# 目录结构

主要指src目录下的文件结构：

├── components 公共vue组件

│ └── breadcrumb

│ └── index.vue

├── imgs 公共图片资源

│ └── element-logo

├── router vue路由相关

│ └── index.js

├── store vuex相关

│ └── index.js

├── styles 公共样式相关

│ └── index.sass

├── views 页面相关

│ └── 页面名称

│ ├── components 页面级组件

│ │ └── index.vue

│ ├── imgs 页面级imgs

│ │ └── xxx.jpg

│ └── 页面.vue

├── app.vue vue页面layout

└── main.js vue打包入口

## components

存放公共vue组件的目录，内部存放的一般是全局公共使用的，如面包屑，后期可根据实际情况将弹窗等抽象出来，做成公共的组件

命名规则：组件名称作为目录，目录下放index.vue存放组件代码，例如面包屑，components 中存放breadcrumb目录，下放index.vue作为组件的入口

## imgs

公共图片资源目录，存放公共图片资源

## router

vue路由相关文件，本项目使用的vue-router，控制前端的路由

何为前端路由：我理解的路由就是绑定一个一个页面或者组件的访问URL地址，当切换菜单或页面时，整个页面不会刷新不会重新加载，仅仅是切换路由来加载不同的组件显示不同的内容而已，分析src\main.hjs入口文件核心代码

import router from './router';

就是加载前端的路由文件，使用ctrl+鼠标左键点击./router即可进入路由配置文件，里面详细配置了前端使用需要的路由及关联的组件，例如：

{

path: '/index',

name: '欢迎',

component: () => import('@/views/welcome'), // ()=> 动态导入

}

即表示当URL为/index时，将显示\src\views\welcome这个组件的内容，分析路由文件代码会发现所有的路由配置，都是一个匹配的url地址，配合一个组件来完成的

再分析下方的代码：

Vue.use(Element, {

size: 'medium'

});

全局插件调用，指全局使用element-ui插件并配置其参数

Vue.prototype.axios = axios;

指使用axios插件作为ajax请求的组件

Vue.component('layout', layout);

全局使用layout，layout的代码在src\views\layout组件下

new Vue({

el: '#app',

router,

store,

template: '<App/>',

components: {

App

}

});

我们的所有页面入口的HTML模版，实际就是一个APP标签，我们进入APP代码定义处.\app.vue中，

<template>

<div id="app">

<layout>

<router-view></router-view>

</layout>

</div>

</template>

其中的<router-view>标签就是表示匹配路由的关联组件或页面加载的HTML内容，替换这个标签

整个前端框架基础实际就是一个路由的匹配过程，通过匹配不同的路由来替换<router-view>标签的内容

整体页面布局在src\views\layout组件中，使用element-ui的layout组件来进行的布局，重点讲解下<slot>标签，这里的slot是插槽的意思，可以理解为一个占位符，.\app.vue文件中<layout>标签内部的<router-view>标签，即会替换掉这里的<slot>标签，整体含义就是使用路由匹配的组件解析出来的HTML代码替换<router-view>标签（这个是vue-rouer的知识点），再使用替换后的内容替换<layout>中的<slot>标签，完成页面布局（这个是element-ui的layout的知识点）

## store

全局数据仓库，所有前端需要全局缓存的数据就在这里进行管理，如登录用户token，登录用户信息等，具体可参考vuex插件的知识点

## styles

全局样式定义，只有最最公共的样式才会定义在这里，其他页面或组件内部的样式都定义在自己的入口文件中

注意，全局样式使用scss编写，与普通css不同点就是子元素样式是嵌套书写的，例如：

.header-operations {

display: inline-block;

float: right;

padding-right: 30px;

height: 100%;

& li {

display: inline-block;

vertical-align: middle;

margin: 0 10px;

line-height: 60px;

cursor: pointer;

&:last-child {

cursor: default;

}

}

.header-download {

opacity: 0.4;

cursor: default;

&.is-available {

opacity: 1;

cursor: pointer;

}

}

span {

opacity: 0.7;

}

.header-lang {

cursor: pointer;

&.is-active {

opacity: 1;

}

}

}

## views

所有页面的代码都在这个目录中，其中layout是布局的组件

注意：内部的所有组件都应该有自己的目录，目录内部使用如下代码结构，示例代码是因为代码较简单，没有创建目录，并按如下代码规则，实际开发时，一定需要注意，使用标准的代码结构

