

Idea安装vue插件

idea 安装 Vue 插件没有Vue component选项

1. 点击 file 打开设置 settings，展开 Editor 找到 file and code templates
2. 找到 Vue single file component 并选中它，然后点击copy
3. 复制后底部出现了一个新的文件
4. 把 Name 改成 Vue Component，然后把代码里的 "COMPONENT_"删掉，最后点 ok 就完事了

创建vue-cli基础工程

1、下载node.js并安装

2、修改nodejs的环境变量

```
npm config set prefix "D:\Node\node_global"
```

```
npm config set cache "D:\Node\node_cache"
```

改变用户变量和系统变量

用户环境变量中添加：Path： D:\Node\node_global

系统变量中新建：NODE_PATH： D:\Node\node_global\node_modules

系统全局变量中添加：Path： %NODE_PATH%

3、安装cnpm命令：

```
npm install cnpm -g
```

4、安装vue-cli：

```
cnpm install vue-cli -g
```

5、查看是否安装成功：

```
vue list
```

```
C:\WINDOWS\system32>vue list

Available official templates:

★ browserify - A full-featured Browserify + vueify setup with hot-reload, linting & unit testing.
★ browserify-simple - A simple Browserify + vueify setup for quick prototyping.
★ pwa - PWA template for vue-cli based on the webpack template
★ simple - The simplest possible Vue setup in a single HTML file
★ webpack - A full-featured Webpack + vue-loader setup with hot reload, linting, testing & css extraction.
★ webpack-simple - A simple Webpack + vue-loader setup for quick prototyping.
```

6、初始化vue

创建vue在指定目录下

```
vue init webpack myvue
```

```
D:\IdeaProject\VUE>vue init webpack myvue

? Project name myvue
? Project description A Vue.js project
? Author liaoyouxin <liaodagege@outlook.com>
? Vue build standalone
? Install vue-router? No
? Use ESLint to lint your code? No
? Set up unit tests No
? Setup e2e tests with Nightwatch? No
? Should we run `npm install` for you after the project has been created? (recommended) no

vue-cli · Generated "myvue".

# Project initialization finished!
# =====

To get started:

  cd myvue
  npm install (or if using yarn: yarn)
  npm run dev

Documentation can be found at https://vuejs-templates.github.io/webpack
```

7、进入创建的vue项目并安装npm

```
cd myvue
npm install
```

如果出现错误，则按照指示解决错误即可

8、用idea打开项目

9、启动项目：

```
npm run dev
```

注：如果出现版本依赖问题可以降低npm版本：npm install npm@6.14.10 -g

Webpack

本质上,webpack是一个现代JavaScript应用程序的静态模块打包器(module bundler)。当webpack处理应用程序时，它会递归地构建一个依赖关系图(dependency graph)，其中包含应用程序需要的每个模块，然后将所有这些模块打包成一个或多个bundle。

webpack 是当下最热门的前端资源模块化管理和打包工具，它可以将许多松散耦合的模块按照依赖和规则打包成符合生产环境部署的前端资源。还可以将按需加载的模块进行代码分离，等到实际需要时再异步加载。通过loader转换，任何形式的资源都可以当做模块，比如CommonsJS、AMD、ES6、Css、JSON、CoffeeScript、LESS 等；

伴随着移动互联网的大潮，当今越来越多的网站已经从网页模式进化到了 WebApp模式。它们运行在现代浏览器里，使用HTML5、CSS3、ES6等新的技术来开发丰富的功能，网页已经不仅仅是完成浏览器的基本需求；WebApp通常是一个SPA(单页面应用)，每一个视图通过异步的方式加载，这导致页面初始化和使用过程中会加载越来越多的JS代码，这给前端的开发流程和资源组织带来了巨大的挑战。

前端开发和其他开发工作的主要区别，首先是前端基于多语言、多层次的编码和组织工作，其次前端产品的交付是基于浏览器的，这些资源是通过增量加载的方式运行到浏览器端，如何在开发环境组织好这些碎片化的代码和资源，并且保证他们在浏览器端快速、优雅的加载和更新，就需要一个模块化系统，这个理想中的模块化系统是前端工程师多年来一直探索的难题。

