小程序: WEB问题速查

1. 问题清单: 历届师兄师姐在今天课程中集中问过的高频的重点的问题！

那些自己发现不了问题，又迫切想知道其它同学都问了什么问题，的同学

小程序->首页->VUE->day04、day05

当老师在回答其它同学的问题时，你不要闲着！你可以打开问题清单浏览别人曾经问过的经典问题！

2. 高清无码小视频: 小程序->在线->VUE->day04、day05

2部分: 1. 课上重点知识和案例的回放！2. 重点高频笔试面试题讲解扩展视频

万一家里或中心网络断了！暂时看不了直播了！

因为某些事儿请假了，错过了当天的直播！

课下复习时，觉得TMOOC视频太大，浏览不方便或TMOOC不能看了！

——打开小程序中"在线"，就可下载观看当前所讲内容和当天所讲全部内容的回放。

所有人！笔试面试前！我的小程序中所有视频必须看完！

3. 单词列表: 小程序->单词->VUE->day04、day05

第三阶段上课约定:

1. 上课千万不要跟着记笔记和写代码！

2. 凡是拼写错误，自己懒得找！我可以帮你找，但是找到后，错误的单词你要手抄20遍，并拍照！

出错: 先开f12，console，看错误提示！

3. 每个带注释的案例至少写三遍:

(1). 注释+代码抄一遍

(2). 保留注释，删除代码，自己试着把代码翻译回来！

(3). 删除注释和代码，自己试着用自己的话把注释写回来！

一个例子只有自己可以把人话的注释写回来！才算自己会！

问题: 一听就会一做就废！原因: 你自己无法用人话精确描述自己想干什么！

如何提问: （vip待遇的剩余时间1+1天）

1. 上课时，案例中，笔记中那句话哪行代码看不懂，立刻问！

2. 小程序中问题清单里哪个问题和回答看不懂！立刻问！

3. 个人项目中哪个功能不会做！立刻问！

4. 刷笔试题时，遇到不会的笔试题！立刻问！

谁问的问题越多！谁的培训费交的越值！

正课:

1. SPA

2. 脚手架

3. 懒加载

一. **SPA**: Single Page Application

单 页面 应用程序

对应小程序视频: 小程序->在线->VUE->day04 1. spa单页面应用...

1. 什么是: **整个应用程序只有一个唯一完整的.html文件**。**其它所谓的"页面"，都是子组件而已。**所谓的"跳转页面"，其实只是在一个页面中切换显示不同的子组件内容而已。

2. 为什么: 多页面 vs 单页面

|  |  |  |
| --- | --- | --- |
|  | 多页面 | 单页面 |
| 请求次数 | 每跳转一个页面都要向服务器重新发送请求，即使访问同一个页面，也需要反复发送请求——请求次数多 | 只在首次就把唯一完整的HTML页面和其它页面的组件用一个请求都下载到客户端本地。每次切换页面时，只切换唯一完整的HTML页面中部分片段而已，无需重复向服务器发送请求。——请求次数少 |
| 公共资源的使用 | 因为每次都要重新下载和扫描完整的HTML页面，所以每次浏览器扫描到新页面中的link,script等文件引入时，都会重新向服务器发送请求。这对于每个页面都要用到的公共的第三方css和js文件来说，是完全没必要的！——请求次数多，浪费流量！ | 因为每次切换页面时，完整的HTML页面不会整体切换，只会切换其中的一小部分而已。公共区域保持不变！浏览器就不会重新下载公共区域引入的css和js文件——极大的减少了请求次数和网络流量！ |
| 加载效率 | 因为每次切换页面，都要完整删除旧页面，加载完整的新页面，所以旧DOM树一点儿也用不上。每次切换页面都需要重建整棵DOM树——加载效率低 | 因为每次切换页面时，不删除整个页面，仅仅修改页面中变化的一部分区域。所以，旧的DOM树无需重建，只要改变部分节点即可！——加载效率高！ |
| 页面切换动画 | 多页面应用不可能实现页面切换动画。因为页面切换动画要求同时显示前一个页面的后一半和后一个页面的前一半，多页面应用因为每次只显示一个页面，所以无法实现！ | 单页面应用因为所有组件已经保存在客户端本地，且组件本质就是一个元素而已，所以单页面应用完全有可能控制元素的显示位置和移动——有可能实现页面切换动画 |

3. 单页面应用的**缺点**: **首屏加载慢**——因为不但下载首页，还要下载其他页面的组件备用。

早就被解决了: 懒加载(稍后讲)

4. 何时: 今后几乎所有移动端，以及大部分PC端项目都使用单页面应用开发。

5. 如何: 3步

(1). **先创建一个唯一完整的HTML页面**

a. 拥有基本的vue页面结构的页面

b. 引入vue-router.js路由器组件文件

c. 在唯一完整的页面中使用**<router-view>**标签为将来的页面组件设置**预留区域**

(2). **为每个"页面"创建对应的页面组件**（就是子组件，只不过当做页面来用）

将所有页面组件的文件，引入唯一完整的HTML页面中

(3). **创建路由器对象和路由字典**

a. 什么是路由器对象: 可自动**侦测**浏览器**地址栏**的url**变化**，**根据**用户输入的**不同地址**，自动**找到对应的组件**对象，帮我们**替换**到唯一完整的**HTML页面中<router-view>**区域

b. 什么是路由字典: 集中保存不同地址与组件对象之间对应关系的**数组**

c. 为什么要使用路由字典: 路由器对象侦测到地址栏变化时，必须去路由字典中找到对应的路径，才能进一步找到对应的组件对象。

d. 如何创建路由器对象: 3步: 创建独立的router.js文件中

1). **先创建路由字典数组:**

**var routes=[**

**{ path:"/", component: 首页组件对象 },**

{ path:"/其它路径", component: 其它页面组件对象 },

... ...