│ └── 页面名称

│ ├── components 页面级组件

│ │ └── index.vue

│ ├── imgs 页面级imgs

│ │ └── xxx.jpg

│ └── 页面.vue

组件内部的样式，是使用css书写的，为防止css冲突，在每个组件的style中，可以增加scoped属性，这样即使style中定义的css类名一样，编辑器也会自动增加命名空间，防止样式污染，如下：

<style lang="css" scoped>

</style>

## app.vue

入口组件标签文件，主要作用就是说明使用路由匹配的组件内容替换<router-view>标签，从而替换<layout>中的<slot>标签，实现侧边栏，顶部，面包屑等统一布局，通过改变路由来改变layout中间的content的内容

## main.js

入口文件，解析参见3、router路由部分

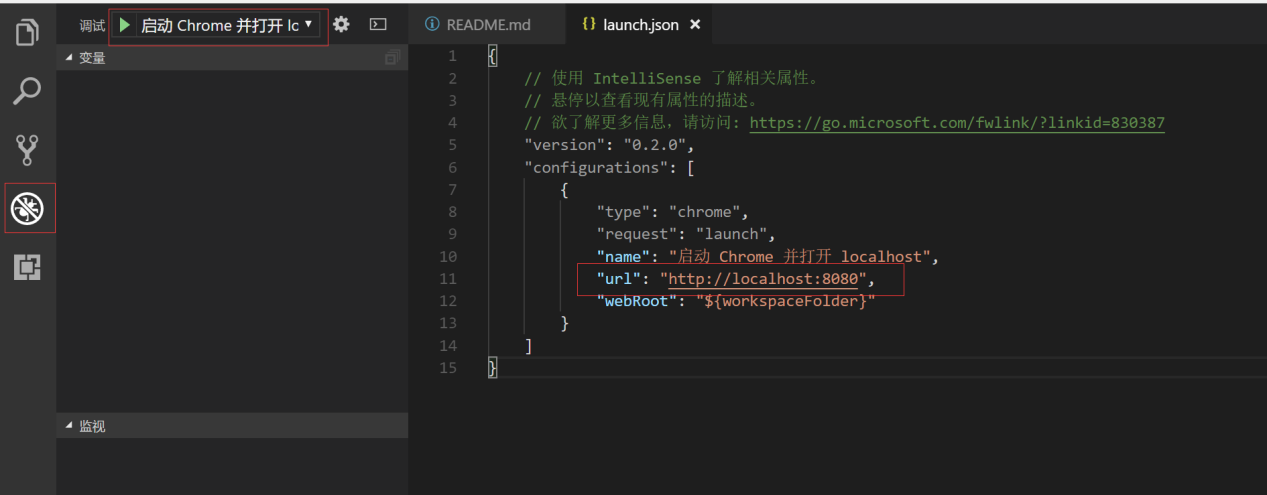
# 如何启动项目

代码编写完毕，输入命令 npm run dev，则启动项目，注意，如果代码不能通过eslint的检测，则无法编译通过

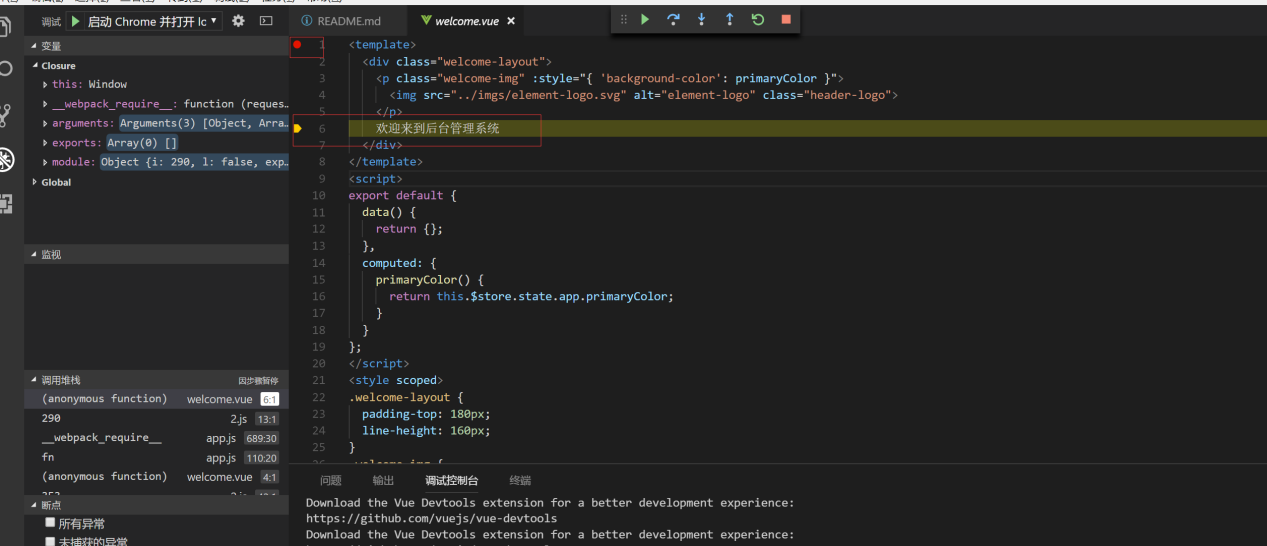
# 如何启动调试

1、IDE调试

点击vs code 调试选项，点击齿轮配置图标，选择chrome，右方配置修改当前项目的端口号

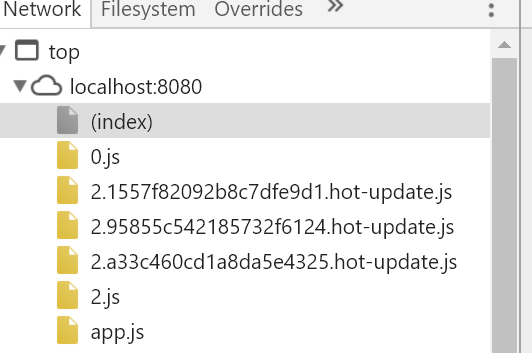


点击启动按钮，将启动一个chrome窗口，调试就在这个窗口进行，可以在代码中打断点（小红点），进行下一步，继续等操作

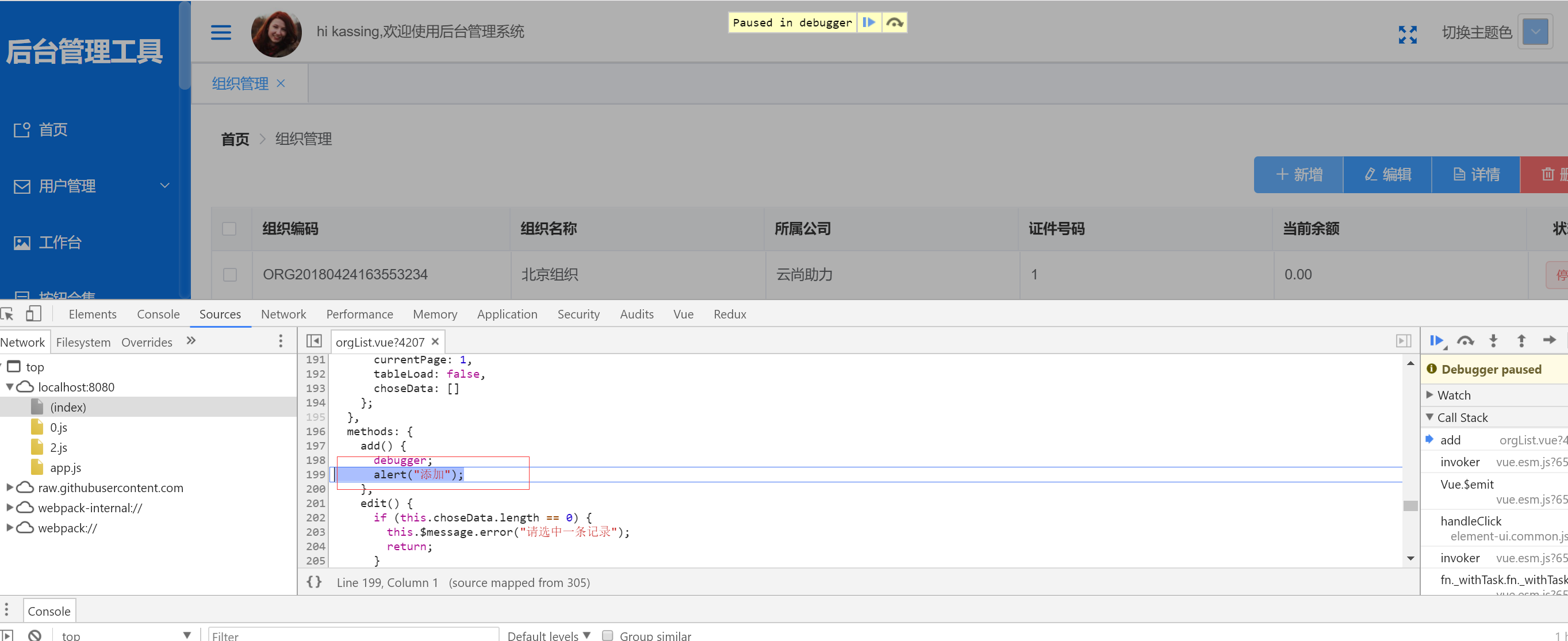
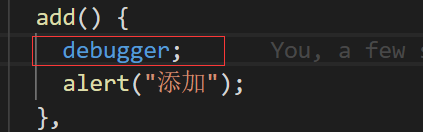


