

01、旅游项目后台pug搭建全过程 - 项目介绍

Vue成熟后台

- Antd – 自研组件
- 若依 - 扩展性
- Vue-admin
- Pug

01、概述

- 官网: <https://cli.vuejs.org/zh/>
- vue3关注文档: <https://v3.cn.vuejs.org/guide/introduction.html>



Vue CLI

 Vue.js 开发的标准工具

开始 →

功能丰富

对 Babel、TypeScript、ESLint、PostCSS、PWA、单元测试和 End-to-end 测试提供开箱即用的支持。

易于扩展

它的插件系统可以让社区根据常见的需求组织和共享可复用的解决方案。

单独弹出

Vue CLI 完全是可配置的，无需弹出。这样你的项目就可以一直保持更新了。

CLI 超越了图形化界面

通过配套的图形化界面创建、开发和管理你的项目。

即刻制作原型

用刻新的 Vue 文件即实践的灵感。

给未来

为现代浏览器轻松打造的ES2015产品代码，或将你的Vue组件构建为果实的Web Components组件。

02、简介

VueCli是一个基于vuejs进行快速开发的完整系统。在Vue3.0版本正式发布时，vuecli将包名由原来的vue-cli改成了@vue/cli。它由三个组件组件：

- CLI（@vue/cli）全局安装的npm包，提供了终端里的vue命令，比如(vue create、vue serve、vue ui 等)它提供了通过vue create 快速搭建创建新项目的能力，或者通过vue serve及时原型化新想法的能力。还可以使用vue ui使用图形化界面管理项目。

- CLI 服务（@vue/cli-server）：cli服务是一个开发环境依赖，它是一个npm包，本地安装到@vue/cli创建的每个项目中，cli服务是构建于webpack和webpack-dev-server之上的，它包含了
 - 加载其他cli插件的核心服务
 - 一个为绝大部分的应用优化过的内部webpack配置 npm run
 - 项目内部的vue-cli-server命令，包含了基本的serve,build和inspect命令
- CLI插件：cli插件是给vue项目提供可选功能的npm包，比如：集成Babel/Typescript，esLint集成，单元测试等等，@vue/cli-plugin-(用于内置插件)或者vue-cli-plugin-(用于社区插件)开头，非常容易使用，当项目内部运行 vue-cli-server命令时。它会自动解析并加载项目的package.json文件列出所有的cli插件。

最核心：单页开发（SPA），前端后分离。

SPA单页开发

- 每个页面都是一个组件，也称之为模块，也称之为：Page（页面）
- 每个SPA页面都独立的vue生命周期页面格式如下：它其实是参考一个页面的完整形态而来

比如传统：login.html

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
```

```
5     <title>Title</title>
6     <style>
7         /*这里是样式*/
8     </style>
9 </head>
10 <body>
11     <!--这里是模板-->
12     <h1>我是登录页面</h1>
13     <script>
14         // 这里是js
15     </script>
16 </body>
17 </html>
```

login.vue

```
1 <template>
2     <!--这里是模板-->
3     <h1>我是登录页面</h1>
4 </template>
5
6 <script>
7     // 这里是js
8 </script>
9
10 <style>
11     /*这里是样式*/
12 </style>
```

使用SPA有一个好处：

- 有完整的vue生命周期，告诉：可以在这里使用vue的语法来完成的业务开发

- 使用SPA有个好处，你可以大胆的去使用ES6和ES7新语法。因为内置的babel编译器和webpack编译器它会自动把你的高版本js语法转换成低版本的浏览器识别的语法。
- 维护更加的方便。
- 模块：说明可以导入导出，其实我们用VUE-CLI其实就是想用js面向对象的语法。它也是js面向对象编程的一种迎合和一种期许。

03、创建项目

准备工作：安装nodejs : <http://nodejs.cn/download/>

Vue CLI 4.x 需要Node.js v8.9 或更高版本（推荐 v10 以上）。你可以使用n, nvm或nvm-windows在同一台电脑中管理多个 Node 版本。

安装

```
1 npm install -g @vue/cli
2 # OR
3 npm install -g yarn
4 yarn global add @vue/cli
```

```
Microsoft Windows [版本 10.0.19044.1415]  
(c) Microsoft Corporation。保留所有权利。  
D:\旅游项目实战开发\07、学相伴旅游项目实战-高级工程师实战部分&发布部署\前端代码\projects>npm install -g @vue/cli  
[理想树] | idealTree:npm: sill idealTree buildDeps
```

在安装之后，你 `vue` 可以通过简单 `vue` 的验证中运行命令。你还可以用这个命令来检查其版本是否正确：

```
1 vue --version
```

创建一个项目

```
1 vue create my-project
```

1、选择好项目存放的目录，打开命令创建创建，输入 `vue create helloworld`，开始创建一个 `helloworld` 项目。如下图：

```
Vue CLI v4.5.15
? Please pick a preset:
  Default ([Vue 2] babel, eslint)
> Default (Vue 3) ([Vue 3] babel, eslint)
  Manually select features
```

用键盘的方向键向下，选择vue3的语法即可。然后敲回车enter即可。会自动安装babel,eslint插件。

```
D:\旅游项目实战开发\07、学相伴旅游项目实战-高级工程师实战部分&发布部署\前端代码\projects>vue create helloworld

Vue CLI v4.5.15
? Please pick a preset: Default (Vue 3) ([Vue 3] babel, eslint)

Vue CLI v4.5.15
☐ Creating project in D:\旅游项目实战开发\07、学相伴旅游项目实战-高级工程师实战部分&发布部署\前端代码\projects\helloworld.
 $\diamond$  Initializing git repository...
 $\square\square$  Installing CLI plugins. This might take a while...

added 1277 packages in 36s

10 packages are looking for funding
  run `npm fund` for details
 $\diamond$  Invoking generators...
 $\diamond$  Installing additional dependencies...

added 73 packages in 4s

10 packages are looking for funding
  run `npm fund` for details
 $\square$  Running completion hooks...

 $\diamond$  Generating README.md...
 $\diamond$  Successfully created project helloworld.
 $\diamond$  Get started with the following commands:

$ cd helloworld
$ npm run serve

D:\旅游项目实战开发\07、学相伴旅游项目实战-高级工程师实战部分&发布部署\前端代码\projects>
```

注意：项目名中不能有大写字母。

2: 然后根据提示在命令提示符窗口依次输入`cd helloworld`和`npm run serve`（运行项目）。运行结果如下:

```
40% building 154/155 modules 1 active ... \前端代码\projects\hello\node_modules\eslint-loader\index.js??ref--14-0!D:\旅游项目实战开发\07、学组件旅游项目实战-高级工程师实战部分&发布部署\前端代码\projec
40% building 154/157 modules 3 active ... 程部实战部分&发布部署\前端代码\projects\hello\node_modules\url-loader\dist\cjs.js??ref--2-0!D:\旅游项目实战开发\07、学组件旅游项目实战-高级工程师实战部分&发布
40% building 162/163 modules 1 active ... 前端代码\projects\hello\node_modules\eslint-loader\index.js??ref--14-0!D:\旅游项目实战开发\07、学组件旅游项目实战-高级工程师实战部分&发布部署\前端代码\projec
40% building 163/164 modules 1 active ... oader\index.js??ref--14-0!D:\旅游项目实战开发\07、学组件旅游项目实战-高级工程师实战部分&发布部署\前端代码\projects\hello\src\components\HelloWorld.vue?vue&typ
40% building 164/165 modules 1 active ... _e_modules\eslint-loader\index.js??ref--14-0!D:\旅游项目实战开发\07、学组件旅游项目实战-高级工程师实战部分&发布部署\前端代码\projects\hello\src\components\Hell
40% building 165/166 modules 1 active ... ?ref--14-0!D:\旅游项目实战开发\07、学组件旅游项目实战-高级工程师实战部分&发布部署\前端代码\projects\hello\src\components\HelloWorld.vue?vue&type=style&index=
40% building 166/167 modules 1 active ... 0!dist\index.js??ref--1-1!D:\旅游项目实战开发\07、学组件旅游项目实战-高级工程师实战部分&发布部署\前端代码\projects\hello\src\components\HelloWorld.vue?vue&type
40% building 166/168 modules 2 active ... ules\vue-loader-v16!dist\index.js??ref--1-1!D:\旅游项目实战开发\07、学组件旅游项目实战-高级工程师实战部分&发布部署\前端代码\projects\hello\src\components\Hell
40% building 166/169 modules 3 active ... s??ref--1-1!D:\旅游项目实战开发\07、学组件旅游项目实战-高级工程师实战部分&发布部署\前端代码\projects\hello\src\components\HelloWorld.vue?vue&type=style&index=
40% building 167/169 modules 2 active ... ules\vue-loader-v16!dist\index.js??ref--1-1!D:\旅游项目实战开发\07、学组件旅游项目实战-高级工程师实战部分&发布部署\前端代码\projects\hello\src\components\Hell
40% building 167/170 modules 3 active ... s??ref--1-1!D:\旅游项目实战开发\07、学组件旅游项目实战-高级工程师实战部分&发布部署\前端代码\projects\hello\src\components\HelloWorld.vue?vue&type=style&index=
40% building 171/173 modules 2 active ... ules\vue-loader-v16!dist\index.js??ref--1-1!D:\旅游项目实战开发\07、学组件旅游项目实战-高级工程师实战部分&发布部署\前端代码\projects\hello\src\components\Hell
40% building 172/173 modules 1 active ... ules\vue-loader-v16!dist\index.js??ref--1-1!D:\旅游项目实战开发\07、学组件旅游项目实战-高级工程师实战部分&发布部署\前端代码\projects\hello\src\components\Hell
98% after emitting CopyPlugin

DONE Compiled successfully in 23422ms 下午2:00:15

App running at:
- Local: http://localhost:8080/
- Network: http://192.168.0.102:8080/

Note that the development build is not optimized.
To create a production build, run yarn build.
```



Welcome to Your Vue.js App

For a guide and recipes on how to configure / customize this project,
check out the [vue-cli documentation](#).

Installed CLI Plugins

[babel](#) [eslint](#)

Essential Links

[Core Docs](#) [Forum](#) [Community Chat](#) [Twitter](#) [News](#)

Ecosystem

[vue-router](#) [vuex](#) [vue-devtools](#) [vue-loader](#) [awesome-vue](#)

vue create 有一些可选项，你可以通过运行以下命令进行探索:

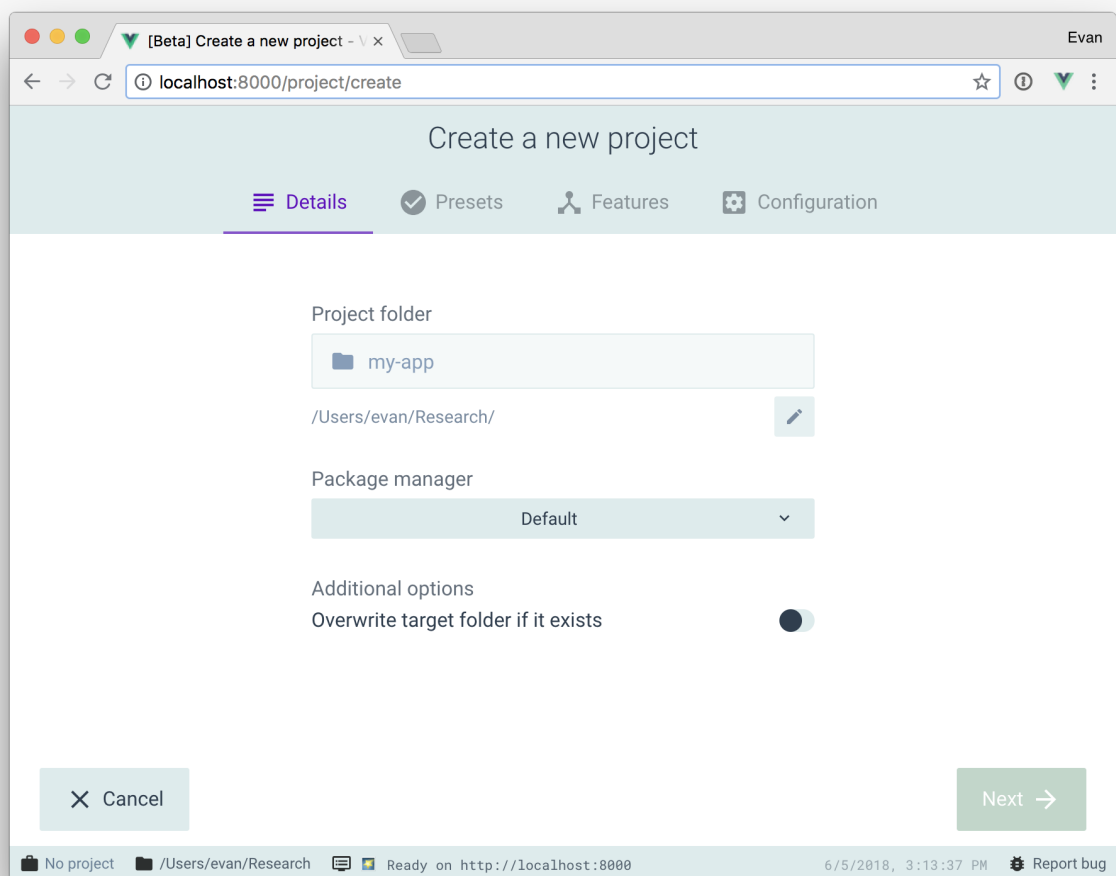