]

2). **创建路由器对象**，并将**路由字典装入路由器对象中**

**var router=new VueRouter({ routes })**

3). 将**路由器对象引入new Vue()中**: 回到唯一完整的HTML页面中:

**先将router.js文件引入唯一完整的HTML页面中，**

再**new Vue({ el:"#app", router })**

6. **公共的页头组件:**

(1). **创建**独立的页头组件文件，将页头组件创建为**全局组件**，因为页头不属于任何一个页面，可用在任意页面中

**Vue.component("my-header", { ... })**

(2). 在唯一完整的HTML页面中引入页头组件文件

(3). 问题: **<my-header/>标签放在那儿**？2种:

**a.** 如果所有页面都有统一的页头，则可以将<my-header/>放在唯一完整的HTML页面中<router-view/>外的上方！

**b.** 如果个别页面不希望有页头！则也可以将<my-header/>标签只放在需要页头的页面组件中！

7. 如果用户路径错误: **自制一个友好的404页面**

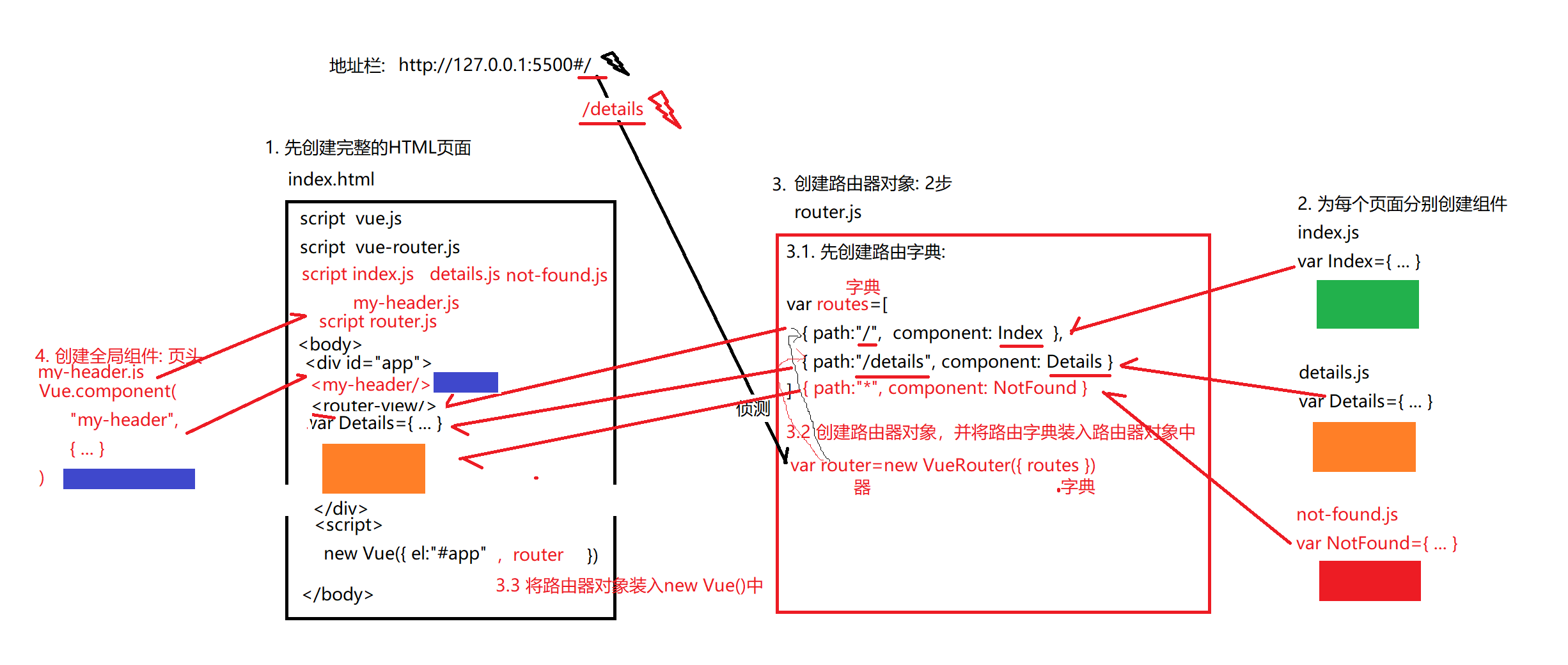
(1). 创建一个子组件保存404页面的内容

(2). 去路由字典中加入404页面组件对应的路径

**{ path:"\*", component: 404页面组件对象 }**

强调: **path:"\*"必须写在其它路由最后！\*代表一切路径！**

(3). 将404页面组件引入到唯一完整的HTML页面中



8. 示例: 使用单页面应用程序实现一个网站，包含三个页面一个页头

spa/js/index.js

|  |
| --- |
| var Index={    template:`<div>      <my-header></my-header>      <h1 style="color:green">这里是首页</h1>    </div>`  } |