模块化演进

最开始的javascript标签:

这种原始的加载方式暴露了很多显而易见的弊端

- 全局作用域下容易造成变量冲突
- 文件只能按照的书写顺序进行加载
- 开发人员必须主观解决模块和代码库的依赖关系
- 在大型项目中各种资源难以管理，长期积累的问题导致代码库混乱不堪

CommonsJS

服务器端的NodeJS 遵循CommonsJS规范，该规范核心思想是允许模块通过require方法来同步加载所需依赖的其它模块，然后通过exports或 module.exports 来导出需要暴露的接口。

```
require("module");  
require("../module.js");  
export.diStuff = function(){};  
module.exports = someValue;
```

优点:

- 服务器端模块便于重用
- NPM中已经有超过45万个可以使用的模块包
- 简单易用

缺点:

- 浏览器资源是异步加载的
- 不能非阻塞的并行加载多个模块

实现:

- 服务器端的NodeJS
- Browserify,浏览器端的CommonsJS实现，可以使用NPM的模块，但是编译打包后的文件体积较大
- modules-webmake，类似Browserify，但不如Browserify灵活
- wreq: Broserify的前身

AMD

Asynchronous Module Definition规范其实主要一个主要接口define(id?,dependencies?, factory);它要在声明模块的时候指定所有的依赖dependencies，并且还要当做形参传到factory 中，对于依赖的模块提前执行。

```
define( "module", [ "dep1 ", "dep2 "], function(d1,d2y ireturn someExportedvalue  
;}) ;  
require( [ "module", " ../file.js"],function( module, file) {});
```

优点:

- 适合在浏览器环境中异步加载模块
- 可以并行加载多个模块

缺点:

- 成本增加

CMD

ES6模块

EcmaScript6标准增加了JavaScript语言层面的模块体系定义。ES6模块的设计思想，是尽量静态化，使编译时就能确定模块的依赖关系，以及输入和输出的变量。CommonJS和AMD模块，都只能在运行时确定这些东西。

```
import "jquery ";
export function dostuff() {}
module "localModule" {}
```

优点：

- 容易静态分析
- 面向伪类的EcmaScript标准

缺点：

- 原生的浏览器端还没有实现该标准
- 全新的命令，新版的NodeJS才支持

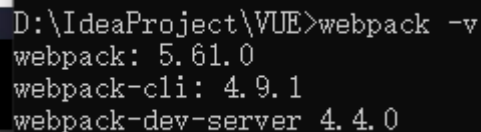
安装webpack

```
npm install webpack -g
```

```
npm install webpack-cli -g
```

测试安装是否成功：

```
webpack -v
```



```
D:\IdeaProject\VUE>webpack -v
webpack: 5.61.0
webpack-cli: 4.9.1
webpack-dev-server 4.4.0
```

使用webpack创建简单工程

- 1、创建webpack-study文件夹
- 2、使用Idea打开
- 3、webpack-study根目录中创建modules文件夹
- 4、创建hello.js并写一个暴露出来的方法：

```
"use strict"
//暴露一个方法
exports.sayHello = function () {
    document.write("<h1>hello,world</h1>")
}
```

- 5、同级目录下创建主入口main.js并接收暴露出来的方法：

```
var say = require("./hello");
say.sayHello();
```

6、创建webpack-config.js文件并导出主方法：

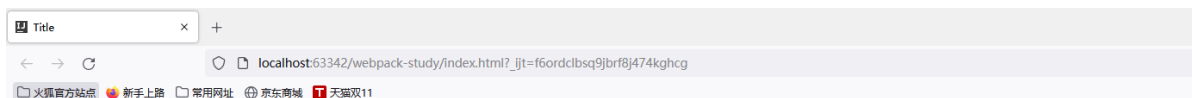
```
module.exports={
  entry: "./modules/main.js",
  output: {
    filename: "./js/bundle.js"
  },
  mode: 'development' // 设置mode
}
```