```
1 vue create --help
2 用法: create [options] <app-name>
3
4 创建一个由 `vue-cli-service` 提供支持的新项目
5
6
7 选项:
8
9   -p, --preset <presetName>      忽略提示符并使用已保存的
   或远程的预设选项
10  -d, --default                    忽略提示符并使用默认预设
   选项
11  -i, --inlinePreset <json>      忽略提示符并使用内联的
   JSON 字符串预设选项
12  -m, --packageManager <command> 在安装依赖时使用指定的
   npm 客户端
13  -r, --registry <url>           在安装依赖时使用指定的
   npm registry
14  -g, --git [message]            强制 / 跳过 git 初始
   化, 并可选的指定初始化提交信息
15  -n, --no-git                   跳过 git 初始化
16  -f, --force                     覆写目标目录可能存在的配
   置
17  -c, --clone                    使用 git clone 获取远
   程预设选项
18  -x, --proxy                    使用指定的代理创建项目
19  -b, --bare                      创建项目时省略默认组件中
   的新手指导信息
20  -h, --help                     输出使用帮助信息
```

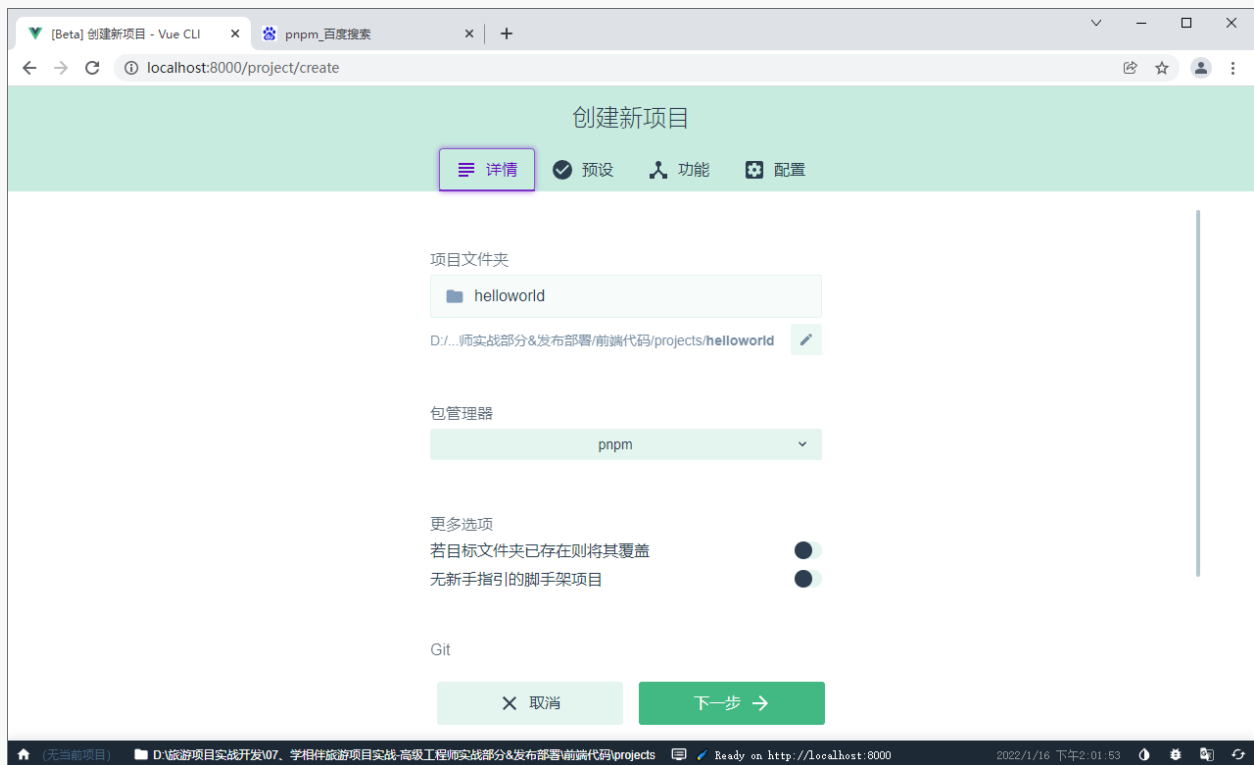
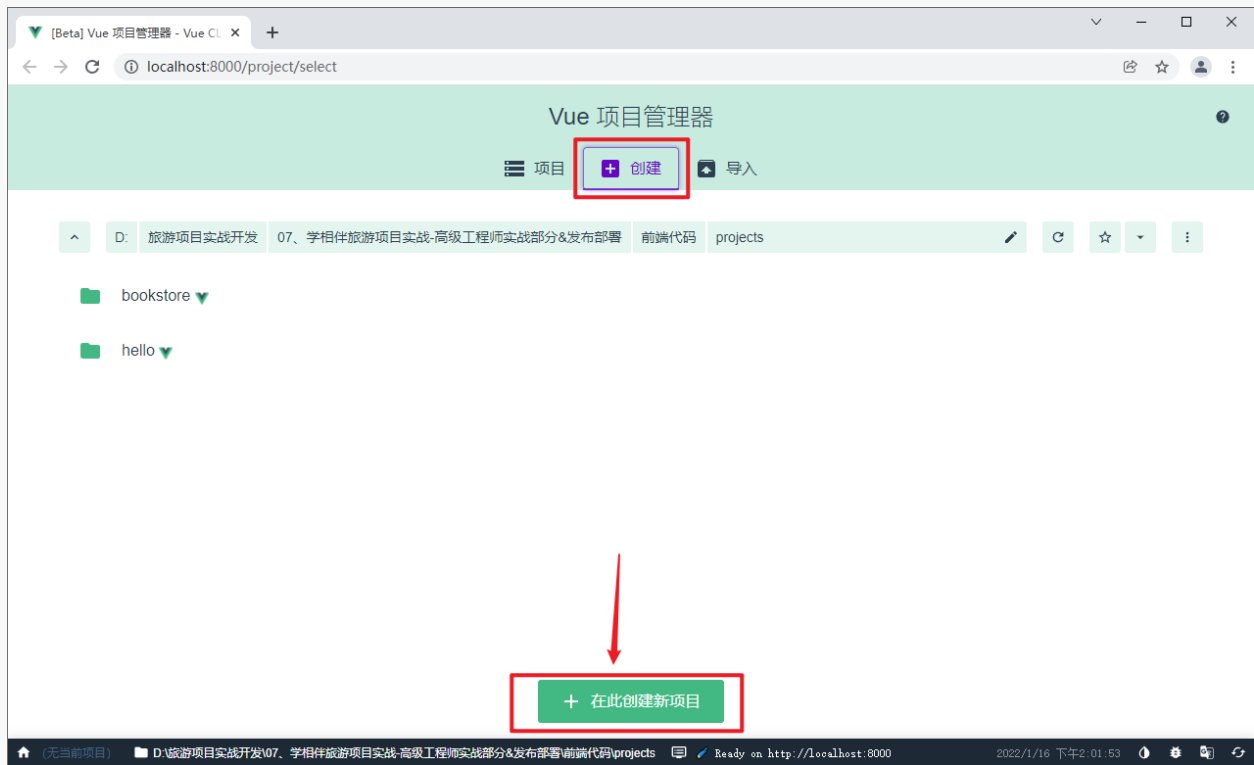
使用图形化界面

你也可以通过 `vue ui` 命令以图形化界面创建和管理项目：

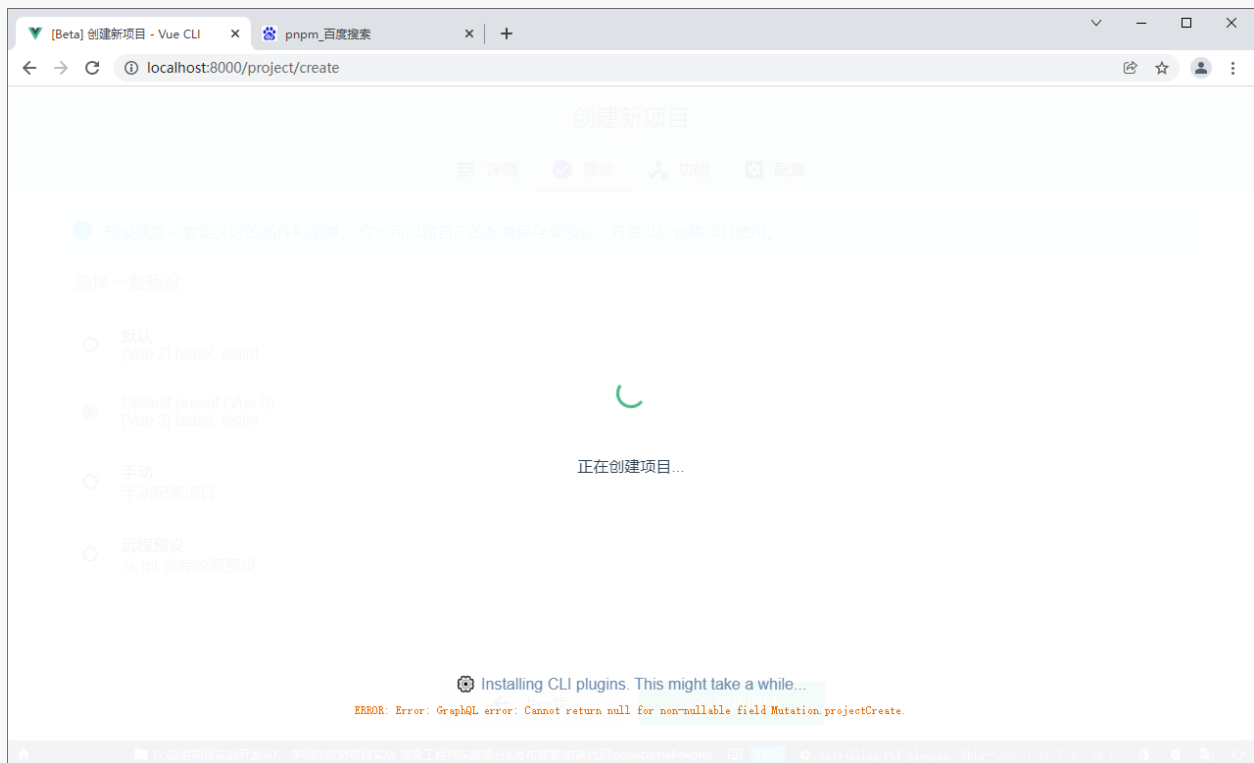
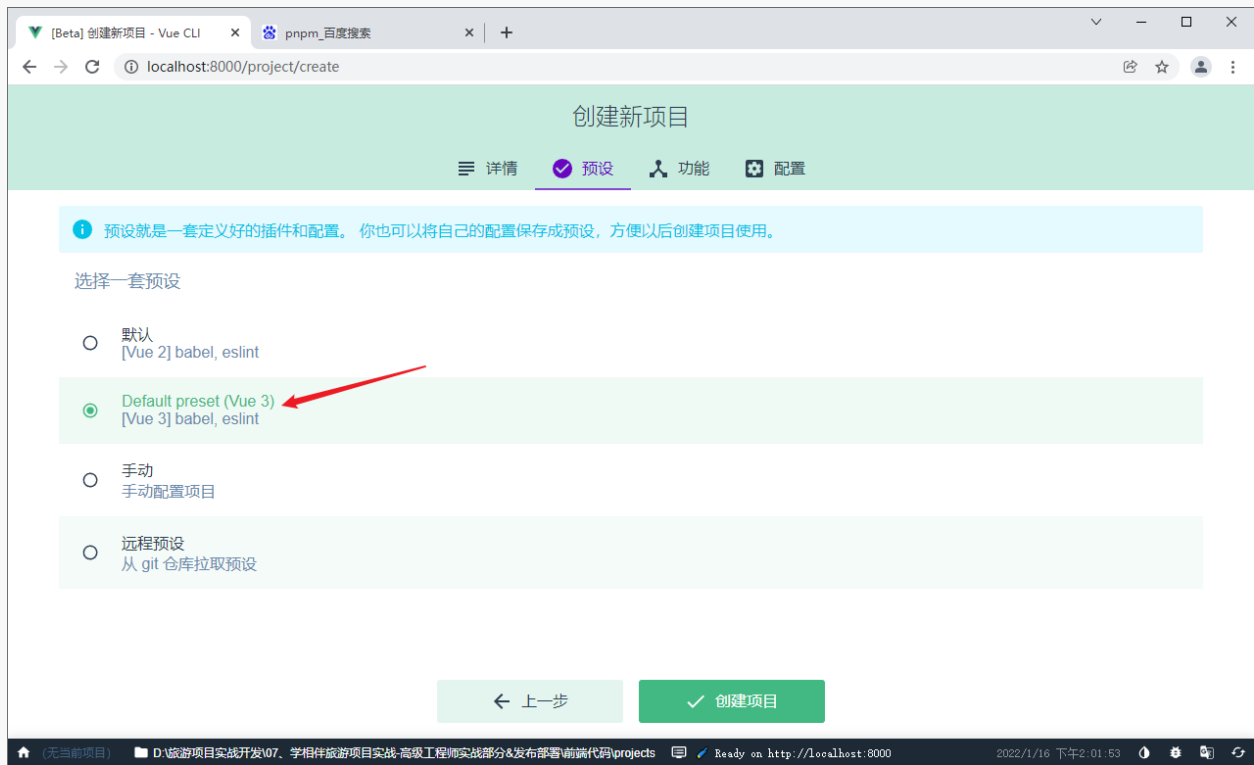
```
1 vue ui
```

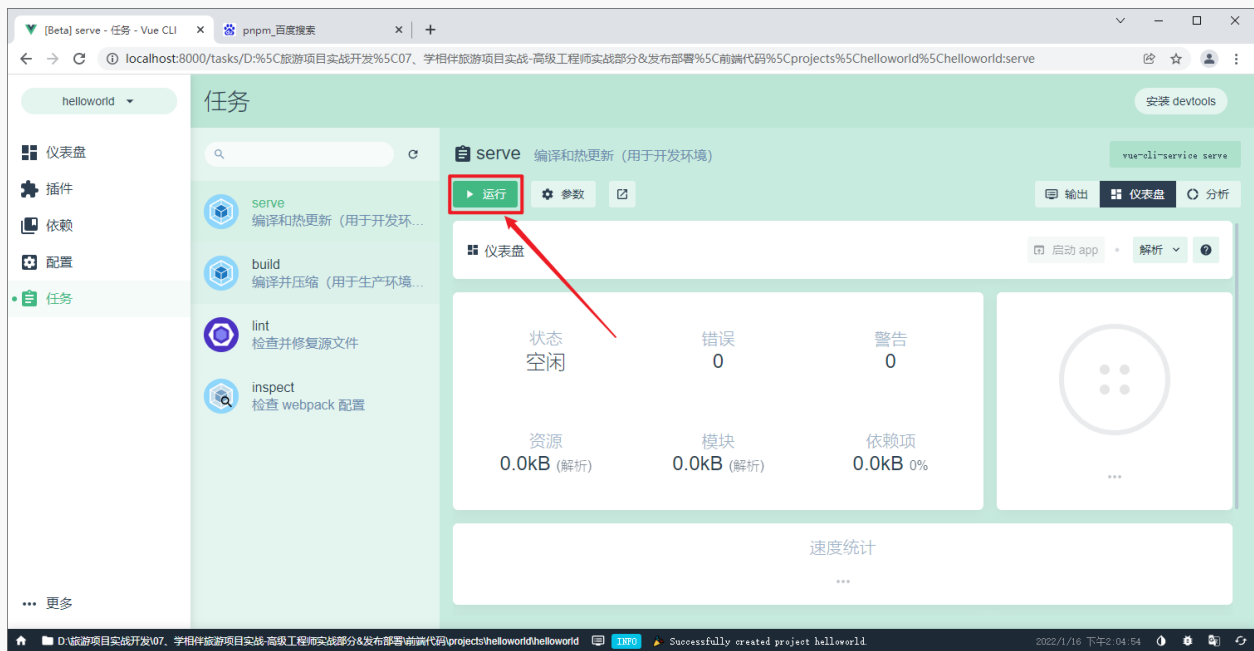
上述命令会打开一个新闻窗口，并以图形化界面将您引导至创建项目的流程。

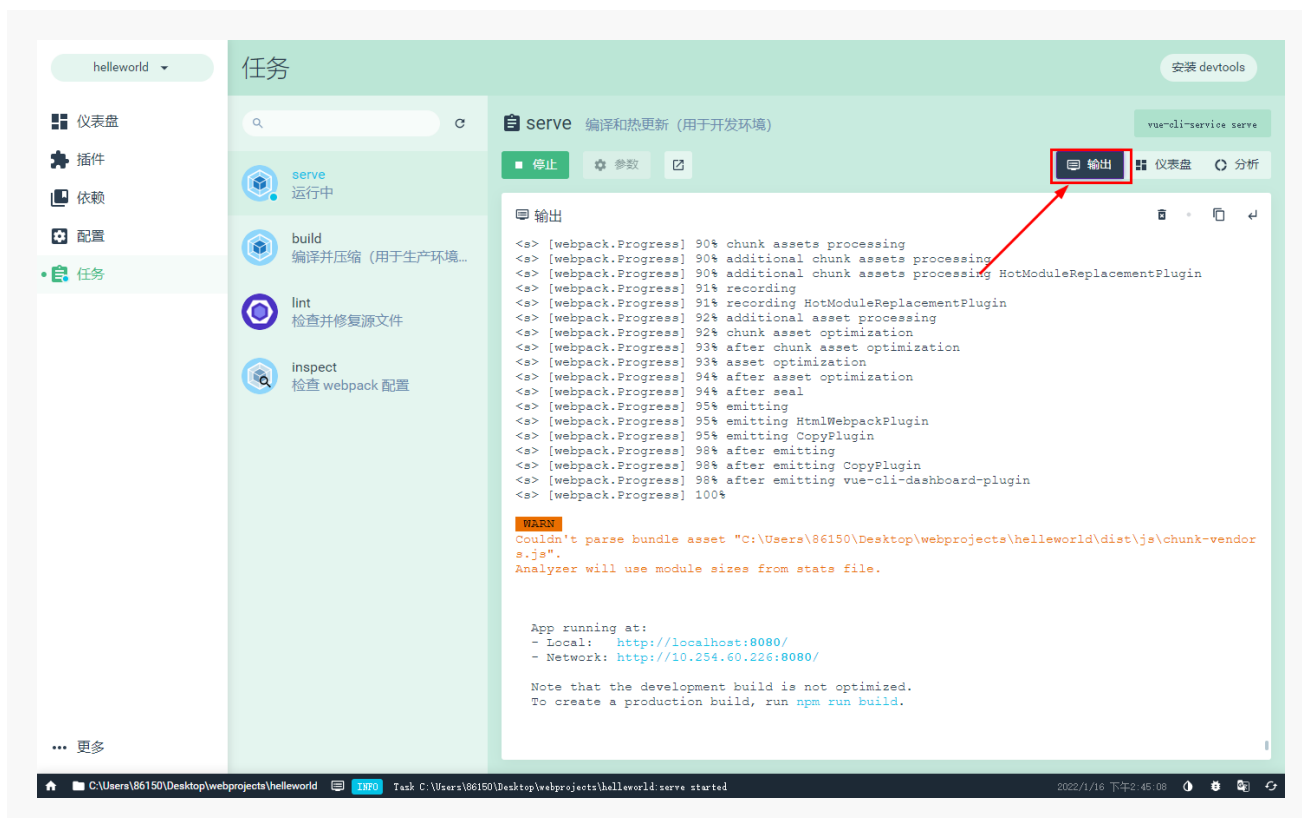




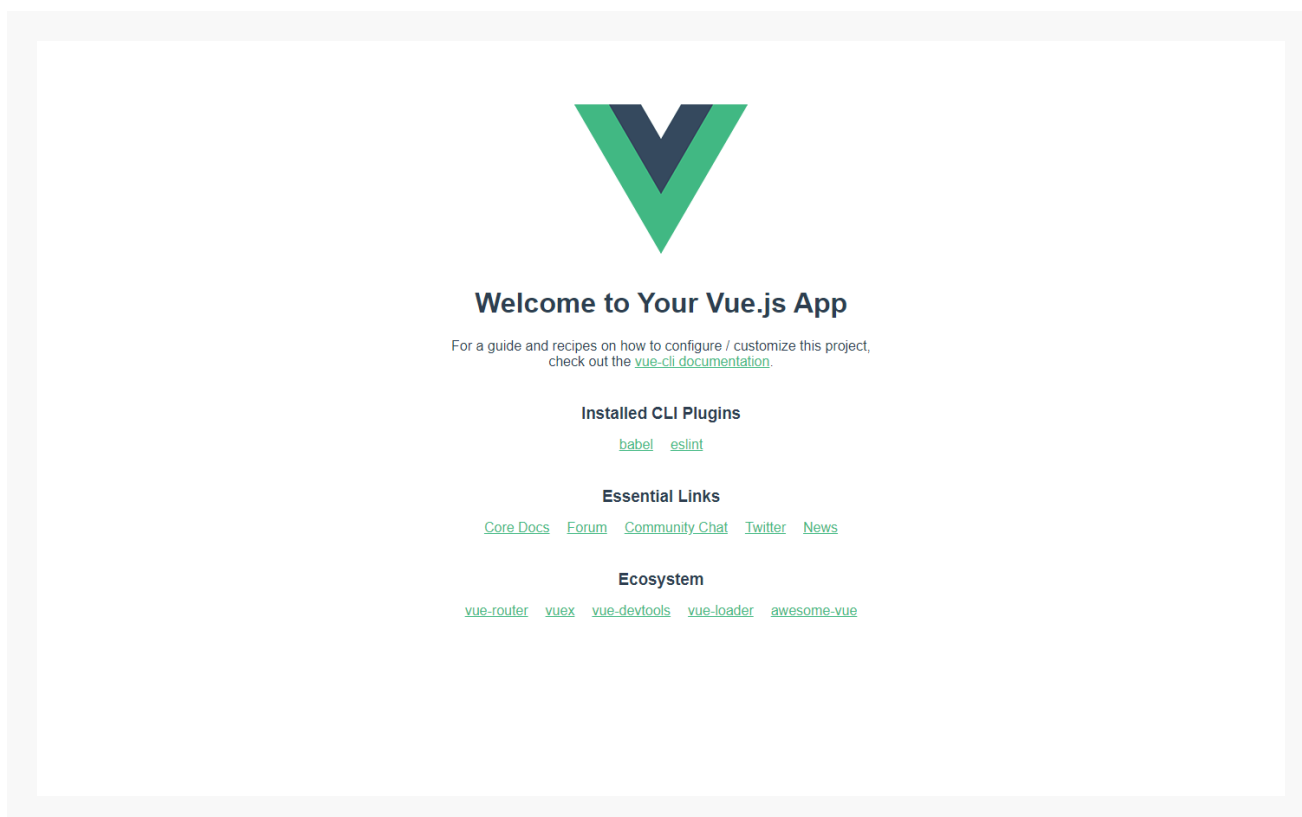
如果这里构建失败，可以尝试选择回npm或者yarn。







浏览器输入: <http://localhost:8080/>



04、牵涉技术栈的说明

react / vue

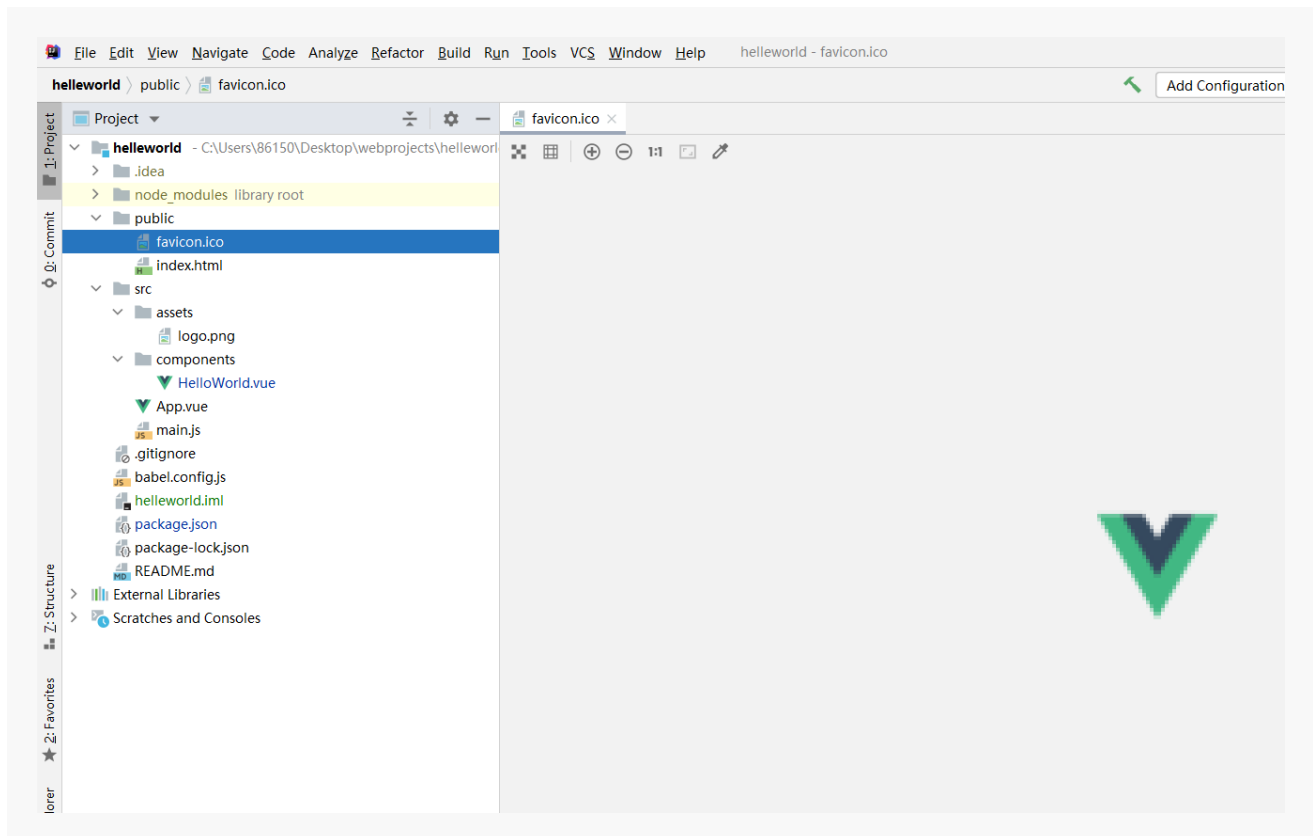
名称	说明
Babel	转码器，用于将ES6语法转换成ES5代码，从而在现有的环境下执行。
TypeScript	Typescript是Javascript的超集，主要提供了类型系统和对ES6的支持，TS是由微软开发出来的开源编程语言，它可以编译成纯Javascript。编译出来的Javascript可以运行在任何浏览器上。
PWA	支持渐进式的Web应用程序
Router	路由管理
Vuex	状态管理
Css Pre-processor	css预处理器，如Less ,sass
Lint/Formatter	代码风格检查和格式校验
Unit Testing	单元测试
E2E Testing	端到端测试

05、开发工具

vscode或者idea都可以。如果使用idea建议安装vue插件。

- vscode
- hbuilder
- idea

- webstorm



06、项目的骨架结构

- 1 **node_modules** 项目依赖模块，开发的时候使用，打包的时候会自动的过滤一些。
- 2 **public** ---该目录下的文件不会被webpack编译压缩处理，引用的第三方库的js文件可以放这里
- 3 **favicon.ico** 图标文件
- 4 **index.html** 项目的主页面
- 5 **src** 项目代码的主目录
- 6 **assets** 存放项目中的静态资源，如css，图片等。
- 7 **logo.png** logo图片
- 8 **components** 组件存放目录
- 9 **helloworld.vue** vuecli默认创建的helloworld组件
- 10 **App.vue** 项目根组件
- 11 **main.js** 程序入口js文件，加载各种公共组件和所需用到的插件。
- 12 **.gitignore** 如果配置了git提交项目代码时忽略那些文件和文件夹进行提交

- 13 `babel.config.js` babel使用的配置文件
- 14 `package.json` npm的配置文件，其中设定了脚本和项目依赖的各种库。
- 15 `package-lock.json` 用于锁定项目实际安装的各个npm包的具体来源和版本号
- 16 `README.md` 项目说明文件

核心文件说明：

App.vue