spa/js/details.js

|  |
| --- |
| var Details={    template:`<div>      <my-header></my-header>      <h1 style="color:orange">这里是详情页</h1>    </div>`  } |

spa/js/not-found.js

|  |
| --- |
| var NotFound={    template:`<div>      <h1 style="color:red">404: 你迷路了！</h1>    </div>`  } |

spa/js/my-header.js

|  |
| --- |
| Vue.component("my-header",{    template:`<div>      <h1 style="color:blue">这里是页头</h1>      <hr>    </div>`  }) |

spa/js/router.js

|  |
| --- |
| var routes=[    {path:"/", component: Index},    {path:"/details", component: Details},    {path:"\*", component: NotFound}  ];  var router=new VueRouter({ routes }) |

spa/index.html

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <title>Document</title>    <!--1. 引入vue和vue-router基础文件-->    <script src="js/vue.js"></script>    <script src="js/vue-router.js"></script>    <!--2. 引入全局组件和页面组件-->    <script src="js/my-header.js">      //Vue.component("my-header")    </script>    <script src="js/index.js">      //var Index={}    </script>    <script src="js/details.js">      //var Details={}    </script>    <script src="js/not-found.js">      //var NotFound={}    </script>    <!--3. 最后引入路由器组件-->    <script src="js/router.js">      //var routes=[        //用到了Index和Details两个对象名      //]      //var router=new VueRouter({ routes })    </script>    </head>  <body>    <div id="app">      <!-- <my-header></my-header> -->      <router-view>        <!--为将来的页面组件预留区域-->      </router-view>    </div>    <script>      new Vue({        el:"#app",        router      })    </script>  </body>  </html>  运行结果: |

9. **页面跳转**: 2种 固定套路！记住！  
 (1). 在**页面上写死**一个标签:

~~a. 不要用<a>~~

b. 改为用**<router-link to="/相对路径">文本</router-link>**

c. 原理: <router-link to>在编译后，也会自动翻译为<a href>

(2). 在j**s程序中跳转**: 绑定事件处理函数

**this.$router.push("/相对路径")**

路由器对象

10. **路由传参**:

(1). 需求: 实际项目中，前一个页面，跳转到后一个页面时，前一个页面有时会传递一个数据到后一个页面中继续使用！

(2). 多页面应用: 使用url中search查询字符串方式传递:

a. 前一个页面: <a href="下一个页面.html?变量名=变量值">

b. 下一个页面:

1). location.search 可获得 "?变量名=变量值"

|

[0] [1]

2). location.search.split("=")[1] 可获得传过来的变量值

复习小程序DOM第四天和JQUERY第四天的学习商城项目视频

(3). vue单页面应用程序中:

a. 修改路由字典中的项目，允许携带参数:

**{ path:"/相对路径/:自定义变量名", component: 页面组件,** **props:true }**

允许携带一个参数变量

props:true, 让参数**自定义变量**自动成为下一个页面的props中的属性。在下个页面中就可直接使用**自定义变量名**获取传过来**的参数值**

b. **跳转时**，路径携带参数值:

**<router-link to=/相对路径/参数值>**

或

**this.$router.push("/相对路径/参数值")**

c. 在下一个组件中如何接：

1). 先**props:[ "自定义变量" ]**

2). 在下一个组件中就可以像使用自己的data中的变量一样，使用页面跳转时出来的参数值

(4). vue中规定: **如果一个路径要求携带参数，每次跳转时都必须携带参数值！**

否则如果将来跳转时不携带参数，就不允许跳转！而是转向404页面！

11. 示例: 为上一个单页面应用示例加上跳转页面功能，以及在跳转时传参

spa/js/index.js

|  |
| --- |
| var Index={    template:`<div>      <my-header></my-header>      <h1 style="color:green">这里是首页</h1>      <router-link to="/details/5">查看5号商品的详情</router-link><br/>      <button @click="goto(10)">查看10号商品的详情</button>    </div>`,    methods:{      goto(lid){        this.$router.push(`/details/${lid}`);      }    }  } |

spa/js/details.js

|  |
| --- |
| var Details={    //因为路由字典中写了props:true    //所以lid参数变量会自动成为props中的属性    props:["lid"], //和data中的变量用法完全一样，只不过来源不同而已。props来自于外部，data是自己创建的。    template:`<div>      <my-header></my-header>      <h1 style="color:orange">这里是详情页</h1>      <h2>这里是{{lid}}号商品的详细信息...</h2>      <button @click="back">返回首页</button>    </div>`,    methods:{      back(){        this.$router.push("/");      }    }  } |