7、控制台使用webpack打包会自动生成根目录下dist/js/bundle.js文件

8、创建index.html并引入bundle.js文件

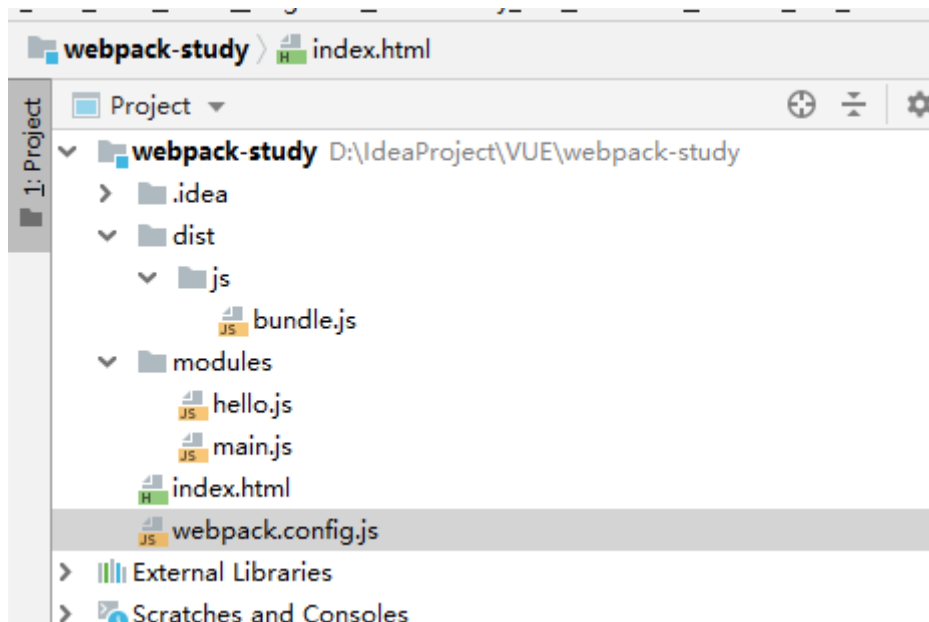
```
<!--
  Created by IntelliJ IDEA.
  User: youxin
  Date: 2021/11/4
  Time: 19:08
  To change this template use File | Settings | File Templates.
-->
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
</head>
<body>
<script src="./dist/js/bundle.js"></script>
</body>
</html>
```

9、启动项目：



hello,world

查看项目目录结构：



VUE-router路由：

Vue-router是Vue.js官方的路由管理器，它和Vue.js的核心深度集成，让构建单页面应用变得易如反掌，包含的功能有：

- 嵌套的路由/视图表
- 模块化的、基于组件的路由配置
- 路由参数、查询、通配符
- 基于Vue.js过渡系统的视图过渡效果
- 细粒度的导航控制
- 带有自动激活的CSS class的链接
- HTML5历史模式或hash模式，在IE9中自动降级
- 自动以的滚动条行为

安装

```
npm install vue-router --save-dev
```

如果在一个模块化工程中使用它，必须要通过Vue.use()明确的安装路由功能：

```
import Vue from 'vue'
import VueRouter from 'vue-router'
Vue.use(VueRouter);
```

在components文件夹中创建Content.vue文件：

```
<template>
  <h1>Hello, Content!</h1>
</template>

<script>
  export default {
    name: "Content"
  }
</script>

<style scoped>

</style>
```

根目录创建路由文件夹router，并且默认配置路由文件名字index.js

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

//引入自定义组件
import Content from '../components/Content'
import youxin from '../components/Main'

//安装路由
Vue.use(VueRouter);

//配置导出路由
export default new VueRouter({
  routes: [
    {
      //路由路径
      path: '/Content',
      //路由名称（可以不要）
      name: 'content',
      //跳转的组件
      component: Content
    },
    {
      //路由路径
      path: '/Main',
      //路由名称（可以不要）
      name: 'main',
      //跳转的组件
      component: youxin
    }
  ]
});
```

main.js中引用路由配置文件：

```
//引入router可以不用加index，默认为index为主配置文件
import router from './router'