```
1 <template> <div id="app">        <!--插件的引入-->
    <HelloWorld msg="welcome to Your Vue.js App"/> </div>
</template><script>// 导入插件import HelloWorld from
'./components/HelloWorld.vue'export default {  name:
'App',  components: {    // 插件的注册    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>
```

main.js

```
1 import Vue from 'vue'import App from
'./App.vue'Vue.config.productionTip = falsenew Vue({
  render: h => h(App),}).$mount('#app')
```

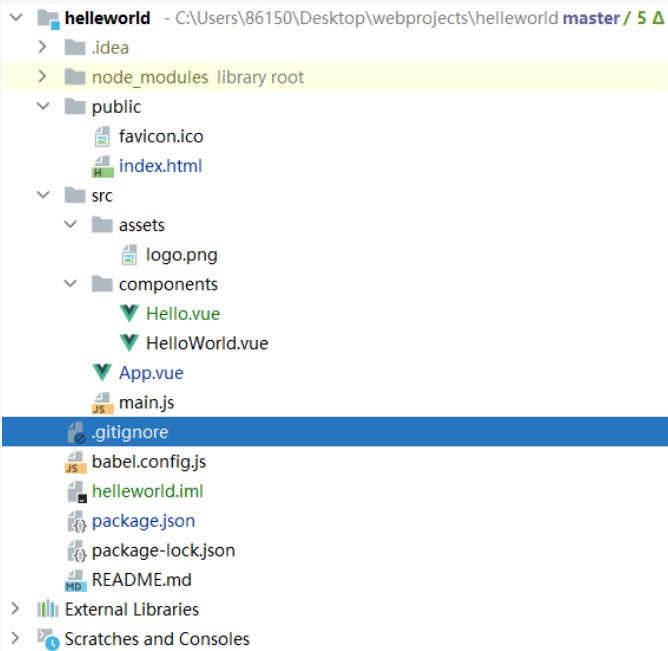
`main.js`是程序的入口js文件。该文件的主要作用用于加载公共组件和项目所需用到的各种插件。

index.html

```
1 <!DOCTYPE html>
2 <html lang="">
3   <head>
4     <meta charset="utf-8">
5     <meta http-equiv="X-UA-Compatible"
6 content="IE=edge">
7     <meta name="viewport" content="width=device-
8 width,initial-scale=1.0">
9     <link rel="icon" href="<%= BASE_URL
10 %>favicon.ico">
11     <title><%= htmlWebpackPlugin.options.title %>
12 </title>
13   </head>
14   <body>
15     <noscript>
16       <strong>We're sorry but <%=
17 htmlWebpackPlugin.options.title %> doesn't work
18 properly without JavaScript enabled. Please enable it
19 to continue.</strong>
20     </noscript>
21     <div><%= JSON.stringify(htmlWebpackPlugin.options)
22 %></div>
23     <div id="app"></div>
24   </body>
25 </html>
```

项目的主页面文件，用于引入其他的js/css的页面，也是编译的主模板页面。后续会明白它具体的用处。

07、编写一个Hello组件



在项目的components目录下新建一个Hello.vue的组件，如下：

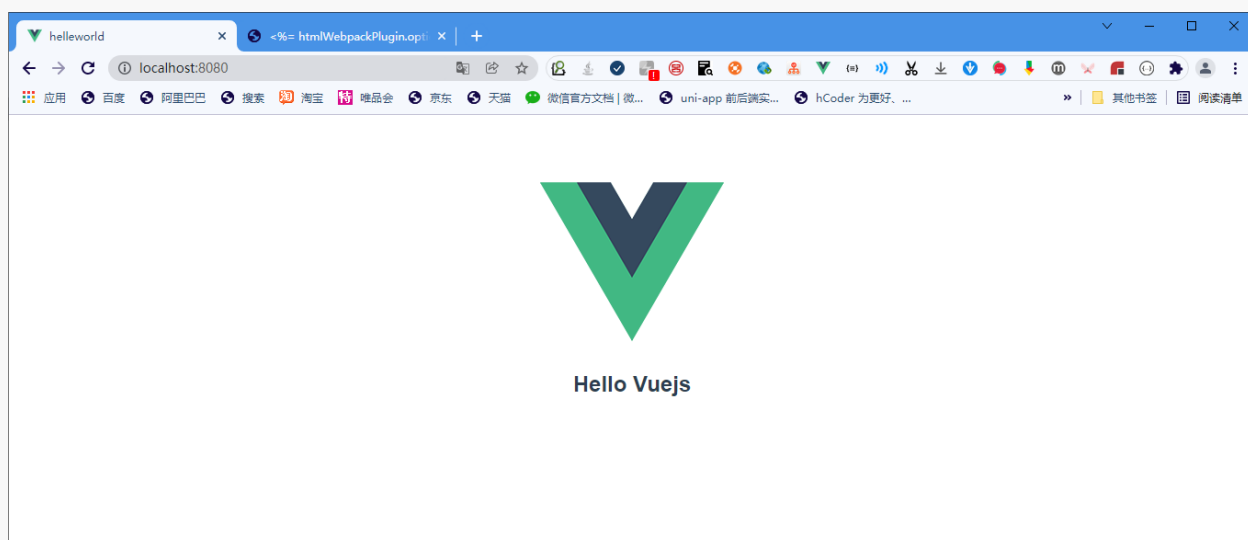
```
1 <template>
2   <h1>{{message}}</h1>
3 </template>
4
5 <script>
6   export default {
7     name: "Hello",
8     data(){
9       return {
10         message:"Hello vuejs"
11       }
12     }
13   }
14 </script>
15
16 <style scoped>
17
```

在App.vue中进行引入即可

```
1 <template> <div id="app">  <Hello></Hello> </div>
</template><script>import Hello from
"@/components/Hello";export default { name: 'App',
  components: { Hello }}</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>
```

- 导入的语句中是使用@符号标识src目录，该符号用于简化路径的访问。Hello组件没有写扩展名，这没有问题，项目中内置的webpack能够自动添加后缀.vue。

测试效果如下：



08、packjson.json

这是一个JSON格式的npm配置文件，定义了项目所需要的各种模块，以及项目的配置信息，在项目开发中经常会需要修改文件的配置内容，所以这里单独对这个文件说明一下，代码如下所示：

类似于java: pom.xml文件：

```
1 {  "name": "helloworld", 项目的名称  "version": "0.1.0",
    项目的版本  "private": true, 项目是否是私有项目  "scripts":
    {  值是一个对象，其中指定了项目生命周期各个环节需要执行的命令
      "serve": "vue-cli-service serve", 执行npm run serve 运行
      项目      "build": "vue-cli-service build", 执行npm run
      build 构建项目      "lint": "vue-cli-service lint"  执行npm
      run lint运行eslint验证并格式化代码  },  "dependencies": {
      //配置项目依赖的模块列表，key是模块的名称，value是版本范围
      "core-js": "^3.6.5",      "vue": "^2.6.11"  },
      "devDependencies": {  // 这里的依赖是用于开发环境的，不发布到生
      产环境。      "@vue/cli-plugin-babel": "~4.5.0",
      "@vue/cli-plugin-eslint": "~4.5.0",      "@vue/cli-
      service": "~4.5.0",      "babel-eslint": "^10.1.0",
      "eslint": "^6.7.2",      "eslint-plugin-vue": "^6.2.2",
      "vue-template-compiler": "^2.6.11"  },  "eslintConfig":
      {      "root": true,      "env": {      "node": true      },
      "extends": [      "plugin:vue/essential",
      "eslint:recommended"      ],      "parserOptions": {
      "parser": "babel-eslint"      },      "rules": {}  },
      "browserslist": [      "> 1%",      "last 2 versions",
      "not dead"  ]}
```

在使用npm安装依赖的模块的时候，可以依据模块是否在生成环境下使用从而选择-S（--save）代表生产环境，或者-D即（--save-dev 开发环境）比如：