spa/js/not-found.js

|  |
| --- |
| var NotFound={    template:`<div>      <h1 style="color:red">404: 你迷路了！</h1>    </div>`  } |

spa/js/my-header.js

|  |
| --- |
| Vue.component("my-header",{    template:`<div>      <h1 style="color:blue">这里是页头</h1>      <ul>        <li><router-link to="/">首页</router-link></li>        <li><router-link to="/details">详情页</router-link></li>      </ul>      <hr>    </div>`  }) |

spa/js/router.js

|  |
| --- |
| var routes=[    {path:"/", component: Index},    //允许/details路径携带一个变量名为lid    //props:true意为，让lid变量自动成为details组件中的props属性    {path:"/details/:lid", component: Details, props:true},    {path:"\*", component: NotFound}  ];  var router=new VueRouter({ routes }) |

spa/index.html

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <title>Document</title>    <!--1. 引入vue和vue-router基础文件-->    <script src="js/vue.js"></script>    <script src="js/vue-router.js"></script>    <!--2. 引入全局组件和页面组件-->    <script src="js/my-header.js">      //Vue.component("my-header")    </script>    <script src="js/index.js">      //var Index={}    </script>    <script src="js/details.js">      //var Details={}    </script>    <script src="js/not-found.js">      //var NotFound={}    </script>    <!--3. 最后引入路由器组件-->    <script src="js/router.js">      //var routes=[        //用到了Index和Details两个对象名      //]      //var router=new VueRouter({ routes })    </script>    </head>  <body>    <div id="app">      <!-- <my-header></my-header> -->      <router-view>        <!--为将来的页面组件预留区域-->      </router-view>    </div>    <script>      new Vue({        el:"#app",        router      })    </script>  </body>  </html>  运行结果: |

**二. 脚手架:**

对应小程序视频: 小程序->在线->VUE->day04 2. 使用vue/cli工具创建学子商城项目

1. 什么是: 已经拥有核心功能以及标准文件夹结构的半成品项目

2. 为什么:

(1). **标准的文件夹结构**

(2). **避免重复编码**

3. 何时: 今后几乎所有前端项目都是在脚手架基础上创建的！

4. 如何: 2步:

(0). 检测当前系统有没有有安装脚手架工具:

vue -V

(1). 安装可反复生成脚手架代码的工具(老母鸡)

**npm i -g @vue/cli**

如果安装不成功:

a. 安装淘宝镜像:

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

npm config get registry

b. 网络: 如果wifi或宽带安装不了，可以用手机开热点，用移动网络安装

c. nodejs: 清空nodejs缓存npm cache clean -f，然后重装nodejs

d. 操作系统: 重装操作系统

(2). 用工具反复生成每个项目的脚手架代码(下的蛋)

vue create 项目名

5. **用vue/cli工具创建脚手架项目的步骤:**

(0). 在想创建项目的文件夹 地址栏中输入cmd，按回车，

输入：**vue create 项目名**

(1). Vue CLI v4.1.1

┌──────────────────┐

│ │

│ New version available 4.1.1 → 4.4.6 │

│ Run npm i -g @vue/cli to update! │

│ │

└──────────────────┘

vue/cli工具的版本不是最新，用npm i -g @vue/cli来升级

**我选: 忽视**！

(2). ? Please pick a preset: (Use arrow keys)

请选择一种预设的项目配置

default (babel, eslint)

默认

**> Manually select features**

手工 选择 功能

(3). ? Check the features needed for your project: (Press <space> to select, <a> to toggle all, <i> to invert selection)