/* eslint-disable no-new */
new Vue({
  el: '#app',
  router,
  components: { App },
  template: '<App/>'
});
```

App.vue中编写模板：

```
<template>
  <div id="app">
    
    <h1>首页</h1>
    <router-link to="/main">主页</router-link>
```

```

    <router-link to="/content">内容页</router-link>
    <router-view></router-view>
  </div>
</template>

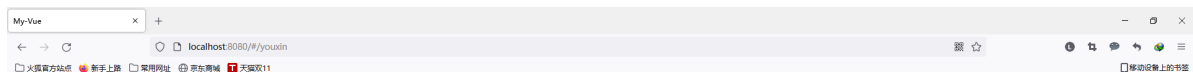
<script>

export default {
  name: 'App',
}
</script>

<style>
#app {
  font-family: 'Avenir', Helvetica, Arial, sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  text-align: center;
  color: #2c3e50;
  margin-top: 60px;
}
</style>

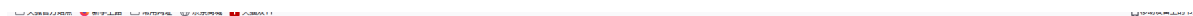
```

结果:



首页

[主页](#) [内容页](#)



首页

[主页](#) [内容页](#)

hello,main!



首页

[主页](#) [内容页](#)

Hello,Content!

vue+elementUI

1、初始化项目

```
vue init webpack "hello-vue"
```

2、安装vue-router 和 elementUI

```
# 进入工程目录
cd hello-vue
# 安装 vue-router
npm install vue-router --save-dev
# 安装 element-ui
npm i element-ui -S
# 安装依赖
npm install
# 安装 SASS 加载器
cnpm install sass-loader node-sass --save-dev
# 启动测试
npm run dev
```

3、进入项目创建自定义登陆组件

```
<template>
  <div>
    <el-form ref="loginForm" :model="form" :rules="rules" label-width="80px"
    class="login-box">
      <h3 class="login-title">欢迎 登录</h3>
      <el-form-item label=" 账号" prop="username">
        <el-input type="text" placeholder="请输入账号" v-model="form.username"/>
      </el-form-item>
      <el-form-item label=" 密码" prop="password">
        <el-input type="password" placeholder=" 请输入密码" v-
        model="form.password"/>
      </el-form-item>
      <el-form-item>
        <el-button type="primary" v-on:click="onSubmit('loginForm')">登录</el-
        button>
      </el-form-item>
    </el-form>
    <el-dialog
      title="温馨提示"
      :visible.sync="dialogvisible"
      width="30%"
      :before-close="handleClose">
```

```

    <span>请输入账号和密码</span>
    <span slot="footer" class="dialog- footer">
      <el-button type="primary" @click="dialogVisible = false">确定</el-button>
    </span>
  </el-dialog>
</div>
</template>•

<script>
  export default {
    name: "Login",
    data() {
      return {
        form: {
          username: '',
          password: ''
        },
        //表单验证，需要在el-form-item 元素中增加prop 属性
        rules: {
          username: [
            {required: true, message: " 账号不可为空", trigger: 'blur'}
          ],
          password: [
            {required: true, message: " 密码不可为空 ", trigger: 'blur'}
          ]
        },
        //对话框显示和隐藏
        dialogVisible: false
      }
    },
    methods: {
      onSubmit(formName) {
        //为表单绑定验证功能
        this.$refs [formName].validate((valid) => {
          if (valid) {
            //使用vue-router路由到指定页面，该方式称之为程式导航
            this.$router.push("/main");
          } else {
            this.dialogVisible = true;
            return false;
          }
        });
      }
    }
  }
</script>

<style scoped>
  .login-box {
    border: 1px solid #DCDFE6;
    width: 350px;
    margin: 180px auto;
    padding: 35px 35px 15px 35px;
    border-radius: 5px;
    -webkit-border-radius: 5px;
    -moz-border-radius: 5px;
    box-shadow: 0 0 25px #909399;
  }

```

```
.login-title {  
  text-align: center;  
  margin: 0 auto 40px auto;  
  color: #303133;  
}  
</style>•
```

4、配置路由index.js

```
import Vue from 'vue'  
import VueRouter from 'vue-router'  
  