```
1 // 写入到 dependenciesnpm install element-ui -S等同于npm
  install element-ui --save// 写入到 devDependenciesnpm
  install element-ui -D等同于npm install element-ui --
  save-dev
```

安装以后会在dependencies中写入依赖项，在项目打包发布的时候，dependencies会写入依赖项也会一起打包。

什么时候加 -D 还是 -S

```
1 - 业务是否需要：-S - 打包编译的，测试，检查：-D
```

如果某个模块只是在开发环境中使用，则可以使用-D参数进行安装，安装完毕以后会写入devDependencies,而在devDependencies中的依赖项，在项目打包发布时并不会一起打包。

在发布代码时，项目下的`node_modules`文件夹都不会发布，那么在下载别人代码后，怎么样安装依赖呢？这时可以在项目的根路径下执行`npm install`命令接口。该命令会根据`package.json`文件下载所需要的依赖

最终webpack打包会生成app.js和/js/chunk-vendors.js

这两个文件生成的内容依据是什么呢？以package.json中的：dependencies进行融合打包。

📖 02、旅游项目后台pug搭建全过程 - VueRouter-基础

🧠 思想参考

参考：struts2、springmvc

🧠 开发前端的三部曲

- 定义单页面SPA ----XXX.vue
- 配置单页面的路由 -- router/index.js
- 单页面SPA业务的Axios的对接
 - service 后端接口的业务对接方法
 - 进行导入单页中

🧠 01、概述

官网：<https://router.vuejs.org/zh/>

Vue Router 是 [Vue.js \(opens new window\)](#)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
- 自定义的滚动条行为

其实和java框架： *springmvc*差不多类同

02、感受前端路由

1: 为项目安装vue-router

```
1 npm install vue-router@next --save
```

提示：如果你使用的 vue3版本，就必须使用 `npm install vue-router@next --save`，而如果你安装的项目Vue2.x的直接 `npm install vue-router --save` 即可。

2: package.json的变化


```
1 { "name": "helloworld", "version": "0.1.0",  
  "private": true, "scripts": {    "dev": "vue-cli-  
service serve",    "build": "vue-cli-service build",  
  "lint": "vue-cli-service lint"  }, "dependencies": {  
    "core-js": "^3.6.5",    "vue": "^2.6.11",    "vue-  
router": "^4.0.12" // 增加了路由vue-router  },  
  "devDependencies": {    "@vue/cli-plugin-babel":  
    "~4.5.0",    "@vue/cli-plugin-eslint": "~4.5.0",  
    "@vue/cli-service": "~4.5.0",    "babel-eslint":  
    "^10.1.0",    "eslint": "^6.7.2",    "eslint-plugin-  
vue": "^6.2.2",    "vue-template-compiler": "^2.6.11"  
  }, "eslintConfig": {    "root": true,    "env": {  
    "node": true    },    "extends": [  
    "plugin:vue/essential",    "eslint:recommended"    ],  
    "parserOptions": {    "parser": "babel-eslint"  
  },    "rules": {}  }, "browserslist": [    "> 1%",  
    "last 2 versions",    "not dead"  ]}
```

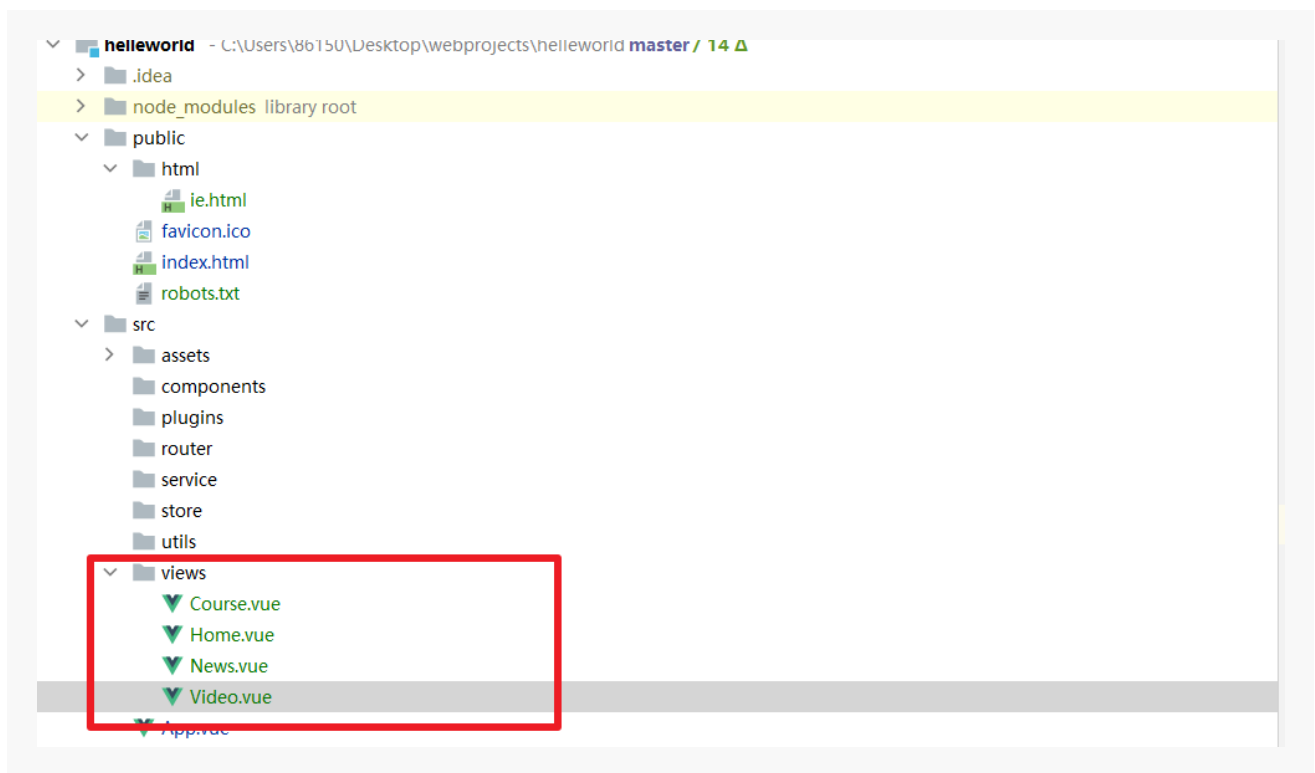
3: 在App.vue中设置导航链接和组件渲染的位置，修改其模板内容。如下:

```
1 <template>  
2   <div>  
3     <header>  
4       <router-link to="/">首页</router-link>  
5       <router-link to="/news">新闻</router-link>  
6       <router-link to="/book">图书</router-link>  
7       <router-link to="/course">课程</router-link>  
8       <router-link to="/video">视频</router-link>  
9     </header>  
10    <article>  
11      <router-view></router-view>  
12    </article>  
13  </div>
```

```
14 </template>
15
16 <script>
17 export default {
18   name: 'App',
19   components: {
20
21   }
22 }
23
24 </script>
25
26 <style>
27 #app {
28   font-family: Avenir, Helvetica, Arial, sans-serif;
29   -webkit-font-smoothing: antialiased;
30   -moz-osx-font-smoothing: grayscale;
31   text-align: center;
32   color: #2c3e50;
33   margin-top: 60px;
34 }
35 </style>
36
```

3: 定义新闻，课程，视频，首页的路由页面组件

在项目的src目录下新建一个views页面，存放视图页面。如下：



Home.vue

```
1 <template>
2   <h1>我是首页</h1>
3 </template>
4
5 <script>
6 export default {
7   name: "Home.vue"
8 }
9 </script>
10
11 <style scoped>
12
13 </style>
```

Book.vue

```
1 <template>
2   <h1>我是课程</h1>
3 </template>
4
5 <script>
6 export default {
7   name: "Course.vue"
8 }
9 </script>
10
11 <style scoped>
12
13 </style>
```

News.vue

```
1 <template>
2   <h1>我是新闻</h1>
3 </template>
4
5 <script>
6 export default {
7   name: "News.vue"
8 }
9 </script>
10
11 <style scoped>
12
13 </style>
```

Video.vue

```
1 <template> <h1>我是视频</h1></template><script>export
  default { name: "Video.vue"</script><style scoped>
  </style>
```

4: 配置路由规则

在src新建一个目录router，定义index.js，配置路由规则信息。这也是项目中经常使用的方式。编辑该文件即可

```
1 // 1: 导入路由// import {createRouter,
  createWebHashHistory} from 'vue-router'import
  {createRouter, createWebHistory} from 'vue-router'// 2:
  导入页面模板import Home from '@views/Home'import News
  from '@views/News'import Video from
  '@views/Video'import Course from '@views/Course'//3:
  创建路由器对象，将模板全部进行路由匹配和注册const router =
  createRouter({    // HASH访问模式，
  http://localhost:8080/#news    //history:
  createWebHashHistory(),    // URL访问模式，
  http://localhost:8080/news    history:
  createWebHistory(),    routes: [    // 设置欢迎页面，
  就path设置成 "/"即可    {path: '/',component: Home},
    {path: '/news',component: News},    {path:
  '/course',component: Course},    {path:
  '/video',component: Video},    ]})// 4: 导出模板router模
  块即可export default router
```

5、在程序的入口main.js文件中，使用router实例让整个应用都有路由功能，代码如下：

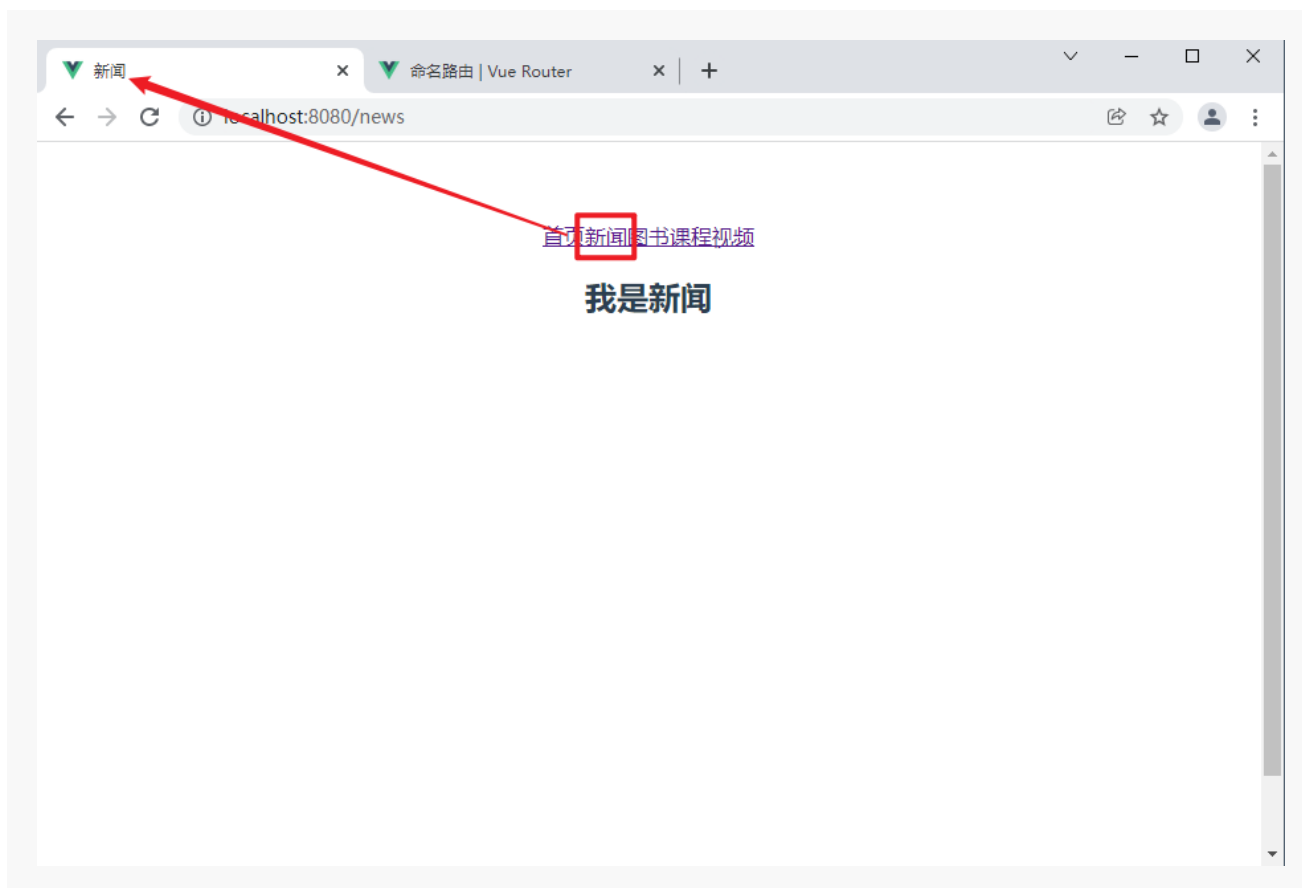
```
1 import { createApp } from 'vue'import App from
  './App.vue'import router from
  './router'createApp(App).use(router).mount('#app')
```

至此前端路由就全部配置完毕了。打开终端窗口，输入：`npm run serve`命令。运行项目体验单页应用的前端路由即可。

03、**meta** 命名路由

给没个路由取一个标题，使用：**meta**即可，如下：

```
1 // 1: 导入路由//import {createRouter,
createWebHashHistory} from 'vue-router'import
{createRouter, createWebHistory} from 'vue-router'// 2:
导入页面模板// import News from '@views/News'// import
Book from '@views/Book'// import Video from
'@views/Video'// import Course from
'@views/Course'//3: 创建路由器对象，将模板全部进行路由匹配和
注册const router = createRouter({  // HASH访问模式，
http://localhost:8080/#news  //history:
createWebHashHistory(),  // URL访问模式，
http://localhost:8080/news  history:
createWebHistory(),  routes: [  // 设置欢迎页面，
就path设置成 "/"即可  {path: '/', meta:{title:"首
页"},component: () => import("@views/Home")},
{path: '/news', meta:{title:"新闻"},component: () =>
import("@views/News")},  {path: '/book', meta:
{title:"书籍"},component: () => import("@views/Book")},
    {path: '/course', meta:{title:"课程"},component:
() => import("@views/Course")},  {path:
'/video', meta:{title:"视频"},component: () =>
import("@views/Video")},  ]})router.afterEach(to =>
{  document.title = to.meta.title;})// 4: 导出模板
router模块即可export default router
```



🧠 03、动态路由匹配-params参数

在实际的开发时，经常需要把匹配某种模式的路由映射到同一个组件上，比如：

- 查看用户明细
- 查看文章明细
- 。 。 。 。 。 。

它们的需求都是：根据不同的id查询对应的数据，但是模板是一样的。都是使用相同的组件来渲染。这个可以使用路径中的动态段，称之为参数（param）。格式如下：

- 参数使用冒号（:）表示，
 - 定义路由阶段：比如： `/book/:id,/user/detail/:id` 等。

- **使用阶段：**即你具体的时候传入：`/book/1`、`/book2`、`/book/foo` 都将映射到相同的路由上。
- 当匹配到一个路由时，参数的值被保存到`this.$route.params`中，`this.route`代表当前那个路由对象中，可以在组件内使用。

代码如下：

Book.vue

```
1 <template>    <h1>我是书籍</h1>    <ul>        <li v-for="
  (book,index) in bookList" :key="index">            <p>
        {{book.id}}===={{book.title}}====
        <router-link :to="'/book/'+book.id">查看明细1</router-
  link>            <a :href="'/book/'+book.id">查看明细
2</a>            </p>        </li>    </ul></template>
<script>  import  Books from '@service/Books'  export
default {    name: "Home",    data(){        return {
    bookList:Books        }    } }</script><style scoped>
</style>
```

2、BookDetail.vue

```
1 <template>  <div>图书ID: {{ $route.params.id }}</div>
</template><script>export default {  data(){    return
{    }  }}</script>
```

3、在router/index.js