选择你的项目所需的功能（**按空格选中**。。。）

**>(\*) Babel** 将脚手架中时髦的ES6和ES7语法，翻译为大多数浏览器认识的ES5语法

( ) TypeScript //严格的js语言，Angularjs框架中强制使用，但是vue中不强制使用

( ) Progressive Web App (PWA) Support

**(\*) Router** //单页面应用必须的核心组件 路由器

**(\*) Vuex** //客户端状态管理用的

( ) CSS Pre-processors //如果在项目中想使用动态css sass或less语法，需要选择

( ) Linter / Formatter //代码质量检查工具，会把书写不规范，也抛出成错误和警告！

( ) Unit Testing

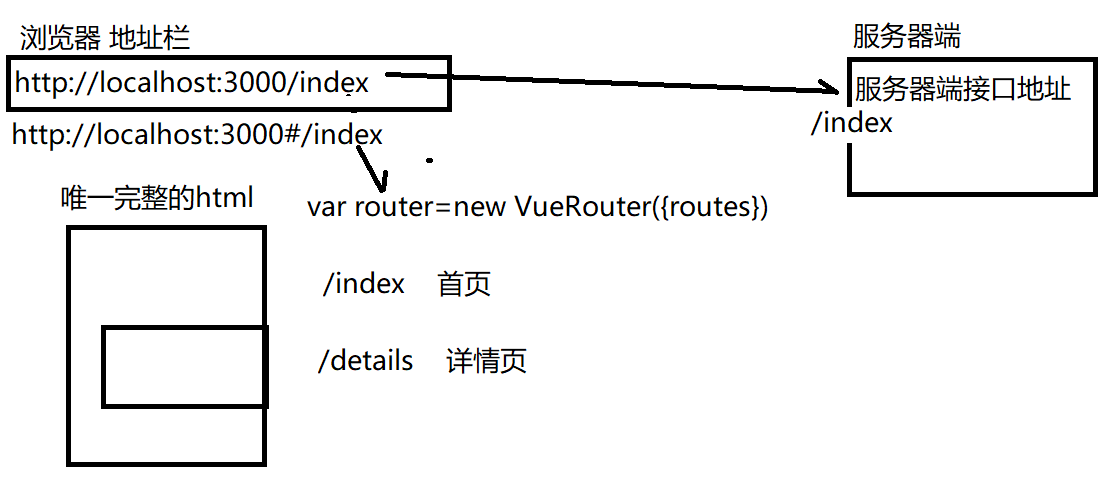
( ) E2E Testing

(4). ? Use history mode for router? (Requires proper server setup for index fallback in production) (Y/n)

是否启用history模式作为路由的方式? (但是，要求服务器端首页重定向的支持)

**我选: n**

vue-router在客户端路由时，使用的是"#/相对路径"，为了和其它向服务器端发送的请求地址区分开。浏览器就可以不带#的/相对路径就发给服务器端处理。而带有#/相对路径，就不发给服务器，而是交给客户端本地的vue-router处理



问题: 如果在一个很长的网页中，也想用锚点地址进行页面内定位导航，就会和vue-router的#/冲突。

解决: 启用history模式路由: 客户端导航也不再使用#/，而是改为/

问题: 启用history模式需要专业的服务器端支持！

所以: 现在学习阶段不要启用history模式！

(5). ? Where do you prefer placing config for Babel, ESLint, etc.? (Use arrow keys)

希望将项目的配置信息放在那里？

In dedicated config files 将不同的配置信息，放在不同的配置文件中

**> In package.json** 将所有配置信息，放在一个统一的package.json文件中

(6). Save this as a preset for future projects? (y/N)

是否保存这个项目的配置为今后其它项目的预设

**我选N**

(7). 看到🎉 Successfully created project xzvue.

👉 Get started with the following commands:

$ cd xzvue

$ npm run serve

说明安装成功！

6. 运行脚手架项目: (中午我发了: xzvue\_start\_with\_axios.zip)

(0). 将xzvue\_start\_with\_axios.zip解压到当前文件夹

(1). 用vscode打开解压出来的xzvue文件夹

(2). **右键单击package.json文件，选择在集成终端打开**

(3). 当弹出的窗口中出现.../xzvue > 输入**npm run serve**

(4). 当看到:

DONE Compiled successfully in 3018ms 15:24:52

App running at:

- Local: http://localhost:8080/

说明2件事:

a. 当前项目的代码编译通过，都正确

b. 这个前端项目已经运行在http://localhost:8080地址下！可用浏览器访问！

(5). 打开网页: 2种:

a. **按住ctrl点Local:右边的链接**

b. 手**工打开浏览器在地址栏里输入http://localhost:8080 回车**

(6). 脚手架代码是热编译: 每次修改后，不用频繁手动关闭，再启动。只要我们修改后，**Ctrl+S一保存**，**脚手架就自动帮我们重现编译了**

对应小程序视频: 小程序->在线->VUE->day04 3. vue脚手架文件夹结构

7. 脚手架项目的基本结构: 脚手架文件夹结构，就是SPA应用的结构，只不过更细致而已

(1). 有一个唯一完整的HTML文件

a. <div id="app">  
 b. <router-view>  
 c. new Vue({el:"#app", router})