2、浏览器调试

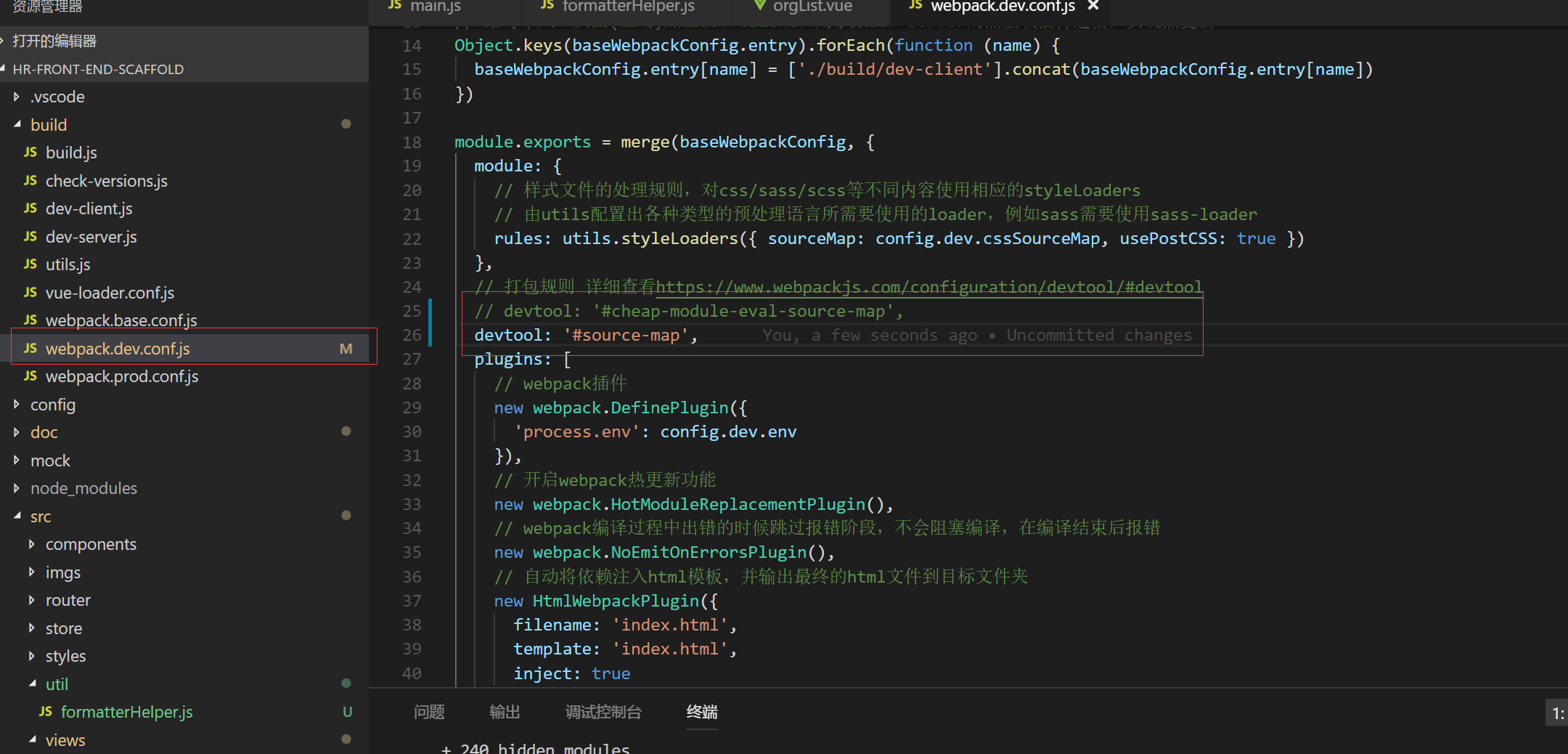
也可直接通过chrome浏览器调试，由于环境打包关系，直接通过浏览器调试看到的JS文件是编译过后的，如图：