```

1 import {createRouter, createWebHistory} from 'vue-router'
import News from '@components/News'
import Books from '@components/Books'
import Videos from '@components/Videos'
import Book from '@components/Book'
import Login from '@components/Login'
const router = createRouter({
  //history: createWebHashHistory(), history:
  createWebHistory(), routes: [
    { path: '/book/:id', name: 'book', meta: {
      title: '图书' }, components: {bookDetail: Book}, }
  ]})
export default router

```

在同一路由中可以有多动态段如下

模式	匹配路径	\$ROUTE.PARAMS
/user/:username	/user/yykk	{“username”:”yykk”}
/user/:username/:id	/user/yykk/123	{“username”:”yykk”,id:123}

除了 `$route.params` 还提供了其他的有用信息，如： `$route.query`

04、查询参数路由匹配-query参数

URL中带有查询参数的形式为： `/book?id=1` ,这在传统的web应用程序中是非常场景的。在单页程序开发中，也支持路径中的查询参数。只不过参数放在的是： `$route.query` 中。

1: App.vue

```
1 <template> <p> <router-link to="/book?id=1">图书
  1</router-link> <router-link to="/book?id=2">图书
  2</router-link> </p> <router-view></router-view>
  </template>
```

2: Book.vue

```
1 <template> <div>图书ID: {{ $route.query.id }}</div>
  </template><script>export default { data(){ return
    { } }}</script>
```

3: router/index.js

```
1 import {createRouter, createWebHistory} from 'vue-
  router'import News from '@components/News'import Books
  from '@components/Books'import Videos from
  '@components/Videos'import Book from
  '@components/Book'import Login from
  '@components/Login'const router = createRouter({
  //history: createWebHashHistory(), history:
  createWebHistory(), routes: [ { path: '/book',
    name: 'book', meta: { title: '图书'
  }, components: {bookDetail: Book, } ]})export
  default router
```

05、路由嵌套

在实际的而应用场景中，一个界面的UI通常是由多层嵌套组件组合而成。URL中的各段也是按某种结果对应嵌套的各层组件。如下图：

```

1 /user/foo/profile /user/foo/posts+-
  -----+ +-----
+| User          | | User
|| +-----+ | | +-----
+ || | Profile   | | +-----> | | Posts
  || |           | |           | |
    || +-----+ | |           | | +-----
-----+ | +-----+ +-----
-----+

```

借助 **vue-router**，使用嵌套路由配置，就可以很简单地表达这种关系。

1: 定义service/Books.js

```

1 function loadBooks(){ return [
  {id:1,title:"Java无难事",author:"yykk1"},
  {id:2,title:"C++深入探索",author:"yykk2"},
  {id:3,title:"Servlet/jsp深入详见",author:"yykk3"},
  {id:4,title:"Javascript面向对象编程",author:"yykk4"},
  ];}var books = loadBooks();export default books;

```

2: 修改Book.vue

```

1 <template>
2   <h1>我是书籍-----基于params</h1>
3   <ul>
4     <li v-for="(book,index) in bookList" :key="index">
5       <p>

```

```
6         <router-link :to="'/book/'+book.id">{{
book.title }}</router-link>
7         <a :href="'/book/'+book.id">查看明细2</a>
8     </p>
9 </li>
10 </ul>
11 <h1>我是书籍-----基于query</h1>
12 <ul>
13     <li v-for="(book,index) in bookList" :key="index">
14         <p>
15             <router-link :to="'/book2?id='+book.id">{{
book.title }}</router-link>
16             <a :href="'/book2?id='+book.id">查看明细2</a>
17         </p>
18     </li>
19 </ul>
20
21 <!--子路由在这里渲染-->
22 <router-view></router-view>
23 </template>
24
25 <script>
26 import Books from '@service/Books'
27
28 export default {
29     name: "Home",
30     data() {
31         return {
32             bookList: Books
33         }
34     }
35 }
36 </script>
37
```

```
38 <style scoped>
39
40 </style>
```

3: 定义明细BookDetail.vue

```
1 <template>
2   <h1>你查看的书籍ID---params是: {{$route.params.id}}
   </h1>
3   <h1>你查看的书籍ID---query是: {{$route.query.id}}
   </h1>
4   <h1>你查看的书籍ID---封装的是: {{getParameter("id")}}
   </h1>
5 </template>
6
7 <script>
8   export default {
9     name: "BooksDetail",
10    created() {
11      console.log(this.getParameter("id"))
12    },
13    methods:{
14      getParameter(field){
15        return this.$route.query[field] ||
16        this.$route.params[field];
17      }
18    }
19  </script>
20
21 <style scoped>
22
23 </style>
```

4: 定义路由router/index.js

```
1 // 1: 导入路由//import {createRouter,
  createWebHashHistory} from 'vue-router'import
  {createRouter, createWebHistory} from 'vue-router'//3:
  创建路由器对象，将模板全部进行路由匹配和注册const router =
  createRouter({    // HASH访问模式，
  http://localhost:8080/#news    //history:
  createWebHashHistory(),    // URL访问模式，
  http://localhost:8080/news    history:
  createWebHistory(),    routes: [    {
  path: '/book', meta: {title: "书籍"}, component: () =>
  import("@/views/Book"), children: [
  {path: '/book/:id', meta: {title: "书籍明细1"},
  component: () => import("@/views/BookDetail")},
    {path: '/book2', meta: {title: "书籍明细2"},
  component: () => import("@/views/BookDetail")}
  ]    },    ]});router.afterEach(to => {
  document.title = to.meta.title;})// 4: 导出模板router模块
  即可export default router
```

5: 明细查询BookDetail.vue

```
1 <template> <h1>你查看的书籍ID---params是: {{
$route.params.id }}</h1> <h1>你查看的书籍ID---query是: {{
$route.query.id }}</h1> <h1>你查看的书籍ID---封装的是: {{
getParameter("id") }}</h1> <p>图书名称是:{{ book.title
}}</p> <p>图书名称是:{{ book.desc }}</p></template>
<script>import Books from "@service/Books";export
default { name: "BooksDetail", data() { return
{book: {}} }, created() { var bookid =
this.getParameter("id"); this.book = Books.find(item
=> item.id == bookid); }, methods: {
getParameter(field) { return
this.$route.query[field] || this.$route.params[field];
} }}</script><style scoped></style>
```

6: 小结

1、要在嵌套的出口（即：Books组件中的）中渲染组件，需要在routes选择的匹配中使用children选项。children选项只是路由配置对象的另一个数组，如同 routes本身一样，因此，可以根据需要继续嵌套路由。

2、子路由只能在父页面去找<router-view>找不到就终止渲染。

06、name命名路由

有时候，通过一个名称来标识一个路由显得更方便一些，特别是在链接一个路由，或者是执行一些跳转的时候。你可以在创建 Router 实例的时候，在 routes 配置中给某个路由设置名称。


```
1 const router = new VueRouter({ routes: [    {  
  path: '/user/:userId',      name: 'user',  
  component: User    }  ]})
```

要链接到一个命名路由，可以给 `router-link` 的 `to` 属性传一个对象：

```
1 <router-link :to="{ name: 'user', params: { userId: 123  
  }}">User</router-link><router-link :to="{ name: 'user',  
  query: { userId: 123 }}">User</router-link>
```

这跟代码调用 `router.push()` 是一回事：

```
1 router.push({ name: 'user', params: { userId: 123 }  
  })router.push({ name: 'user', query: { userId: 123 } })
```

这两种方式都会把路由导航到 `/user/123` 和 `/user?userId=123` 路径。

07、命名视图（父子路由）

有时候想同时 (同级) 展示多个视图，而不是嵌套展示，例如创建一个布局，有 `sidebar` (侧导航) 和 `main` (主内容) 两个视图，这个时候命名视图就派上用场了。你可以在界面中拥有多个单独命名的视图，而不是只有一个单独的出口。如果 `router-view` 没有设置名字，那么默认为 `default`。

```
1 <router-view class="view header" name="header">  
  </router-view><router-view class="view slider"  
  name="slider"></router-view><router-view class="view  
  main"></router-view>
```

1、App.vue


```
1 <template> <p> <router-view></router-view>
  <router-view name="bookDetail"></router-view> </p>
  </template>
```

2、router/index.js

```
1 { path: '/book/:id', name: 'book', meta: {
  title: '图书' }, components: {bookDetail:
  Book},},
```

08、程式化导航（JSAPI）的方式

除了使用 `<router-link>` 创建 a 标签来定义导航链接，我们还可以借助 router 的实例方法，通过编写代码来实现。

 `#router.push(location, onComplete?, onAbort?)`

注意：在 **Vue** 实例内部，你可以通过 `$router` 访问路由实例。因此你可以调用 `this.$router.push`。

想要导航到不同的 URL，则使用 `router.push` 方法。这个方法会向 `history` 栈添加一个新的记录，所以，当用户点击浏览器后退按钮时，则回到之前的 URL。

当你点击 `<router-link>` 时，这个方法会在内部调用，所以说，点击 `<router-link :to="...">` 等同于调用 `router.push(...)`。

声明式	编程式
<code><router-link :to="..."></code>	<code>router.push(...)</code>

该方法的参数可以是一个字符串路径，或者一个描述地址的对象。例如：

```
1 // 字符串router.push('home')// 对象router.push({ path:
  'home' })// 命名的路由router.push({ name: 'user', params:
  { userId: '123' }})// 带查询参数，变成 /register?
  plan=privaterouter.push({ path: 'register', query: {
  plan: 'private' }})
```

注意：如果提供了 `path`，`params` 会被忽略，上述例子中的 `query` 并不属于这种情况。取而代之的是下面例子的做法，你需要提供路由的 `name` 或手写完整的带有参数的 `path`：

```
1 const userId = '123'router.push({ name: 'user', params:
  { userId }}) // -> /user/123router.push({ path:
  `/user/${userId}` }) // -> /user/123// 这里的 params 不生
  效router.push({ path: '/user', params: { userId }}) // -
  > /user
```

同样的规则也适用于 `router-link` 组件的 `to` 属性。

在 2.2.0+, 可选的在 `router.push` 或 `router.replace` 中提供 `onComplete` 和 `onAbort` 回调作为第二个和第三个参数。这些回调将会在导航成功完成 (在所有的异步钩子被解析之后) 或终止 (导航到相同的路由、或在当前导航完成之前导航到另一个不同的路由) 的时候进行相应的调用。在 3.1.0+, 可以省略第二个和第三个参数, 此时如果支持 `Promise`, `router.push` 或 `router.replace` 将返回一个 `Promise`。

注意: 如果目的地和当前路由相同, 只有参数发生了改变 (比如从一个用户资料到另一个 `/users/1` -> `/users/2`), 你需要使用 `beforeRouteUpdate` 来响应这个变化 (比如抓取用户信息)。

`router.replace(location, onComplete?, onAbort?)`

跟 `router.push` 很像, 唯一的不同就是, 它不会向 `history` 添加新记录, 而是跟它的方法名一样 —— 替换掉当前的 `history` 记录。

声明式

```
<router-link :to="..." replace>
```

编程式

```
router.replace(...)
```

```
1 // 对象
2 router.push({ path: 'home', replace: true })
3 #相当于
4 router.replace({ path: 'home' })
```

`router.go(n)`

这个方法的参数是一个整数, 意思是在 `history` 记录中向前或者后退多少步, 类似 `window.history.go(n)`。

例子

```
1 // 在浏览器记录中前进一步，等同于 history.forward()
2 router.go(1)
3
4 // 后退一步记录，等同于 history.back()
5 router.go(-1)
6
7 // 前进 3 步记录
8 router.go(3)
9
10 // 如果 history 记录不够用，那就默默地失败呗
11 router.go(-100)
12 router.go(100)
```

#操作 History

你也许注意到 `router.push`、`router.replace` 和 `router.go` 跟 `window.history.pushState`、`window.history.replaceState` 和 `window.history.go (opens new window)` 好像，实际上它们确实是效仿 `window.history` API 的。

因此，如果你已经熟悉 [Browser History APIs \(opens new window\)](#)，那么在 Vue Router 中操作 history 就是超级简单的。

还有值得提及的，Vue Router 的导航方法 (`push`、`replace`、`go`) 在各类路由模式 (`history`、`hash` 和 `abstract`) 下表现一致。

09、路由组件传参

在组件中使用 `$route` 会使之与其对应路由形成高度耦合，从而使组件只能在某些特定的 URL 上使用，限制了其灵活性。

使用 `props` 将组件和路由解耦：

1: 定义BookDetail2.vue

```
1 <template>    <div>参数是: {{id}} === {{title}}</div>
  </template><script>export default {  props:{    id:
    {type:Number,default:0},    title:
    {type:String,default:"ok"},  },  name: "User.vue"}
  </script><style scoped></style>
```

2: router/index.js路由注册

```
1 {path: '/book', meta: {title: "书籍"}, component: () =>
  import("@/views/Book"), children: [    {path:
    '/bookdetail/:id/:title',name:"bookdetail3", meta:
    {title: "书籍明细3"}, component: () =>
    import("@/views/BookDetail2"),props:true}}],
```

3: 在Book.vue定义

```
1 <router-link
  :to="`/bookdetail/${book.id}/${book.title}`">查看明细
  3</router-link>
```

```

1 <template> <h1>我是书籍-----基于params</h1> <ul> <li
  v-for="(book,index) in bookList" :key="index"> <p>
    <router-link :to="'/book/'+book.id">{{
book.title }}</router-link> <a
  :href="'/book/'+book.id">查看明细1</a> <router-
link :to="{ name: 'bookdetail1', params: { id: book.id
  }}">查看明细2</router-link> <router-link
  :to="`/bookdetail/${book.id}/${book.title}`">查看明细
3</router-link> </p> </li> </ul> <h1>我是书籍--
---基于query</h1> <ul> <li v-for="(book,index) in
bookList" :key="index"> <p> <router-link
  :to="'/book2?id='+book.id">{{ book.title }}</router-
link> <a :href="'/book2?id='+book.id">查看明细
2</a> <router-link :to="{ name: 'bookdetail2',
query: { id: book.id }}">查看明细2</router-link>
  <router-link
  :to="`/bookdetail/${book.id}/${book.title}`">查看明细
3</router-link> </p> </li> </ul> <hr> <!--子
路由在这里渲染--> <router-view></router-view></template>
<script>import Books from '@service/Books'export
default { name: "Home", data() { return {
  bookList: Books    } }}</script><style scoped>
</style>

```

可以很轻松的看到，会自动把id和title在内部传递给props。而不需要其他的获取和操作。

10、HTML5 History 模式

`vue-router` 默认 hash 模式 —— 使用 URL 的 hash 来模拟一个完整的 URL，于是当 URL 改变时，页面不会重新加载。

如果不想要很丑的 hash，我们可以用路由的 **history** 模式，这种模式充分利用 `history.pushState` API 来完成 URL 跳转而无须重新加载页面。

```
1 const router = new VueRouter({ mode: 'history',  
  routes: [...]}])
```

当你使用 history 模式时，URL 就像正常的 url，例如 `http://yoursite.com/user/id`，也好看！

不过这种模式要玩好，还需要后台配置支持。因为我们的应用是个单页客户端应用，如果后台没有正确的配置，当用户在浏览器直接访问 `http://oursite.com/user/id` 就会返回 404，这就不好看了。

所以呢，你要在服务端增加一个覆盖所有情况的候选资源：如果 URL 匹配不到任何静态资源，则应该返回同一个 `index.html` 页面，这个页面就是你 app 依赖的页面。

```
1 import {createRouter, createWebHistory} from 'vue-  
  router'  
2 //import Home from '@components/Home'  
3 import News from '@components/News'  
4 import Books from '@components/Books'  
5 import Videos from '@components/Videos'  
6 import Book from '@components/Book'  
7 import Login from '@components/Login'
```



```
8
9 const router = createRouter({
10   //history: createWebHashHistory(),
11   history: createWebHistory(),
12   routes: [
13     {
14       path: '/',
15       redirect: {
16         name: 'news'
17       }
18     },
19     {
20       path: '/news',
21       name: 'news',
22       component: News,
23       meta: {
24         title: '新闻'
25       }
26     },
27     {
28       path: '/books',
29       name: 'books',
30       component: Books,
31       meta: {
32         title: '图书列表'
33       }
34     },
35     {
36       path: '/videos',
37       name: 'videos',
38       component: Videos,
39       meta: {
40         title: '视频',
41
```