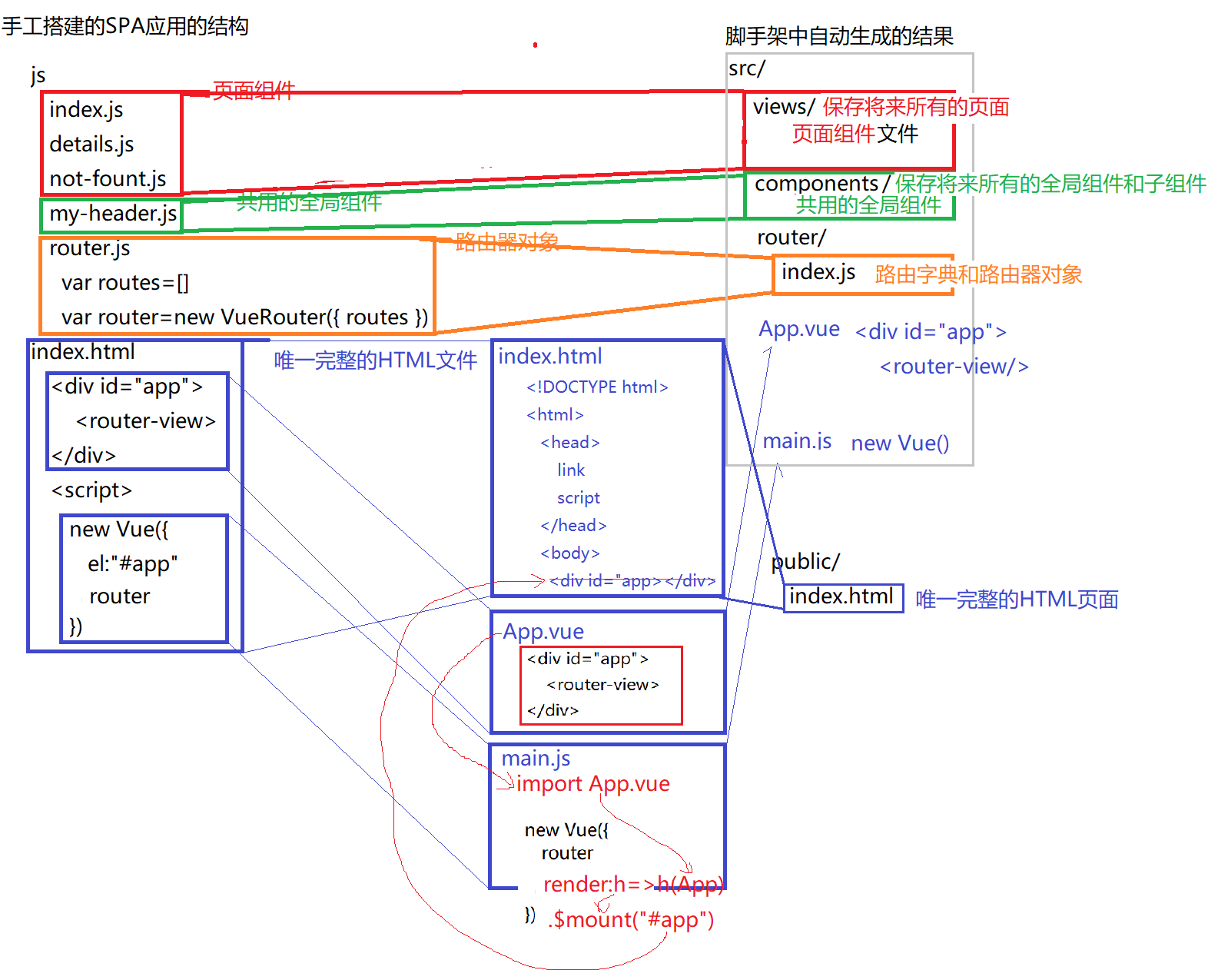
(2). 为每个"页面"分别创建页面组件文件，其中包含这个页面的内容

(3). 有router.js文件，包含路由器对象和路由字典数组

a. var routes=[ ... ]

b. var router=new VueRouter({ routes })

(4). 为多个页面共用的全局组件: 比如页头，创建专门的组件文件



8. 脚手架项目的运行过程:

(1). 程序的入口在**main.js**，先执行new Vue()

(2). main.js中，用**import**引入了**App.vue**文件中的HTML内容

(3). new Vue()读取App文件中的内容，并用App中的HTML内容代替index.html中的空的<div id="app">

(4). 还是路由器对象router监视地址栏变化。只要地址栏变化，就会去路由字典中查找对应的组件。

(5). router会将找到的组件对象，替换<router-view>所在的位置

9. .vue文件:

(1). 什么是: **集中保存一个页面或组件的HTML+JS+CSS内容的特殊的文件**

(2). 如何: 每个.vue文件由3部分组成:

a. **<template>标签**: 专门放这个页面或组件的HTML内容

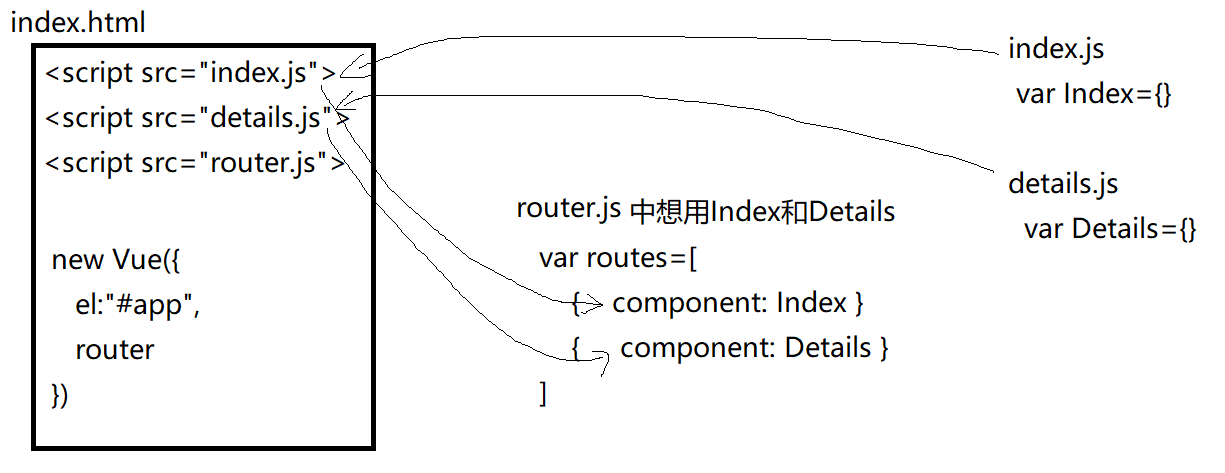
b. **<script>**标签: 专门放这个页面或组件的js内容/组件对象

c. **<style>**标签: 专门放仅这个页面或组件内使用的css内容

10. vue脚手架中的**模块化开发**:

(1). 旧的<script>引入方式的缺点: 不直观！

所有页面都需要先引入index.html中，再分给其它需要的组件使用

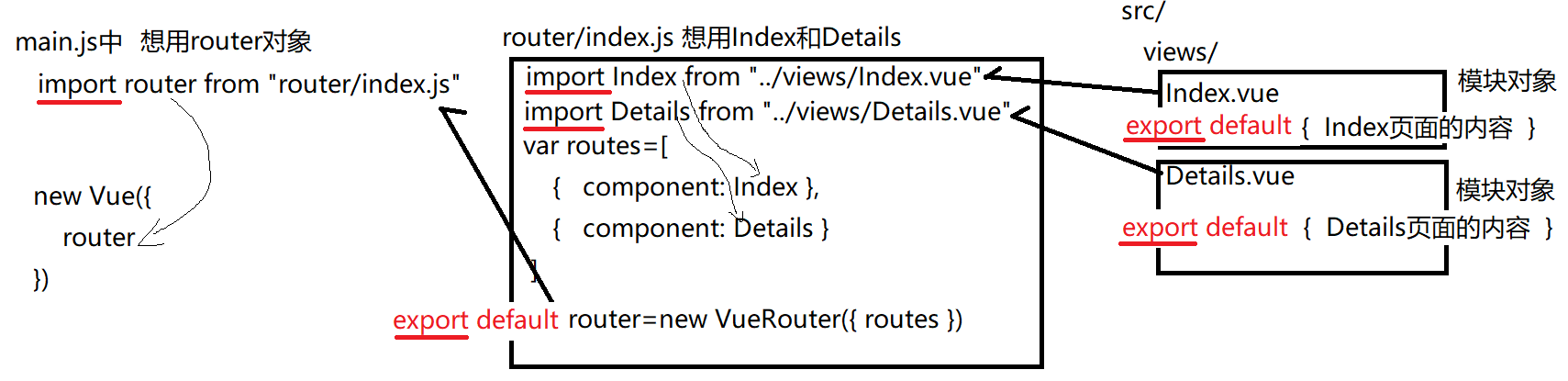


(2). 脚手架中采用的**模块开发**:

a. **每个组件文件，都是一个模块对象**

b. 任意一个组件想使用另一个模块对象中的内容，都需要**:import 对象名 from "相对路径"**

c. 强调: 只有用**export default抛出的模块对象**，才能被其它模块用import引入！



11. **如何定义和使用全局组件**: 比如页头:

(1). 在src/components文件夹下创建全局组件的**.vue文件**

|  |
| --- |
| <template>    <div id="nav">    <!--将App.vue中<router-view>上方的div剪切到这里-->      <router-link to="/">Home</router-link> |      <router-link to="/about">About</router-link>    </div>  </template>  <!--如果组件对象中没有js内容，则<script>部分可省略！-->  <!--<script>  export default {    }  </script>-->  <style>  /\*将App.vue中<style>里#nav开头的所有样式剪切到这里\*/  #nav {    padding: 30px;  }  #nav a {    font-weight: bold;    color: #2c3e50;  }  #nav a.router-link-exact-active {    color: #42b983;  }  </style> |

(2). 在**main.js中引入**页头组件（暂时还不是全局组件！）

import 自定义组件对象名 from "./components/页头组件.vue"

(3). **在main.js中new Vue()之前**，用Vue.component()将引入的页头组件对象**转化为一个全局组件**

**Vue.component("标签名", 自定义组件对象名)**

|  |
| --- |
| import Vue from 'vue'  //从App.vue文件中引入根组件的HTML内容  import App from './App.vue'  import router from './router'  import store from './store'  import MH from "./components/MyHeader"  Vue.config.productionTip = false  Vue.component("my-header", MH);  new Vue({    router,    store,    //读取App组件中的HTML内容    render: h => h(App)  }).$mount('#app')//用App中的HTML内容代替index.html中空的<div id="#app"> |

(4). 结果: **可以在**项目中**任意一个组件**的html中**使用全局组件**标签引入全局组件的内容

比如: 在App.vue中:

|  |
| --- |
| <template>    <div id="app">      <my-header></my-header>      <router-view/>    </div>  </template>  <style>  #app {    font-family: 'Avenir', Helvetica, Arial, sans-serif;    -webkit-font-smoothing: antialiased;    -moz-osx-font-smoothing: grayscale;    text-align: center;    color: #2c3e50;  }  </style> |

12. **配置公共的axios组件**：

(1). 问题: 脚手架代码不包含axios的

(2). 解决: **单独安装**axios模块: **npm i -save axios**

(3). 问题: 所有组件中都需要用axios发送ajax请求

(4). 解决: 在**main.js中new VUe()之前**配置好一个axios对象，放入Vue的原型对象中。所有组件中都可用**this.axios**访问Vue原型对象中的axios对象。