//引入自定义组件  
import Login from '../views/Login'  
import Main from '../views/Main'  
  
Vue.use(VueRouter);  
  
export default new VueRouter({  
  routes: [  
    {  
      path: '/login',  
      component: Login  
    },  
    {  
      path: '/main',  
      component: Main  
    }  
  ]  
});
```

5、main.js中引入

```
// The vue build version to load with the `import` command  
// (runtime-only or standalone) has been set in webpack.base.conf with an alias.  
import Vue from 'vue'  
import App from './App'  
import ElementUI from 'element-ui';  
import 'element-ui/lib/theme-chalk/index.css';  
import router from './router'  
  
Vue.config.productionTip = false;  
  
Vue.use(ElementUI, router);  
  
/* eslint-disable no-new */  
new Vue({  
  el: '#app',  
  router,  
  render: h => h(App)  
});
```

6、App.vue中加入组件

```
<template>
  <div id="app">
    <!---->
    <!--<HelloWorld/>-->
    <router-link to="/login">登陆</router-link>
    <router-view></router-view>
  </div>
</template>

<script>
// import HelloWorld from './components/HelloWorld'

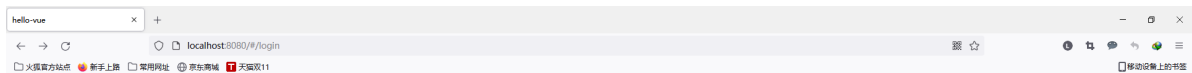
export default {
  name: 'App',
  /*components: {
    HelloWorld
  }*/
}
</script>

<style>
#app {
  font-family: 'Avenir', Helvetica, Arial, sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  text-align: center;
  color: #2c3e50;
  margin-top: 60px;
}
</style>
```

6、启动项目

```
npm run dev
```

7、启动成功



登陆

欢迎 登录

* 账号

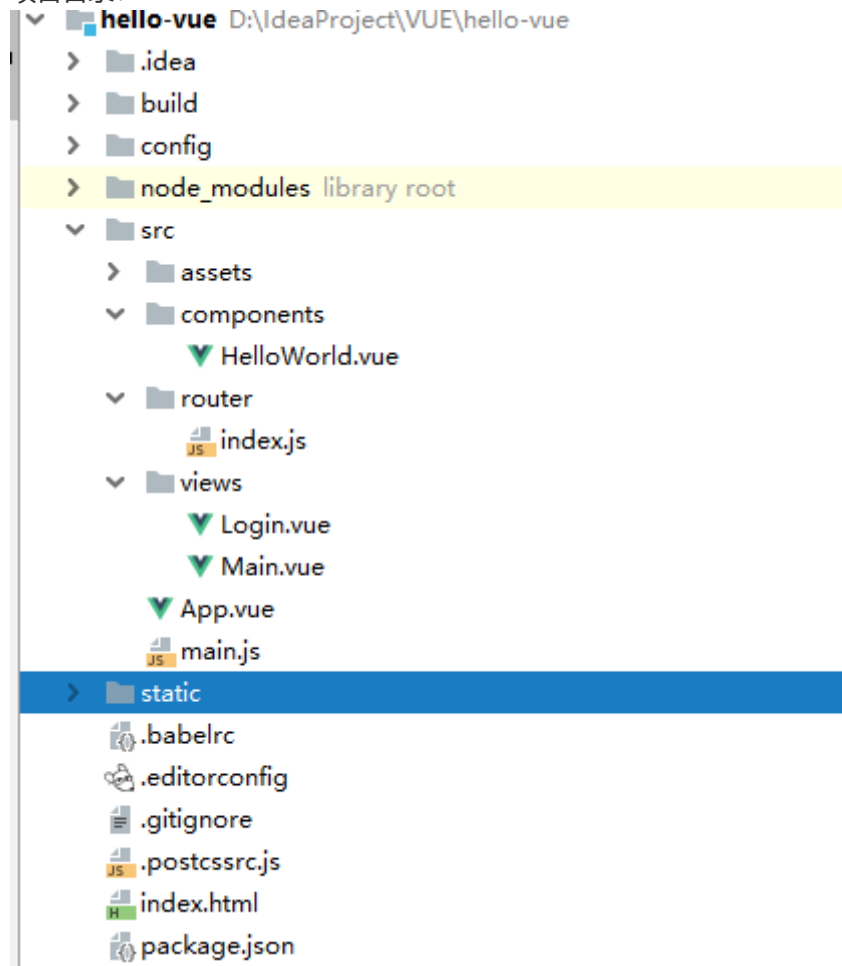
请输入账号

* 密码

请输入密码

登录

项目目录：



如果出现getOpition则可以降低版本解决问题

路由嵌套

1、创建子组件如user/List.vue