```
42     }
43   },
44   {
45     path: '/book/:id',
46     name: 'book',
47     meta: {
48       title: '图书'
49     },
50     components: {bookDetail: Book},
51   },
52   {
53     path: '/login',
54     name: 'login',
55     component: Login,
56     meta: {
57       title: '登录'
58     }
59   }
60 ]
61 })
62
63 router.beforeEach(to => {
64   //判断目标路由是否是/login, 如果是, 则直接返回true
65   if(to.path == '/login'){
66     return true;
67   }
68   else{
69     //否则判断用户是否已经登录, 注意这里是字符串判断
70     if(sessionStorage.isAuth === "true"){
71       return true;
72     }
73     //如果用户访问的是受保护的资源, 且没有登录, 则跳转到登录页面
74     //并将当前路由的完整路径作为查询参数传给Login组件, 以便登录
    成功后返回先前的页面
```

```
75     else{
76         return {
77             path: '/login',
78             query: {redirect: to.fullPath}
79         }
80     }
81 }
82 })
83
84 router.afterEach(to => {
85     document.title = to.meta.title;
86 })
87
88 export default router
```

📖 03、旅游项目后台pug搭建全过程 - VueRouter-进阶

01、导航守卫

正如其名，`vue-router` 提供的导航守卫主要用来通过跳转或取消的方式守卫导航。有多种机会植入路由导航过程中：全局的, 单个路由独享的, 或者组件级的。

记住参数或查询的改变并不会触发进入/离开的导航守卫。你可以通过[观察 `\$route` 对象](#)来应对这些变化，或使用 `beforeRouteUpdate` 的组件内守卫。

02、全局前置守卫

全局守卫：分为：前置守卫、全局守卫和全局后置钩子。

02-01、全局前置守卫

当一个导航触发时，全局前置守卫按照创建的顺序调用，守卫可以是异步解析执行，此时导航在所有守卫解析完成之前一直处于挂起状态。全局前置守卫使用 `router.beforeEach` 注册一个全局前置守卫，代码如下：

```
1 const router = new VueRouter({ ...
  })router.beforeEach((to, from, next) => {      return
  false;})
```

除了返回`false`以取消导航外，还可以返回一个路由位置对象，这将导致路由重定向到另一个位置，如同正在调用`router.push()`方法一样，可以传递诸如`replace:true`或者`name:"home"`之类的选项。返回路由位置对象时，将删除当前导航，并使用相同的`from`创建一个新的导航。

02-02、全局解析守卫

全局解析守卫使用`router.beforeResolve()`注册，它和`router.beforeEach`类型，区别就在于，在导航被确认之前，在所有组件内守卫和异步路由组件被解析之后，解析守卫被调用。

```
1 router.beforeResolve ((to, from, next) => { })
```

02-03、全局后置钩子

```
1 router.afterEach((to, from, next) => {})
```

案例1：实现拦截登录

```
1 //1: 导入路由
2 import {createRouter, createWebHistory} from 'vue-
  router'
3
4 //2: 创建路由器对象，将模板全部进行路由匹配和注册
5 const router = createRouter({
6   history: createWebHistory(),
7   routes: [
8     {
9       path: "/",
10      name: "layout",
```

```
11         component: () =>
import('@layout/Layout'),
12         redirect: "/index",
13         //这里只是重定向浏览器地址，但是渲染还是在父级，
14         // 好处：就是可以打破不用和父同名默认是首页的规则
15         children: [
16             {
17                 path: "/index",
18                 name: "index",
19                 meta:{title:"首页"},
20                 component: () =>
import('@views/index')
21             },
22             {
23                 path: "/user",
24                 name: "user",
25                 meta:{title:"用户管
理",isAuth:true},
26                 component: () =>
import('@views/user/index')
27             },
28             {
29                 path: "/course",
30                 name: "course",
31                 meta:{title:"课程管理"},
32                 component: () =>
import('@views/course/index')
33             },
34             {
35                 path: "/order",
36                 name: "order",
37                 meta:{title:"订单管理"},
38
```

```

39             component: () =>
import('@views/order/index')
40         }
41
42     ]
43 },
44 {
45     path: "/login",
46     name: "login",
47     component: () => import('@views/login')
48 },
49 {
50     path: "/toLogin",
51     redirect: "/login"
52 },
53 {
54     path: "/error",
55     name: "error",
56     component: () => import('@views/error')
57 }
58 ]
59 });
60
61
62 // 1: 路由请求进入前置守卫
63 // 参数:
64 // 参数1: to : 当前你访问路由
65 // 参数2: from: 上次访问的路由
66 // 参数3: next: 进入下一个守卫 ,执行的时候必须是next()
67 // 返回值:
68 // 返回值1: return true 相当于 next() 继续执行请求
69 // 返回值2: return false : 终止请求
70 // 返回值3: return {path:""} , 如果你指定path, 就直接转发到你指定的地址, 相当于$router.push({path:"/xxx"});

```

```
71 // 返回值4: next()// 继续请求
72 router.beforeEach((to, from, next) => {
73     if (to.matched.length == 0) {
74         next({path: "/error"})
75     }
76     next();
77 })
78
79
80 router.beforeEach((to, from, next) => {
81     // 1: 排除不需要拦截的授权的页面
82     if (to.path == "/login") {
83         next()
84     }
85
86     //1: 如果登录成功，直接访问成功
87     if (sessionStorage.isAuth) {
88         next()
89     } else {
90         // 2: 如果用户访问的受保护的资源，且没有登录，则跳转
           到登录页面
91         // 举个场景：在操作系统，但是突然要吃饭了，过了30分钟
           以后继续来访问系统。因为token有时间限制。这个肯定会转发登录去、
92         // 为了良好的体验
93         // 将当前路由的完整的地址做为参数传递给Login.vue的组
           件，以便登录成功以后继续回到该位置
94         next({path: "/login", query: {back:
           to.fullPath}})
95     }
96
97 })
98
99 // 后置防卫
100 router.afterEach(to => {
```



```
101     document.title = to.meta.title;
102 })
103
104
105 // 3: 导出模板router模块即可
106 export default router
```

案例2：实现错误页面的

error.vue

```
1 <template>    <h1>我是错误页面</h1>    <router-link
  to="/">点我返回首页</router-link></template>
  <script>export default {  name: "error.vue",}</script>
  <style scoped></style>
```

路由注册

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256	257	258	259	260	261	262	263	264	265	266	267	268	269	270	271	272	273	274	275	276	277	278	279	280	281	282	283	284	285	286	287	288	289	290	291	292	293	294	295	296	297	298	299	300	301	302	303	304	305	306	307	308	309	310	311	312	313	314	315	316	317	318	319	320	321	322	323	324	325	326	327	328	329	330	331	332	333	334	335	336	337	338	339	340	341	342	343	344	345	346	347	348	349	350	351	352	353	354	355	356	357	358	359	360	361	362	363	364	365	366	367	368	369	370	371	372	373	374	375	376	377	378	379	380	381	382	383	384	385	386	387	388	389	390	391	392	393	394	395	396	397	398	399	400	401	402	403	404	405	406	407	408	409	410	411	412	413	414	415	416	417	418	419	420	421	422	423	424	425	426	427	428	429	430	431	432	433	434	435	436	437	438	439	440	441	442	443	444	445	446	447	448	449	450	451	452	453	454	455	456	457	458	459	460	461	462	463	464	465	466	467	468	469	470	471	472	473	474	475	476	477	478	479	480	481	482	483	484	485	486	487	488	489	490	491	492	493	494	495	496	497	498	499	500	501	502	503	504	505	506	507	508	509	510	511	512	513	514	515	516	517	518	519	520	521	522	523	524	525	526	527	528	529	530	531	532	533	534	535	536	537	538	539	540	541	542	543	544	545	546	547	548	549	550	551	552	553	554	555	556	557	558	559	560	561	562	563	564	565	566	567	568	569	570	571	572	573	574	575	576	577	578	579	580	581	582	583	584	585	586	587	588	589	590	591	592	593	594	595	596	597	598	599	600	601	602	603	604	605	606	607	608	609	610	611	612	613	614	615	616	617	618	619	620	621	622	623	624	625	626	627	628	629	630	631	632	633	634	635	636	637	638	639	640	641	642	643	644	645	646	647	648	649	650	651	652	653	654	655	656	657	658	659	660	661	662	663	664	665	666	667	668	669	670	671	672	673	674	675	676	677	678	679	680	681	682	683	684	685	686	687	688	689	690	691	692	693	694	695	696	697	698	699	700	701	702	703	704	705	706	707	708	709	710	711	712	713	714	715	716	717	718	719	720	721	722	723	724	725	726	727	728	729	730	731	732	733	734	735	736	737	738	739	740	741	742	743	744	745	746	747	748	749	750	751	752	753	754	755	756	757	758	759	760	761	762	763	764	765	766	767	768	769	770	771	772	773	774	775	776	777	778	779	780	781	782	783	784	785	786	787	788	789	790	791	792	793	794	795	796	797	798	799	800	801	802	803	804	805	806	807	808	809	810	811	812	813	814	815	816	817	818	819	820	821	822	823	824	825	826	827	828	829	830	831	832	833	834	835	836	837	838	839	840	841	842	843	844	845	846	847	848	849	850	851	852	853	854	855	856	857	858	859	860	861	862	863	864	865	866	867	868	869	870	871	872	873	874	875	876	877	878	879	880	881	882	883	884	885	886	887	888	889	890	891	892	893	894	895	896	897	898	899	900	901	902	903	904	905	906	907	908	909	910	911	912	913	914	915	916	917	918	919	920	921	922	923	924	925	926	927	928	929	930	931	932	933	934	935	936	937	938	939	940	941	942	943	944	945	946	947	948	949	950	951	952	953	954	955	956	957	958	959	960	961	962	963	964	965	966	967	968	969	970	971	972	973	974	975	976	977	978	979	980	981	982	983	984	985	986	987	988	989	990	991	992	993	994	995	996	997	998	999	1000	1001	1002	1003	1004	1005	1006	1007	1008	1009	1010	1011	1012	1013	1014	1015	1016	1017	1018	1019	1020	1021	1022	1023	1024	1025	1026	1027	1028	1029	1030	1031	1032	1033	1034	1035	1036	1037	1038	1039	1040	1041	1042	1043	1044	1045	1046	1047	1048	1049	1050	1051	1052	1053	1054	1055	1056	1057	1058	1059	1060	1061	1062	1063	1064	1065	1066	1067	1068	1069	1070	1071	1072	1073	1074	1075	1076	1077	1078	1079	1080	1081	1082	1083	1084	1085	1086	1087	1088	1089	1090	1091	1092	1093	1094	1095	1096	1097	1098	1099	1100	1101	1102	1103	1104	1105	1106	1107	1108	1109	1110	1111	1112	1113	1114	1115	1116	1117	1118	1119	1120	1121	1122	1123	1124	1125	1126	1127	1128	1129	1130	1131	1132	1133	1134	1135	1136	1137	1138	1139	1140	1141	1142	1143	1144	1145	1146	1147	1148	1149	1150	1151	1152	1153	1154	1155	1156	1157	1158	1159	1160	1161	1162	1163	1164	1165	1166	1167	1168	1169	1170	1171	1172	1173	1174	1175	1176	1177	1178	1179	1180	1181	1182	1183	1184	1185	1186	1187	1188	1189	1190	1191	1192	1193	1194	1195	1196	1197	1198	1199	1200	1201	1202	1203	1204	1205	1206	1207	1208	1209	1210	1211	1212	1213	1214	1215	1216	1217	1218	1219	1220	1221	1222	1223	1224	1225	1226	1227	1228	1229	1230	1231	1232	1233	1234	1235	1236	1237	1238	1239	1240	1241	1242	1243	1244	1245	1246	1247	1248	1249	1250	1251	1252	1253	1254	1255	1256	1257	1258	1259	1260	1261	1262	1263	1264	1265	1266	1267	1268	1269	1270	1271	1272	1273	1274	1275	1276	1277	1278	1279	1280	1281	1282	1283	1284	1285	1286	1287	1288	1289	1290	1291	1292	1293	1294	1295	1296	1297	1298	1299	1300	1301	1302	1303	1304	1305	1306	1307	1308	1309	1310	1311	1312	1313	1314	1315	1316	1317	1318	1319	1320	1321	1322	1323	1324	1325	1326	1327	1328	1329	1330	1331	1332	1333	1334	1335	1336	1337	1338	1339	1340	1341	1342	1343	1344	1345	1346	1347	1348	1349	1350	1351	1352	1353	1354	1355	1356	1357	1358	1359	1360	1361	1362	1363	1364	1365	1366	1367	1368	1369	1370	1371	1372	1373	1374	1375	1376	1377	1378	1379	1380	1381	1382	1383	1384	1385	1386	1387	1388	1389	1390	1391	1392	1393	1394	1395	1396	1397	1398	1399	1400	1401	1402	1403	1404	1405	1406	1407	1408	1409	1410	1411	1412	1413	1414	1415	1416	1417	1418	1419	1420	1421	1422	1423	1424	1425	1426	1427	1428	1429	1430	1431	1432	1433	1434	1435	1436	1437	1438	1439	1440	1441	1442	1443	1444	1445	1446	1447	1448	1449	1450	1451	1452	1453	1454	1455	1456	1457	1458	1459	1460	1461	1462	1463	1464	1465	1466	1467	1468	1469	1470	1471	1472	1473	1474	1475	1476	1477	1478	1479	1480	1481	1482	1483	1484	1485	1486	1487	1488	1489	1490	1491	1492	1493	1494	1495	14
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	----

```

1 //1: 导入路由import {createRouter, createWebHistory}
  from 'vue-router'//2: 创建路由器对象, 将模板全部进行路由匹配和
  注册const router = createRouter({    history:
  createWebHistory(),    routes: [        {
  path:"/",                name:"layout",
  component:()=>import('@layout/Layout'),
  redirect:"/index",                //这里只是重定向浏览器地址,
  但是渲染还是在父级,                // 好处: 就是可以打破不用和父同
  名默认是首页的规则                children:[                    {
                                path:"/index",
  name:"index",                component:
  ()=>import('@views/index')                },
                                {
                                path:"/user",
                                name:"user",                component:
  ()=>import('@views/user/index')                },
                                {
                                path:"/course",
                                name:"course",                component:
  ()=>import('@views/course/index')                },
                                {
                                path:"/order",
                                name:"order",
  component:()=>import('@views/order/index')
                                }
                                ]                },                {
  path:"/login",                name:"login",
  component:()=>import('@views/login')                },
  {
    path:"/error",                name:"error",
    component:()=>import('@views/error')                }
  ]});// 1: 路由请求进入前置守卫// 参数: // 参数1: to : 当前
你访问路由// 参数2: from: 上次访问的路由// 参数3: next: 进入下
一个守卫 ,执行的时候必须是next()// 返回值: // 返回值1: return
true 相当于 next() 继续执行请求// 返回值2: return false :
终止请求// 返回值3: return {path:""} , 如果你指定path, 就直接
转发到你指定的地址, 相当于$router.push({path:"/xxx"});// 返
回值4: next()// 继续请求router.beforeEach((to) => {
  if(to.matched.length == 0){                return

```

```
{path: "/error"}    })})// 3: 导出模板router模块即可export default router
```

核心代码

```
1 // 1: 路由请求进入前置守卫// 参数: // 参数1: to : 当前你访问路由// 参数2: from: 上次访问的路由// 参数3: next: 进入下一个守卫, 执行的时候必须是next()// 返回值: // 返回值1: return true 相当于 next() 继续执行请求// 返回值2: return false : 终止请求// 返回值3: return {path: ""} , 如果你指定path, 就直接转发到你指定的地址, 相当于$router.push({path: "/xxx"});// 返回值4: next()// 继续请求router.beforeEach((to) => {
  if(to.matched.length == 0){      return
    {path: "/error"}    })})
```

其实就利用：路由的前置防卫来处理拦截。

案例3：实现登录权限的控制

```
1 //1: 导入路由
2 import {createRouter, createWebHistory} from 'vue-router'
3
4 //2: 创建路由器对象, 将模板全部进行路由匹配和注册
5 const router = createRouter({
6   history: createWebHistory(),
7   routes: [
8     {
9       path: "/",
10      name: "layout",
```

```
11         component: () =>
import('@layout/Layout'),
12         redirect: "/index",
13         //这里只是重定向浏览器地址，但是渲染还是在父级，
14         // 好处：就是可以打破不用和父同名默认是首页的规则
15         children: [
16             {
17                 path: "/index",
18                 name: "index",
19                 meta:{title:"首页"},
20                 component: () =>
import('@views/index')
21             },
22             {
23                 path: "/user",
24                 name: "user",
25                 meta:{title:"用户管
理",isAuth:true},
26                 component: () =>
import('@views/user/index')
27             },
28             {
29                 path: "/course",
30                 name: "course",
31                 meta:{title:"课程管理"},
32                 component: () =>
import('@views/course/index')
33             },
34             {
35                 path: "/order",
36                 name: "order",
37                 meta:{title:"订单管理"},
38
```