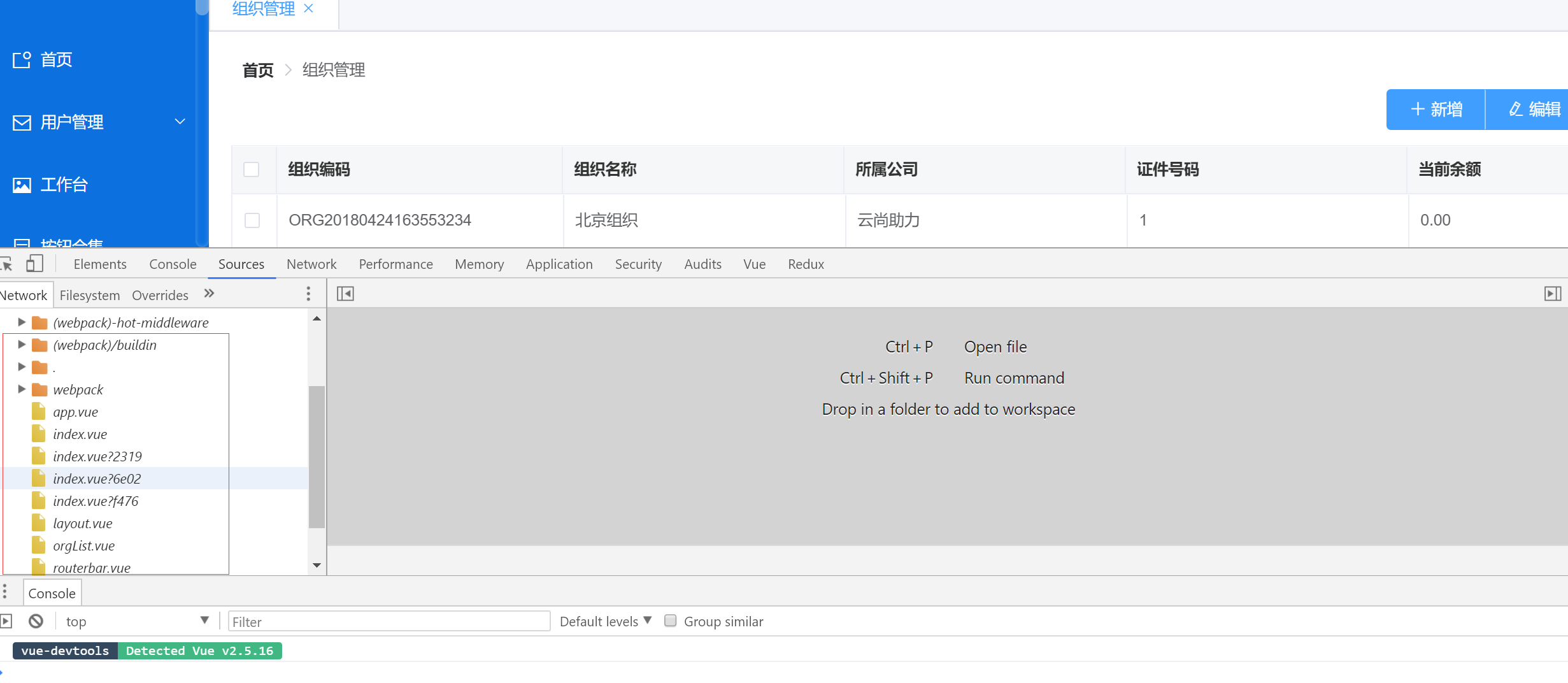
此时，可通过在代码中加入debugger;断点，这样执行到断点代码时，浏览器就会自动停止，并显示代码原来的文件内容，然后按需进行调试就可以了



