermission')  
 require('./services/other')  
 require('./services/tagCloud')  
 require('./services/article')  
  
 Mock.setup({  
 timeout: 800 // setter delay time  
 })  
 console.log('[antd-pro] mock mounted')  
}

1. mock/util.js

import { Result } from './entity'  
const \_ = require('lodash')  
const responseBody = Result  
  
export const builder = (data, msg = '操作成功', code = 200, headers = {}) => {  
 responseBody.result = data  
 if (msg !== undefined && msg !== null) {  
 responseBody.msg = msg  
 }  
 if (code !== undefined && code !== 0) {  
 responseBody.code = code  
 }  
 if (headers !== null && typeof headers === 'object' && Object.keys(headers).length > 0) {  
 responseBody.\_headers = headers  
 }  
 return responseBody  
}  
  
export const getQueryParameters = (options) => {  
 const url = options.url  
 const search = url.split('?')[1]  
 if (!search) {  
 return {}  
 }  
 return JSON.parse('{"' + decodeURIComponent(search)  
 .replace(/"/g, '\\"')  
 .replace(/&/g, '","')  
 .replace(/=/g, '":"') + '"}')  
}  
  
export const getBody = (options) => {  
 return options.body && JSON.parse(options.body)  
}  
  
*/\*\*  
 \** ***@title*** *获取分页数据  
 \** ***@param*** *data 数据源  
 \** ***@param*** *current 页码  
 \** ***@param*** *size 分页数  
 \** ***@return*** *{ list, current, pages, size, total }  
 \*/*export const getPageData = (data, current, size) => {  
 if (!data.length) {  
 return {  
 current: Number(current),  
 size: Number(size),  
 records: [],  
 pages: 0,  
 total: 0  
 }  
 }  
  
 return {  
 current: Number(current),  
 size: Number(size),  
 pages: Math.ceil(data.length / size),  
 total: data.length,  
 records: \_.chunk(data, size)[current - 1]  
 }  
}  
  
*/\*\*  
 \** ***@title*** *新增数据  
 \** ***@param*** *data 数据源  
 \** ***@param*** *params 新增的数据  
 \** ***@return*** *{ data }  
 \*/*export const addData = (data, params) => {  
 const \_data = [...data]  
 \_data.unshift(params)  
 return \_data  
}  
  
*/\*\*  
 \** ***@title*** *根据ID更新数据  
 \** ***@param*** *data 数据源  
 \** ***@param*** *id ID值  
 \** ***@param*** *params 需要更新的数据  
 \** ***@return*** *{ data }  
 \*/*export const updateData = (data, id, params) => {  
 if (!data.length) return  
 const \_data = [...data]  
 const index = \_data.findIndex(item => item.id === id)  
 \_data[index] = { ...\_data[index], ...params }  
 return \_data  
}  
  
*/\*\*  
 \** ***@title*** *根据ID删除数据  
 \** ***@param*** *data 数据源  
 \** ***@param*** *id ID值  
 \** ***@return*** *{ data }  
 \*/*export const deleteData = (data, id) => {  
 if (!data.length) return  
 const \_data = [...data]  
 debugger  
 const index = \_data.findIndex(item => item.id === id)  
 \_data.splice(index, 1)  
 console.log('\_data', \_data)  
 return \_data  
}

1. services/xx.js

import Mock from 'mockjs'  
// import { sortBy } from 'lodash'  
import { builder, getQueryParameters, getBody, addData, deleteData, updateData } from '../util'  
  
// function treeData (source, id, parentId, children) {  
// const cloneData = JSON.parse(JSON.stringify(source))  
// return cloneData.filter(father => {  
// const branchArr = cloneData.filter(child => father[id] === child[parentId])  
// // eslint-disable-next-line no-unused-expressions  
// branchArr.length > 0 ? father[children] = branchArr : []  
// return !father[parentId] // 如果第一层不是parentId=0，请自行修改  
// })  
// }  
  