```
<template>
  <h1>用户列表</h1>
</template>

<script>
  export default {
    name: "List"
  }
</script>

<style scoped>

</style>
```

2、index.js中路由嵌套子路由：

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

//引入自定义组件
import Login from '../views/Login'
```

```

import Main from '../views/Main'
import UserList from '../views/user/List'
import UserProFile from '../views/user/ProFile'

Vue.use(VueRouter);

export default new VueRouter({
  routes: [
    {
      path: '/login',
      component: Login
    },
    {
      path: '/main',
      component: Main,
      children: [
        {
          path: '/user/List',
          component: UserList
        },
        {
          path: '/user/ProFile',
          component: UserProFile
        }
      ]
    }
  ]
});

```

4、因为是在Main页面嵌套，所以在Main.vue中引用即可：

```

<template>
  <div>
    <el-container>
      <el-aside width="200px">
        <el-menu :default-openeds="['1']">
          <el-submenu index="1">
            <template slot="title"><i class="el-icon-caret-right"></i>用户管理
          </template>
          <el-menu-item-group>
            <el-menu-item index="1-1">
              <router-link to="/user/profile">个人信息</router-link>
            </el-menu-item>
            <el-menu-item index="1-2">
              <router-link to="/user/list">用户列表</router-link>
            </el-menu-item>
          </el-menu-item-group>•
        </el-submenu>
        <el-submenu index="2">
          <template slot="title"><i class="el-icon-caret-right"></i>内容管理
        </template>
        <el-menu-item-group>
          <el-menu-item index="2-1">分类管理</el-menu-item>
          <el-menu-item index="2-2">内容列表</el-menu-item>
        </el-menu-item-group>
      </el-submenu>
    </el-aside>
  </div>
</template>

```

```

        </el-menu>
      </el-aside>

      <el-container>
        <el-header style="text-align: right; font-size: 12px;">
          <el-dropdown>
            <i class="el-icon-setting" style="margin-right: 15px;"></i>
            <el-dropdown-menu slot="dropdown">
              <el-dropdown-item>个人信息</el-dropdown-item>
              <el-dropdown-item>退出登陆</el-dropdown-item>
            </el-dropdown-menu>
          </el-dropdown>
        </el-header>•
        <el-main>
          <router-view/>
        </el-main>
      </el-container>
    </el-container>
  </div>
</template>

<script>
  export default {
    name: "Main"
  }
</script>

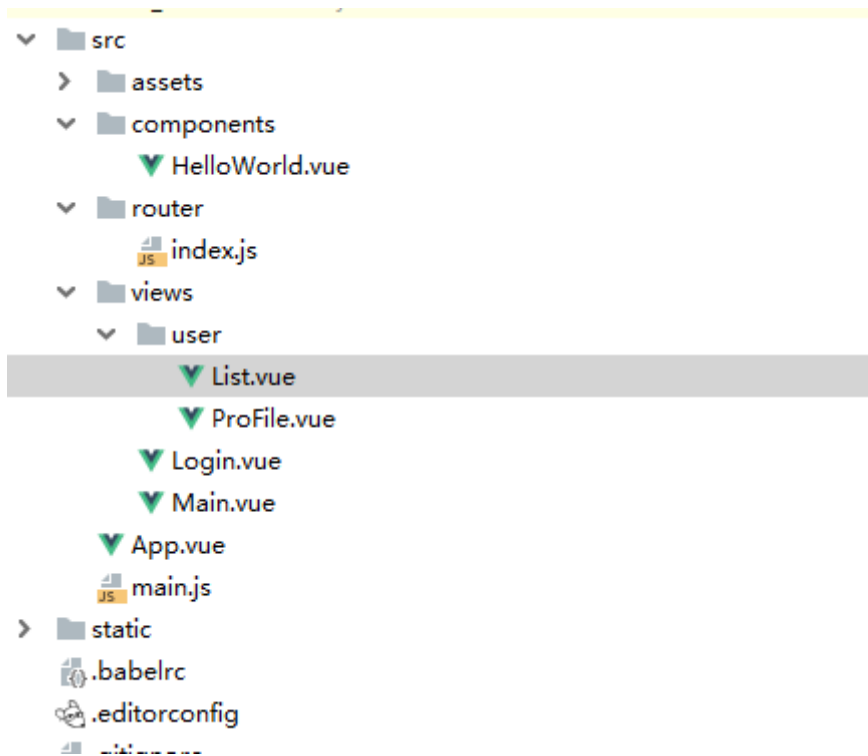
<style scoped>
  .el-header{
    background-color: lightblue;
    color: #333;
    line-height: 60px;
  }
  .el-aside{
    color: #333;
  }
</style>•