```

39             component: () =>
import('@views/order/index')
40         }
41
42     ]
43 },
44 {
45     path: "/login",
46     name: "login",
47     component: () => import('@views/login')
48 },
49 {
50     path: "/toLogin",
51     redirect: "/login"
52 },
53 {
54     path: "/error",
55     name: "error",
56     component: () => import('@views/error')
57 }
58 ]
59 });
60
61
62 // 1: 路由请求进入前置守卫
63 // 参数:
64 // 参数1: to : 当前你访问路由
65 // 参数2: from: 上次访问的路由
66 // 参数3: next: 进入下一个守卫 ,执行的时候必须是next()
67 // 返回值:
68 // 返回值1: return true 相当于 next() 继续执行请求
69 // 返回值2: return false : 终止请求
70 // 返回值3: return {path:""} , 如果你指定path, 就直接转发到你指定的地址, 相当于$router.push({path:"/xxx"});

```

```
71 // 返回值4: next()// 继续请求
72 router.beforeEach((to, from, next) => {
73     if (to.matched.length == 0) {
74         next({path: "/error"})
75     }
76     next();
77 })
78
79
80 router.beforeEach((to, from, next) => {
81     // 1: 排除不需要拦截的授权的页面
82     if (to.path == "/login") {
83         next()
84     }
85
86     if(to.meta.isAuth) {
87         //1: 如果登录成功，直接访问成功
88         if (sessionStorage.isAuth) {
89             next()
90         } else {
91             // 2: 如果用户访问的受保护的资源，且没有登录，则
            跳转到登录页面
92             // 举个例子：在操作系统，但是突然要吃饭了，过了30
            分钟以后继续来访问系统。因为token有时间限制。这个肯定会转发登录
            去、
93             // 为了良好的体验
94             // 将当前路由的完整的地址做为参数传递给
            Login.vue的组件，以便登录成功以后继续回到该位置
95             next({path: "/login", query: {back:
            to.fullPath}}})
96         }
97     }
98
99     next();
```

```
100
101 })
102
103 // 后置防卫
104 router.afterEach(to => {
105     document.title = to.meta.title;
106 })
107
108
109 // 3: 导出模板router模块即可
110 export default router
```

03、守卫的执行流程

守卫：拦截器或者filter

1. 导航被触发。
2. 在失活的组件里调用 `beforeRouteLeave` 守卫。
3. 调用全局的 `beforeEach` 守卫
4. 在重用的组件里调用 `beforeRouteUpdate` 守卫 (2.2+)。
5. 在路由配置里调用 `beforeEnter`。
6. 解析异步路由组件。
7. 在被激活的组件里调用 `beforeRouteEnter`。
8. 调用全局的 `beforeResolve` 守卫 (2.5+)。
9. 导航被确认。
10. 调用全局的 `afterEach` 钩子。
11. 触发 DOM 更新。
12. 调用 `beforeRouteEnter` 守卫中传给 `next` 的回调函数，创建好的组件实例会作为回调函数的参数传入。

beforeEnter

守卫在全局守卫调用之后，只在进入路由时触发。每次进入只触发一次。不会监听参数和hash发生变化时触发。比如：/user/:id 匹配：/user/1 和 /user/2

```
1 // 设置欢迎页面，就path设置成 "/"即可
2 {
3   path: '/', name:"index",meta: {title: "首页"},
4   component: () => import("@/views/Home"),
5   // 针对性的处理
6   beforeEnter:function(){
7     console.log(2)
8   }
9 },
```

组件路由

定义的位置：在每个SPA页面（页面）

- beforeRouteEnter
- beforeRouteLeave
- beforeRouteUpdate

```
1 beforeRouteEnter(to,from,next){
2   // 在渲染该组件的路由被确认之前调用,不能通过this访问组件实例,因为在守卫执行前,组件实例还没有被创建。
3   console.log(to,from)
4   console.log("111")
}
```

```

5     next();
6 },
7
8     beforeRouteUpdate(to, from){
9         // 在渲染该组件的路由改变，但是在该组件被复用时调用
10        // 比如： 对一个带参数的路由： /user/:id 在/user/1
        和/user/2之间跳转时这个方法会被触发，如果是相同的访问不会被触
        发。
11        // 可以访问组件实例this
12        console.log(to, from)
13        console.log("2222")
14    },
15
16    beforeRouteLeave(to, from){
17        // 导航即将离开组件的路由时调用。
18        // 可以访问组件实例this
19        console.log(to, from)
20        console.log("33333")
21        var con = confirm("你还有数据没有保存成功！你真的要离开
        吗? ")
22        if(!con){
23            return false;
24        }
25    }

```

04、动态路由

就是通过js来添加路由：

- `addRoute({})`添加路由

```

1 // 方法注册 路由
2 router.addRoute({path: '/error', name:"error",meta:
  {title: "错误"}, components:
  {Video:import("@/views/Error")}}});
3 router.addRoute({path: '/course',
  name:"course",meta: {title: "课程"}, component: ()
  => import("@/views/Course")});
4 router.addRoute({path: '/video', name:"video",meta:
  {title: "视频"}, components:
  {Video:import("@/views/Video")}}});
5 // 方法注册 父子路由
6 router.addRoute( "book",{path:
  '/book/:id',name:"bookdetail",meta: {title: "书籍明
  细1"}, component: () =>
  import("@/views/BookDetail")});
7

```

- `removeRoute(name)`: 根据名字删除路由

```

1 router.removeRoute("book")

```

如果你删除的父，内部会自动把子路由全部删除。

- `hasRoute` 检查路由是否存在

```

1 router.hasRoute("book") 存在返回 true 不存在返回 false

```

- `getRoutes()`： 获取所有的路由

```

1 var routes = router.getRoutes();

```

04、旅游项目后台pug搭建全过程 - Axios

01 、Axios概述

官网: <http://www.axios-js.com/>

Axios 是一个基于 promise 的 HTTP 库, 可以用在浏览器和 node.js 中。

特性

- 从浏览器中创建 [XMLHttpRequests](#)
- 从 node.js 创建 [http](#) 请求
- 支持 [Promise](#) API
- 拦截请求和响应
- 转换请求数据和响应数据
- 取消请求
- 自动转换 JSON 数据
- 客户端支持防御 [XSRF](#)

浏览器支持: IE8以下不支持。

02、Axios安装

使用 cdn:

```
1 <script
  src="https://unpkg.com/axios/dist/axios.min.js">
</script>
```

使用 npm:

```
1 npm install axios -S
```

使用 yarn

```
1 yarn add axios -S
```

03、Axios&Vue-Axios注册

在vue的脚手架中使用，可以将 axios和vue-axios插件一起使用，该插件只是将axios集成到了vue.js的轻度封装。vue-axios本身不能独立使用，用如下命令一起安装，当然如果你前面安装了在覆盖安装也不会有太大的问题如下：

vue它自己没有异步处理框架。它写插件vue-axios，但是插件必须依赖 axios。也就告诉你：vue-axios必须安装axios。因为它的最终还是使用： axios

安装

```
1 yarn add axios vue-axios -S
2 或
3 npm install axios vue-axios -S
```

注册

```
1 import { createApp } from 'vue'
2 import App from './App.vue'
3 import router from './router'
4 import axios from 'axios'
5 import VueAxios from 'vue-axios'
6
7 var app = createApp(App);
8 app.use(router);
9 app.use(VueAxios);
10 app.mount('#app');
11
```

之后在任何的单页SPA中或者组件内都可以通过`this.axios`和`this.$http`调用`axios`的方法发送异步请求。

```
1 this.axios.get()-----axios.get()
2 this.$http.get()-----axios.get()
```

04、为什么不使用：vue-axios

使用`vue-axios`和`axios`结合使用不是很好吗？而且在任何的单页SPA中或者组件内都可以通过`this.axios`和`this.$http`调用`axios`的方法发送异步请求。

那为什么后面我们要自己去封装axios:

- vue-axios 满足不了业务需求，而且它只能在SPA和组件中使用。
- 如果在js模块中，你就没办法使用，只能自己去导入。模块不是收到vue的管理。外部的东西。只能使用导入。

05、Axios基本用法

HTTP最基本的两个请求是get请求和post请求，使用axios发送get请求形式如下：

执行 GET 请求

```
1  var axios = function (config){
2      return new Promise((resolve,reject) =>{
3          xmlhttprequest xhr = new xxxx;
4          xhr.url = config.url;
5          xhr.onreadystatechange = function(){
6              if(true){
7                  resolve()
8              }else{
9                  reject();
10             }
11         }
12         xhr.send();
13     })
14 }
15
16 var promise = axios({})
17 promise.then().catch();
```

```
1
2 // 为给定 ID 的 user 创建请求
3 var promise = axios.get('/user?ID=12345&name=yykk');
4 promise.then(function (response) {
5     console.log(response);
6 }).catch(function (error) {
7     console.log(error);
8 });
9
10
11 // 上面的请求也可以这样做
12 axios.get('/user', {
13     params: {
14         ID: 12345,
15         name: "yykk"
16     }
17 }).then(function (response) {
18     console.log(response);
19 }).catch(function (error) {
20     console.log(error);
21 });
```

执行 **POST** 请求


```
1 axios.post('/user', {
2   firstName: 'Fred',
3   lastName: 'Flintstone'
4 }).then(function (response) {
5   console.log(response);
6 }).catch(function (error) {
7   console.log(error);
8 });
```

执行多个并发请求

```
1 function getUserAccount() {
2   return axios.get('/user/12345');
3 }
4
5 function getUserPermissions() {
6   return axios.get('/user/12345/permissions');
7 }
8
9 axios.all([getUserAccount(), getUserPermissions()])
10  .then(axios.spread(function (acct, perms) {
11    // 两个请求现在都执行完成
12  }, function(err){
13    // 只要又一个报错，就执行结束
14  })));
```

也可以使用ES2017的async和await执行异步请求如下：

```
1  async function getBook(){
2      try{
3          const response1 = await axios.get("/book?
id=1");
4          const response2 = await axios.get("/book?
id=1");
5          console.log(response1,response2);
6      }catch(error){
7          console.error(error);
8      }
9  }
10
```

06、Axios API

可以通过向 `axios` 传递相关配置来创建请求

`axios(config)`

```
1  // 发送 POST 请求
2  axios({
3      method: 'post',
4      url: '/user/12345',
5      data: {
6          firstName: 'Fred',
7          lastName: 'Flintstone'
8      }
9  });
10
11
12 // 获取远端图片
```

```

13 axios({
14   method: 'get',
15   url: 'http://bit.ly/2mTM3nY',
16   responseType: 'stream'
17 })
18   .then(function(response) {
19     response.data.pipe(fs.createWriteStream('ada_lovelace.jpg'))
20   });

```

为方便起见，为所有支持的请求方法提供了别名

```

1 axios.request(config)
2 -----axios.get(url[, config])-----select
3 axios.delete(url[, config]) -----delete
4 axios.head(url[, config])
5 axios.options(url[, config])
6 -----axios.post(url[, data[, config]])-----update
7 axios.put(url[, data[, config]]) -----insert
8 axios.patch(url[, data[, config]])

```

注意

在使用别名方法时， url、method、data 这些属性都不必在配置中指定。

07、Axios 并发

处理并发请求的助手函数

- axios.all(iterable)——共生共死
- axios.spread(callback)——龟兔赛跑

08、Axios请求配置 config

```
1  {
2    // `url` 是用于请求的服务器 URL
3    url: '/user',
4
5    // `method` 是创建请求时使用的方法
6    method: 'get', // default
7
8    // `baseUrl` 将自动加在 `url` 前面，除非 `url` 是一个绝对 URL。
9    // 它可以通过设置一个 `baseUrl` 便于为 axios 实例的方法传递相对 URL
10   baseUrl: 'https://some-domain.com/api/',
11
12   // `transformRequest` 允许在向服务器发送前，修改请求数据
13   // 只能用在 'PUT', 'POST' 和 'PATCH' 这几个请求方法
14   // 后面数组中的函数必须返回一个字符串，或 ArrayBuffer，或 Stream
15   transformRequest: [function (data, headers) {
16     // 对 data 进行任意转换处理
17     return data;
18   }],
19
20   // `transformResponse` 在传递给 then/catch 前，允许修改响应数据
21   transformResponse: [function (data) {
22     // 对 data 进行任意转换处理
23     return data;
24   }],
25
26   // `headers` 是即将被发送的自定义请求头
27   headers: {'X-Requested-With': 'XMLHttpRequest'},
28 }
```

```
29 // `params` 是即将与请求一起发送的 URL 参数
30 // 必须是一个无格式对象(plain object)或 URLSearchParams
    对象
31 params: {
32     ID: 12345
33 },
34
35 // `paramsSerializer` 是一个负责 `params` 序列化的函数
36 // (e.g. https://www.npmjs.com/package/qs,
    http://api.jquery.com/jquery.param/)
37 paramsSerializer: function(params) {
38     return Qs.stringify(params, {arrayFormat:
    'brackets'})//?id=123456
39 },
40
41 // `data` 是作为请求主体被发送的数据
42 // 只适用于这些请求方法 'PUT', 'POST', 和 'PATCH'
43 // 在没有设置 `transformRequest` 时, 必须是以下类型之一:
44 // - string, plain object, ArrayBuffer,
    ArrayBufferView, URLSearchParams
45 // - 浏览器专属: FormData, File, Blob
46 // - Node 专属: Stream
47 data: {
48     firstName: 'Fred'
49 },
50
51 // `timeout` 指定请求超时的毫秒数(0 表示无超时时间)
52 // 如果请求话费了超过 `timeout` 的时间, 请求将被中断
53 timeout: 0,
54
55 // `withCredentials` 表示跨域请求时是否需要使用凭证
56 withCredentials: false, // default
57
58 // `adapter` 允许自定义处理请求, 以使测试更轻松
```

```
59 // 返回一个 promise 并应用一个有效的响应 (查阅 [response
docs](#response-api)).
60 adapter: function (config) {
61     /* ... */
62 },
63
64 // `auth` 表示应该使用 HTTP 基础验证, 并提供凭据
65 // 这将设置一个 `Authorization` 头, 覆写掉现有的任意使用
`headers` 设置的自定义 `Authorization` 头
66 auth: {
67     username: 'janedoe',
68     password: 's00pers3cret'
69 },
70
71 // `responseType` 表示服务器响应的数据类型, 可以是
'arraybuffer', 'blob', 'document', 'json', 'text',
'stream'
72 responseType: 'json', // default
73
74 // `responseEncoding` indicates encoding to use for
decoding responses
75 // Note: Ignored for `responseType` of 'stream' or
client-side requests
76 responseEncoding: 'utf8', // default
77
78 // `xsrCookieName` 是用作 xsrf token 的值的cookie的
名称
79 xsrfCookieName: 'XSRF-TOKEN', // default
80
81 // `xsrHeaderName` is the name of the http header
that carries the xsrf token value
82 xsrfHeaderName: 'X-XSRF-TOKEN', // default
83
84 // `onuploadProgress` 允许为上传处理进度事件
```

```
85   onUploadProgress: function (progressEvent) {
86     // Do whatever you want with the native progress
event
87   },
88
89   // `onDownloadProgress` 允许为下载处理进度事件
90   onDownloadProgress: function (progressEvent) {
91     // 对原生进度事件的处理
92   },
93
94   // `maxContentLength` 定义允许的响应内容的最大尺寸
95   maxContentLength: 2000,
96
97   // `validateStatus` 定义对于给定的HTTP 响应状态码是
resolve 或 reject promise 。如果 `validateStatus` 返回
`true` (或者设置为 `null` 或 `undefined`), promise 将被
resolve; 否则, promise 将被 rejecte
98   validateStatus: function (status) {
99     return status >= 200 && status < 300; // default
100   },
101
102   // `maxRedirects` 定义在 node.js 中 follow 的最大重定向
数目
103   // 如果设置为0, 将不会 follow 任何重定向
104   maxRedirects: 5, // default
105
106   // `socketPath` defines a UNIX Socket to be used in
node.js.
107   // e.g. '/var/run/docker.sock' to send requests to
the docker daemon.
108   // Only either `socketPath` or `proxy` can be
specified.
109   // If both are specified, `socketPath` is used.
110   socketPath: null, // default
```