  
let areaListData = Mock.mock([  
 {  
 id: '530400',  
 pid: null,  
 title: '玉溪市',  
 adcd: '530400',  
 correlation: [],  
 status: 1  
 },  
 {  
 id: '530401',  
 pid: '530400',  
 title: '市辖区',  
 adcd: '530401',  
 correlation: [],  
 status: 1  
 },  
 {  
 id: '530402',  
 pid: '530400',  
 title: '红塔区',  
 adcd: '530402',  
 correlation: [],  
 status: 1  
 },  
 {  
 id: '530402001',  
 pid: '530402',  
 title: '玉兴街道',  
 adcd: '530402001',  
 correlation: [],  
 status: 1  
 },  
 {  
 id: '530402002',  
 pid: '530402',  
 title: '凤凰街道',  
 adcd: '530402002',  
 correlation: [],  
 status: 1  
 },  
 {  
 id: '530403',  
 pid: '530400',  
 title: '江川区',  
 adcd: '530403',  
 correlation: [],  
 status: 1  
 },  
 {  
 id: '530422',  
 pid: '530400',  
 title: '澄江县',  
 adcd: '530422',  
 correlation: [],  
 status: 1  
 },  
 {  
 id: '530423',  
 pid: '530400',  
 title: '通海县',  
 adcd: '530423',  
 correlation: [],  
 status: 1  
 },  
 {  
 id: '530424',  
 pid: '530400',  
 title: '华宁县',  
 adcd: '530424',  
 correlation: [],  
 status: 1  
 },  
 {  
 id: '530425',  
 pid: '530400',  
 title: '易门县',  
 adcd: '530425',  
 correlation: [],  
 status: 1  
 },  
 {  
 id: '530426',  
 pid: '530400',  
 title: '峨山彝族自治县',  
 adcd: '530426',  
 correlation: [],  
 status: 1  
 },  
 {  
 id: '530427',  
 pid: '530400',  
 title: '新平彝族傣族自治县',  
 adcd: '530427',  
 correlation: [],  
 status: 1  
 },  
 {  
 id: '530428',  
 pid: '530400',  
 title: '元江哈尼族彝族傣族自治县',  
 adcd: '530428',  
 correlation: [],  
 status: 1  
 }  
])  
  
  
  
  
const areaList = (options) => {  
 console.log('options', options)  
  
 return builder(areaListData)  
}  
  
  
  
  
  
// const areaList = (options) => {  
// const queryParameters = getQueryParameters(options)  
// if (queryParameters.type === 'back') {  
// // 过滤出属于管理系统的菜单(systemType为1)  
// const \_area = sortBy(areaListData.filter((d) => d.status === 1), (d) => d.sortNo)  
// // 封装树形结构  
// const list = treeData(\_area, 'id', 'pid', 'children')  
// return builder(list)  
// } else {  
// return builder([])  
// }  
// }  
  
const addArea = (options) => {  
 const body = getBody(options)  
 let \_data = Mock.mock({  
 id: '@id'  
 })  
 \_data = { ...\_data, ...body }  
 areaListData = addData(areaListData, \_data)  
 return builder(\_data, '添加成功.')  
}  
  
const updateArea = (options) => {  
 const body = getBody(options)  
 const \_data = { ...body }  
 areaListData = updateData(areaListData, body.id, \_data)  
 return builder(\_data, '更新成功.')  
}  
  
const deleteArea = (options) => {  
 console.log('options', options)  
 const { id } = getQueryParameters(options)  
 console.log('id', id)  
  
 console.log('areaListData', areaListData)  
 areaListData = deleteData(areaListData, id)  
 return builder(null, '删除成功.')  
}  
  
Mock.mock(/\/api\/area\/list/, 'get', areaList)  
Mock.mock(/\/api\/area\/add/, 'post', addArea)  
Mock.mock(/\/api\/area\/update/, 'put', updateArea)  
Mock.mock(/\/api\/area\/delete/, 'delete', deleteArea)

1. 细节差别

mock的post，参数可以传去body里边

但get ，delete这些传不过去，而且不会换成地址栏目

所以写的时候写成这种

export const organizDelectAPI = (id) => axios.delete('/api/area/delete?id=' + id)

1. mock文档生成规则

var aa = Mock.mock({  
 'data|2': [{ // 生成N条数据  
 'id|+1': '@ctitle(3,5)' + 0, // Id 自增  
 title: '@ctitle(3,12)', // 生成随机字符串  
 status: '@integer(0,3)', // 生成随机数字  
 title2: '@ctitle(3,5) @integer(0,3)', // 字符 数字混合  
 'star|1-5': '★', // 随机循环几次  
 createTime: '@now()', // 生成当前时间  
 marketing\_start: '@datetime()', // 生成随机时间  
 city: '@county(true)', // 生成随机城市  
 img: '@image("200x200", "@color", "pic")',  
 'goodsStar|1-5': '★'  
 }]  
})