还可以修改开发配置，如图：



这样，打开浏览器F12，下方将出现所有vue文件，可以直接看到源代码调试了：



# 如何mock数据

mock开关在：config/mock-filter.json处

"IS\_CLOSE\_ALL\_MOCK":false,

是否关闭所有本地mock，这个是总开关，当为true时，关闭所有本地mock

"/api/demo":true

单独请求mock开关，当设置为true时，表示这个请求关闭本次mock

mock数据配置在mock下

以/api/demo请求为例，api为目录，demo.json为模拟数据文件，文件内部为模拟请求返回的json格式数据

这样，当mock开关打开时，请求/api/demo就会返回mock\api\demo.json文件中的json格式数据

# 常用开发步骤

目前整体框架已经搭建好，我们日常的开发，实际就是在views中编写新页面的组件，在router中配置这个组件对应的路由即可，这样当url改变匹配配置的路由时，即可将路由对应的组件内容加载到layout的content中

组件的添加，可参见《昊润框架培训文档.pdf》中的《组件的添加和使用》版块，重点掌握组件间的数据通讯，父传子，子传父，非父子组件互传，重点理解props的定义和使用

当需要使用新的全局数据时，可操作store中的内容，知识点是vuex

# 如何打包发布

执行命令npm run build即可执行发布，脚手架会将编译后的内容统一发布在dist目录下

发布有几个重点配置项，在config\index.js中

// html入口文件打包后之后的文件名称和格式

index: path.resolve(\_\_dirname, '../dist/index.html'),

// 打包文件的生产路径

assetsRoot: path.resolve(\_\_dirname, '../dist'),

// 二级目录，存放静态资源文件的目录，位于dist文件夹下

assetsSubDirectory: 'static',

// 发布路径，如果构建后的产品文件有用于发布CDN或者放到其他域名的服务器，可以在这里进行设置例如(//cdn.xxx.com)

// 设置之后构建的产品文件在注入到index.html中的时候就会带上这里的发布路径

assetsPublicPath: '/',

生成文件的后缀，如果发布的前端代码需要跟后端模版引擎组合使用，则需要生成的页面必须符合后端引擎的后缀，则可以修改这里的设置

如果希望发布在其他目录下，则可以修改打包文件的生产路径

assetsSubDirectory静态问价内目录，目前配置的是static，如果有其他特殊要求，也可更改生成的静态目录名称

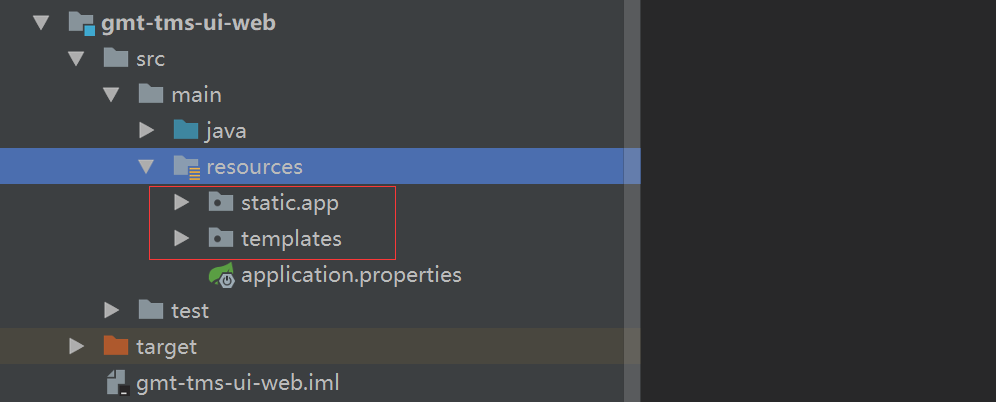
assetsPublicPath最重要的一个配置，如果需要生成的地址带有域名前缀，则可以通过这里进行配置

# 部署方式

前端目前有2种方式部署：

1. 与后端一起部署

结合后端的目录及文件命名规范，以我们自己的JAVA项目为例：



所有的静态资源和HTML都放在resources目录下，那么可以修改配置，将静态资源目录名称改为static.app，将生成的html代码放入templates目录中，即可结合后端框架一起，与后端进行部署

1. 单独部署

脚手架生成的文件最终就是HTML，JS，CSS等，直接通过域名指向部署目录即可完成部署，不需要再配置单独的部署环境