```

运行:



查看目录结构:



参数传递及重定向

传递参数方法一：

主页面：

```
<el-menu-item index="1-1">
  <!-- v-bind:to=""能够传递参数，其中name：路由名称，params：{参数名：参
数值} -->
  <router-link :to="{name: 'UserProFile',params: {id: 1}}">个人信息
</router-link>
</el-menu-item>
```

路由配置器：

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

//引入自定义组件
import Login from '../views/Login'
import Main from '../views/Main'
import UserList from '../views/user/List'
import UserProFile from '../views/user/ProFile'

Vue.use(VueRouter);

export default new VueRouter({
  routes: [
    {
      path: '/login',
      component: Login
    },
    {
      path: '/main',
```



```

    component: Main,
    children: [
      {
        path: '/user/List',
        component: UserList
      },
      {
        //绑定接收参数, props: true 开启参数传递
        path: '/user/ProFile/:id',
        name: 'UserProFile',
        component: UserProFile
      }
    ]
  }
]
});

```

获取参数:

```

<template>
  <!-- 注意: 所有标签和组件必须在一个根标签中 (如div标签) -->
  <div>
    <h1>个人信息</h1>
    {{ $route.params.id }}
  </div>
</template>

```

参数传递方法二: 绑定参数时props: true

```

import Vue from 'vue'
import VueRouter from 'vue-router'

//引入自定义组件
import Login from '../views/Login'
import Main from '../views/Main'
import UserList from '../views/user/List'
import UserProFile from '../views/user/ProFile'

Vue.use(VueRouter);

export default new VueRouter({
  routes: [
    {
      path: '/login',
      component: Login
    },
    {
      path: '/main',
      component: Main,
      children: [
        {
          path: '/user/List',
          component: UserList
        },
        {

```

```

        //绑定接收参数, props: true 开启参数传递
        path: '/user/ProFile/:id',
        name: 'UserProFile',
        props: true,
        component: UserProFile
      }
    ]
  }
]
});

```

接收:

```

<template>
  <!-- 注意: 所有标签和组件必须在一个根标签中 (如div标签) -->
  <div>
    <h1>个人信息</h1>
    {{id}}
  </div>
</template>

<script>
  export default {
    props: ['id'],
    name: "Profile"
  }
</script>

<style scoped>

</style>

```

最后得出效果一样

页面重定向

路由配置中绑定重定向:

```

import Vue from 'vue'
import VueRouter from 'vue-router'

//引入自定义组件
import Login from '../views/Login'
import Main from '../views/Main'
import UserList from '../views/user/List'
import UserProFile from '../views/user/ProFile'

Vue.use(VueRouter);

export default new VueRouter({
  routes: [
    {
      path: '/login',
      component: Login
    },
  ],

```