```
111
112 // `httpAgent` 和 `httpsAgent` 分别在 node.js 中用于定
    义在执行 http 和 https 时使用的自定义代理。允许像这样配置选
    项：
113 // `keepAlive` 默认没有启用
114 httpAgent: new http.Agent({ keepAlive: true }),
115 httpsAgent: new https.Agent({ keepAlive: true }),
116
117 // 'proxy' 定义代理服务器的主机名称和端口
118 // `auth` 表示 HTTP 基础验证应当用于连接代理，并提供凭据
119 // 这将会设置一个 `Proxy-Authorization` 头，覆写掉已有的
    通过使用 `header` 设置的自定义 `Proxy-Authorization` 头。
120 proxy: {
121   host: '127.0.0.1',
122   port: 9000,
123   auth: {
124     username: 'mikeymike',
125     password: 'rapunz31'
126   }
127 },
128
129 // `cancelToken` 指定用于取消请求的 cancel token
130 // （查看后面的 Cancellation 这节了解更多）
131 cancelToken: new CancelToken(function (cancel) {
132 })
133 }
```

09、Axios响应结构

```
1 {
2   // `data` 由服务器提供的响应
```



```

3   data: {
4
5   },
6
7   // `status` 来自服务器响应的 HTTP 状态码
8   status: 200,
9
10  // `statusText` 来自服务器响应的 HTTP 状态信息
11  statusText: 'OK',
12
13  // `headers` 服务器响应的头
14  headers: {},
15
16  // `config` 是为请求提供的配置信息
17  config: {},
18  // 'request'
19  // `request` is the request that generated this
  response
20  // It is the last ClientRequest instance in node.js
  (in redirects)
21  // and an XMLHttpRequest instance the browser
22  request: {}
23 }

```

使用 `then` 时，你将接收下面这样的响应：

```

1  axios.get('/user/12345')
2    .then(function(response) {
3      console.log(response.data);
4      console.log(response.status);
5      console.log(response.statusText);
6      console.log(response.headers);
7      console.log(response.config);
8    });

```

在使用 `catch` 时，或传递 `rejection callback` 作为 `then` 的第二个参数时，响应可以通过 `error` 对象可被使用，正如在[错误处理](#)这一节所讲。

10、Axios执行一个异步请求

- 启动项目，发布和部署项目
- 准备接口：<http://120.77.34.190:8081/admin/v1/user/get/1>
- 在项目中导入：`npm install axios vue-axios -S`
- 在index.vue 使用一部调用接口

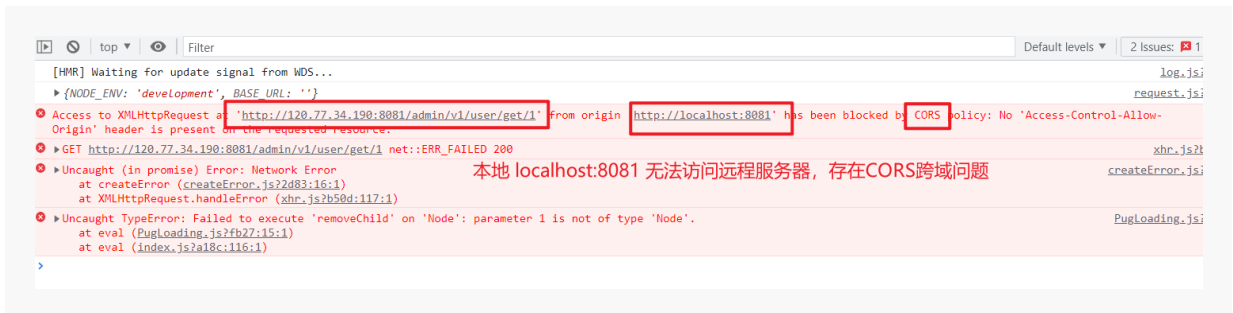
```
1 <template>
2   <h1>我是首页</h1>
3 </template>
4
5 <script>
6
7   import request from "@/utils/request";
8
9   export default {
10     name: "index.vue",
11     data() {
12       return {}
13     },
14
15     created() {
16       var url =
17         "http://120.77.34.190:8081/admin/v1/user/get/1";
18       this.axios.get(url).then(res => {
19         console.log(res);
20       })
21     }
22   }
```

```

19     })
20   },
21
22   methods: {}
23
24 }
25 </script>
26
27 <style scoped>
28
29 </style>

```

- 出现异常



🐼 11、Axios什么是跨域

参考文章: <https://www.cnblogs.com/yuansc/p/9076604.html>

只要访问过程中, 协议, 端口, ip不同就存在跨域。

- 本地域: <http://localhost:8081>
- 服务域: <https://120.77.34.190:8081>

解决方案:

```
1 package com.ksd.pug.config.webmvc;
2
3 import
  com.ksd.pug.config.interceptor.jwt.JwtInterceptor;
4 import
  com.ksd.pug.config.interceptor.repeat.RepeatSubmitInte
  rceptor;
5 import
  org.springframework.beans.factory.annotation.Autowired
  ;
6 import
  org.springframework.context.annotation.Configuration;
7 import
  org.springframework.web.servlet.config.annotation.Cors
  Registry;
8 import
  org.springframework.web.servlet.config.annotation.Inte
  rceptorRegistry;
9 import
  org.springframework.web.servlet.config.annotation.WebM
  vcConfigurer;
10
11 /**
12  * @author 飞哥
13  * @Title: 学相伴出品
14  * @Description: 飞哥B站地址:
15  * https://space.bilibili.com/490711252
16  * 记得关注和三连哦!
17  * @Description: 我们有一个学习网站:
18  * https://www.kuangstudy.com
19  * @date 2022/1/6 17:50
20  */
21 @Configuration
```

```
20 public class WebMvcConfiguration implements
webMvcConfigurer {
21
22     /**
23      * jwt的token校验
24      */
25     @Autowired
26     public JwtInterceptor jwtInterceptor;
27
28     /**
29      * 表单重复提交
30      */
31     @Autowired
32     private RepeatSubmitInterceptor
repeatSubmitInterceptor;
33
34     /**
35      * 拦截的注册
36      * @param registry
37      */
38     @Override
39     public void addInterceptors(InterceptorRegistry
registry) {
40
41         registry.addInterceptor(repeatSubmitInterceptor).addP
athPatterns("/admin/v1/**");
42
43         registry.addInterceptor(jwtInterceptor).addPathPatter
ns("/admin/v1/**")
44
45         .excludePathPatterns("/admin/v1/user/**");
46     }
47
48     /**
```

```
46      * 解决跨域问题
47      * @param registry
48      */
49      @Override
50      public void addCorsMappings(CorsRegistry registry)
51      {
52          registry
53              .addMapping("/**")
54              // .allowedOrigins("http://yyy.com",
55              "http://xxx.com") //
56              // 允许跨域的域名
57              .allowedOriginPatterns("*") // 允许所有
58              域
59              .allowedMethods("POST", "GET", "PUT",
60              "OPTIONS", "DELETE")
61              .allowedMethods("*") // 允许任何方法
62              (post、get等)
63              .allowedHeaders("*") // 允许任何请求头
64              .allowCredentials(true) // 允许证书、
65              cookie
66              .maxAge(3600L); // maxAge(3600)表明在
67              3600秒内，不需要再发送预检验请求，可以缓存该结果
68      }
69  }
```

12、创建Axios实例

注意：

使用`vue-axios`和`axios`结合使用不是很好吗？而且在任何的单页SPA中或者组件内都可以通过`this.axios`和`this.$http`调用`axios`的方法发送异步请求。

那为什么后面我们要自己去封装`axios`：

- `vue-axios` 满足不了业务需求，而且它只能在SPA和组件中使用。
- 如果在js模块中，你就没办法使用，只能自己去导入。模块不是收到vue的管理。外部的东西。只能使用导

可以使用自定义配置调用`axios.create([config])`方法创建一个`axios`实例，之后使用该实例向服务器端发起请求，就不用每次请求时重复设置配置项了。如下：

```
1 const instance = axios.create({
2   baseURL: 'https://some-domain.com/api/',
3   timeout: 1000,
4   headers: {'X-Custom-Header': 'foobar'}
5 });
```

13、Axios配置默认值

你可以指定将被用在各个请求的配置默认值

全局的 axios 默认值

```
1 axios.defaults.baseURL = 'https://api.example.com';
2 axios.defaults.headers.common['Authorization'] =
  AUTH_TOKEN;
3 axios.defaults.headers.post['Content-Type'] =
  'application/x-www-form-urlencoded';
```

自定义实例默认值

```
1 // Set config defaults when creating the instance
2 const instance = axios.create({
3   baseURL: 'https://api.example.com'
4 });
```

```
1 // Alter defaults after instance has been created
2 instance.defaults.headers.common['Authorization'] =
  AUTH_TOKEN;
```

14、Axios拦截器

在请求或响应被 `then` 或 `catch` 处理前拦截它们。

```
1 // 添加请求拦截器
2 axios.interceptors.request.use(function (config) {
3   // 在发送请求之前做点什么
4   return config;
```



```

5     }, function (error) {
6         // 对请求错误做些什么
7         return Promise.reject(error);
8     });
9
10    // 添加响应拦截器
11    axios.interceptors.response.use(function (response) {
12        // 对响应数据做点什么
13        return response;
14    }, function (error) {
15        // 对响应错误做点什么
16        return Promise.reject(error);
17    });

```

如果你想在稍后移除拦截器，可以这样：

```

1    const myInterceptor =
    axios.interceptors.request.use(function () { /*...*/ });
2    axios.interceptors.request.eject(myInterceptor);

```

可以为自定义 axios 实例添加拦截器

```

1    const instance = axios.create();
2    instance.interceptors.request.use(function ()
    { /*...*/ });

```

15、Axios 错误处理

```

1    axios.get('/user/12345')
2        .catch(function (error) {
3            if (error.response) {

```

```

4      // The request was made and the server responded
    with a status code
5      // that falls out of the range of 2xx
6      console.log(error.response.data);
7      console.log(error.response.status);
8      console.log(error.response.headers);
9  } else if (error.request) {
10     // The request was made but no response was
    received
11     // `error.request` is an instance of
    XMLHttpRequest in the browser and an instance of
12     // http.ClientRequest in node.js
13     console.log(error.request);
14 } else {
15     // Something happened in setting up the request
    that triggered an Error
16     console.log('Error', error.message);
17 }
18 console.log(error.config);
19 });

```

可以使用 `validateStatus` 配置选项定义一个自定义 HTTP 状态码的错误范围。

```

1 axios.get('/user/12345', {
2   validateStatus: function (status) {
3     return status < 500; // Reject only if the status
    code is greater than or equal to 500
4   }
5 })

```

```

1 npm install node-sass --save-dev
2 npm install sass-loader --save-dev

```

📖 Vue中的系统环境变量参数

vue中 .env .env.development .env.production 详细说明

🧠 1.配置文件有：

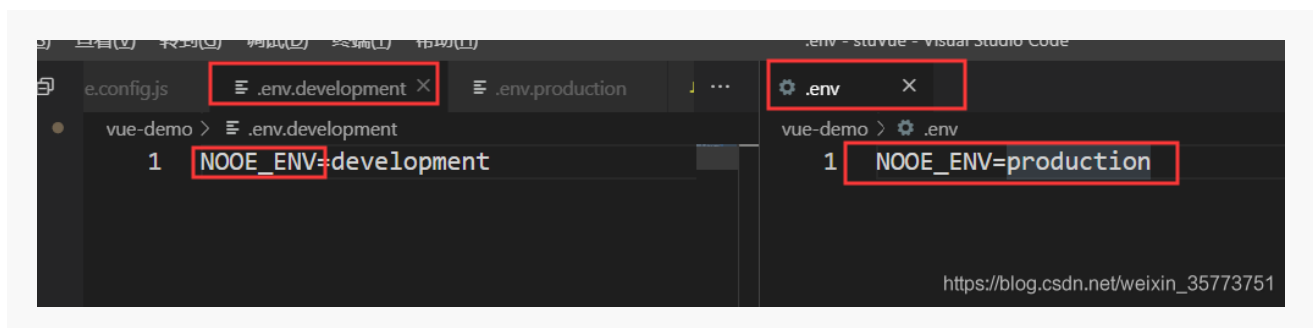
.env 全局默认配置文件，不论什么环境都会加载合并

.env.development 开发环境下的配置文件

.env.production 生产环境下的配置文件

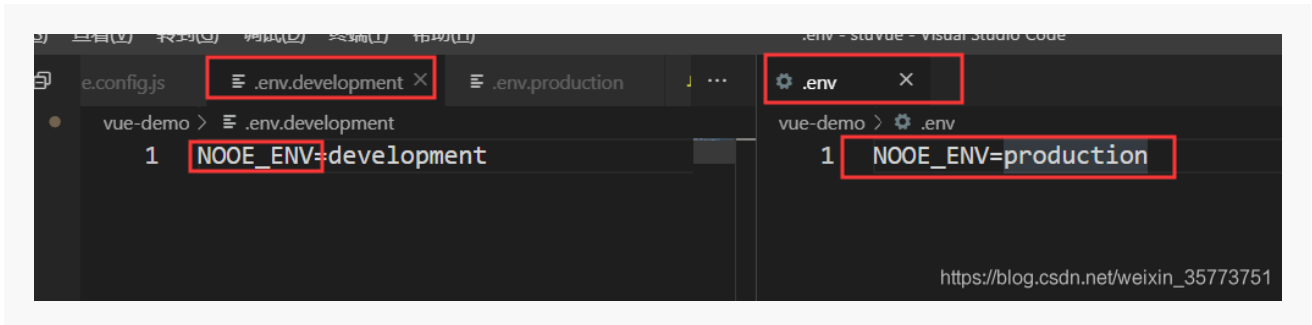
🧠 2.命名规则：

属性名必须以VUE_APP_开头，比如VUE_APP_XXX



🧠 3.关于文件的加载:

- 根据启动命令vue会自动加载对应的环境，vue是根据文件名进行加载
- 比如执行npm run serve命令，会自动加载.env.development文件
- 注意：.env文件无论是开发还是生成都会加载

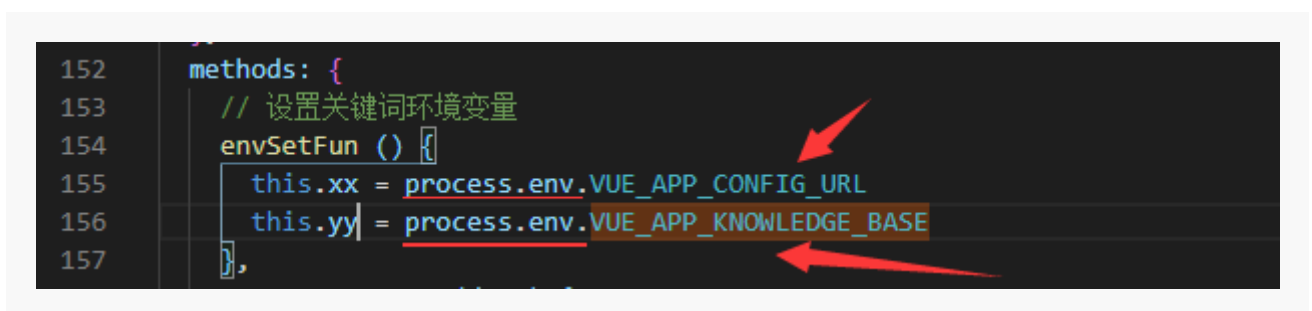


如上图所示，如过我们运行npm run serve 在就先加载.env文件，之后加载.env.development文件，两个文件有同一个项，则后加载的文件就会覆盖掉第一个文件，即.env.development文件覆盖掉了.env文件的NOOE_ENV选项。

同理如果npm run build 就执行了.env和.env.production。

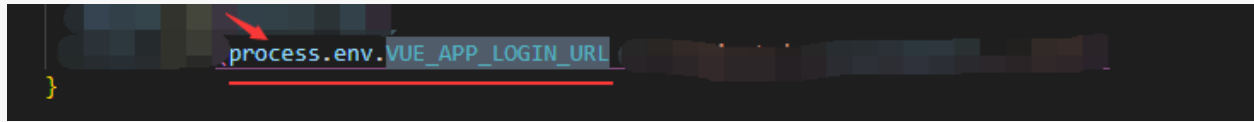
🧠 4.使用

(1) 在vue文件中使用



注意：配置文件在vue模板文件的data之后加载

(2) 在js文件中使用



备注：js文件中可以添加JSON.stringify(xxxxxx)，解析成字符，但是vue中不能。

📖 05、旅游项目后台pug搭建全过程 - Vuex

📖 06、旅游项目后台pug搭建全过程 - 项目划分重构