```

{
  path: '/main',
  component: Main,
  children: [
    {
      path: '/user/List',
      component: UserList
    },
    {
      //绑定接收参数, props: true 开启参数传递
      path: '/user/ProFile/:id',
      name: 'UserProFile',
      props: true,
      component: UserProFile
    }
  ]
},
{
  path: '/toHome',
  redirect: '/main'
}
]
});

```

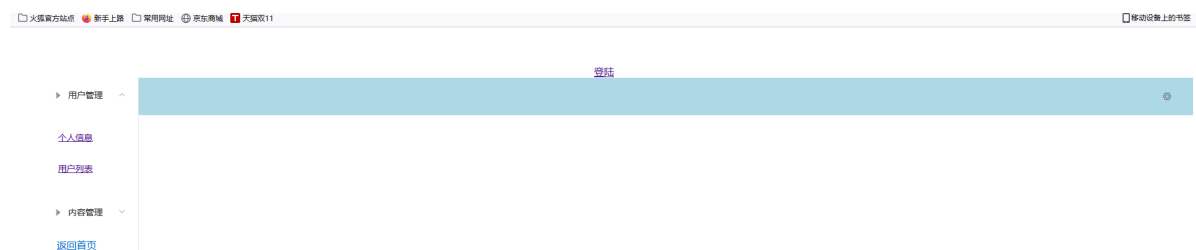
main.vue中添加侧边栏

```

<el-menu-item-group>
  <router-link to="/toHome">返回首页</router-link>
</el-menu-item-group>

```

测试:



路由模式与404

路由模式有两种:

- hash: 路径带#号, 如<http://localhost/#/login>
- history: 路径不带#号, 如<http://localhost/login>

修改路由配置如下:

```

export default new Router({
  mode: 'history',
  routes: []
})

```

处理404页面：

创建notfound.vue并导入路由配置

```
<template>
  <div>
    <h1>糟糕，页面走丢了！ </h1>
  </div>
</template>

<script>
  export default {
    name: "404"
  }
</script>

<style scoped>

</style>
```

添加路由配置：

```
import NotFound from '../views/404'
{
  path: '*',
  component: NotFound
}
```

钩子路由及异步请求

`beforeRouteEnter`：在进入路由前执行

`beforeRouteLeave`：在离开路由之前执行

参数说明：

- to：路由将要跳转的路径信息
- from：路径跳转前的路径信息
- next：路由的控制参数
 - next()跳入下一个页面
 - next('/path') 改变路由的跳转方向，使其跳转到另一个路由
 - next(false) 返回原来的页面
 - next(vm=>{}) 仅在beforeRouteEnter中可用，vm是组件实例

```
<script>
  export default {
    props: ['id'],
    name: "Profile",
    //相当于过滤器: to : request, from : response, next : chain
    beforeRouteEnter: (to, from, next)=> {
      alert("进入个人信息之前");
      next();
    }
  }
</script>
```

```

    },
    beforeRouteLeave: (to, from, next) => {
      alert("离开路由之前");
      next();
    }
  }
}
</script>

```

异步请求

安装axios:

```
npm install --save axios vue-axios
```

引入到主入口文件:

```

import axios from 'axios'
import VueAxios from 'vue-axios'

Vue.use(VueAxios, axios)

```

编写请求:

```

<template>
  <!-- 注意: 所有标签和组件必须在一个根标签中 (如div标签) -->
  <div>
    <h1>个人信息</h1>
    {{id}}
  </div>
</template>

<script>
  export default {
    props: ['id'],
    name: "Profile",
    //相当于过滤器: to : request, from : response, next : chain
    beforeRouteEnter: (to, from, next) => {
      alert("进入个人信息之前");//加载数据
      next(vm => {
        vm.getData();//进入路由之前执行getData
      });
    },
    beforeRouteLeave: (to, from, next) => {
      alert("离开路由之前");
      next();
    },
    methods: {
      getData: function () {
        this.axios({
          method: 'get',
          url: 'http://localhost:8080/static/mock/data.json',
        }).then(function (response) {
          console.log(response);
        });
      }
    }
  }
}

```

```
}  
</script>  
  
<style scoped>  
  
</style>
```