其实，无论根组件(new Vue())还是全局组件Vue.component()，还是子组件，都是Vue类型的子对象。虽然我们看不见每次new Vue()但是底层确实是用new Vue()创建的每个组件。所以，放入Vue的原型对象中的成员，所有vue组件都能通过**this.**方式访问！

(5). 如何: **main.js中 new Vue()之前:**

**import axios from "axios"** //安装在node\_modules中的模块，引入时不需要写路径

**axios.defaults.baseURL="服务器端基础路径"**

**Vue.prototype.axios=axios;**

三. **懒加载:**

对应小程序视屏: 小程序->在线->VUE->day05 -1 懒加载

1. 如果不用懒加载，问题:

(1). 所有的组件内容将会被编译为一个巨大的app.js文件

(2). 在首屏加载时浏览器被迫要下载整个app.js文件，又大，又慢

src/router/index.js中

|  |
| --- |
| ... ...  import Home from '../views/Home.vue'  import About from "../views/About.vue"  ... ...  const routes = [    { path: '/', component: Home },    { path:"/about", component:About }  ]  ... ...  运行结果: f12 network |

2. 解决: 2种:

(1). 脚手架中默认的一种: **异步延迟下载**

a. 不要过早的引入不用的组件

b. 在路由字典中使用匿名函数方式，临时引入组件对象

{ //只有当用户尝试访问/about路径时，才临时引入About组件内容

path: '/about',

name: 'about',

component: () => import(/\* webpackChunk(分段)Name: "about" \*/ '../views/About.vue')

/\* 相当于

function(){

return import '../views/About.vue'

}

\*/

}

其中: **/\* webpackChunk(分段)Name: "about" \*/ 在为分段的组件文件起名**

c. 结果: 2个

1). about组件的内容不会打包在首屏加载的app.js中

2). 框架会将about组件的内容**额外**新建一个js文件保存，文件名就是ChunkName注释中的名字

3). 当浏览器加载首屏内容app.js时，会在另一个任务中**异步延迟下载**about.js文件

i. 不会影响首屏内容app.js的加载速度

ii. 也不会影响后续用户访问about的加载速度（因为在用户切换到about之前，已经悄悄下载到本地了！）

d. 问题: 即使用户不想访问其余页面，也会在后台偷偷下载，**偷跑流量**！

src/router/index.js

|  |
| --- |
| ... ...  import Home from '../views/Home.vue'  //因为首屏暂时用不到About页面，所以不要过早引入  //如果不引入，打包时，就不会将About页面的内容打入App.js中  // import About from "../views/About.vue"  ... ...  const routes = [    { path: '/', component: Home },    // { path:"/about", component:About }    { //这里是懒加载，暂时不要问，明天讲      path: '/about',      name: 'about',      // route level code-splitting      // this generates a separate chunk (about.[hash].js) for this route      // which is lazy-loaded when the route is visited.      component: () => import(/\* webpackChunkName: "about" \*/ '../views/About.vue')      /\*      function(){        return import '../views/About.vue'      }      \*/    }  ]  运行结果: |

(2). **彻底懒加载:** 如果用户不访问，就绝对不会下载其余组件的内容！

a. 以上两步a和b依然要做：不要过早引入，使用匿名函数临时引入！

b. **新增设置**: 禁**止其他组件内容异步延迟加载:**

1). 在脚手架项目的**根目录新建vue.config.js**

2). 其中包含以下代码:

**module.exports={**

**chainWebpack:config=>{**

**config.plugins.delete("prefetch")**

} 删除 提前获取

}

c. 结果:

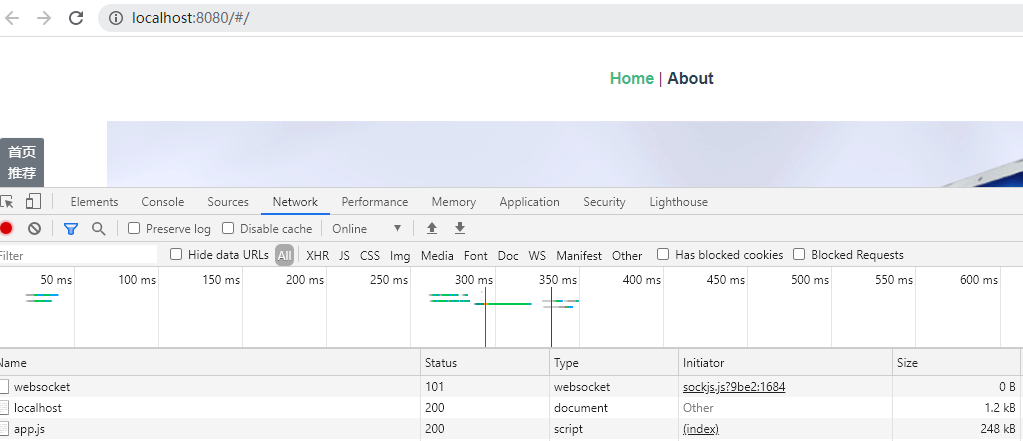
1). about组件的内容不会打包在首屏加载的app.js中

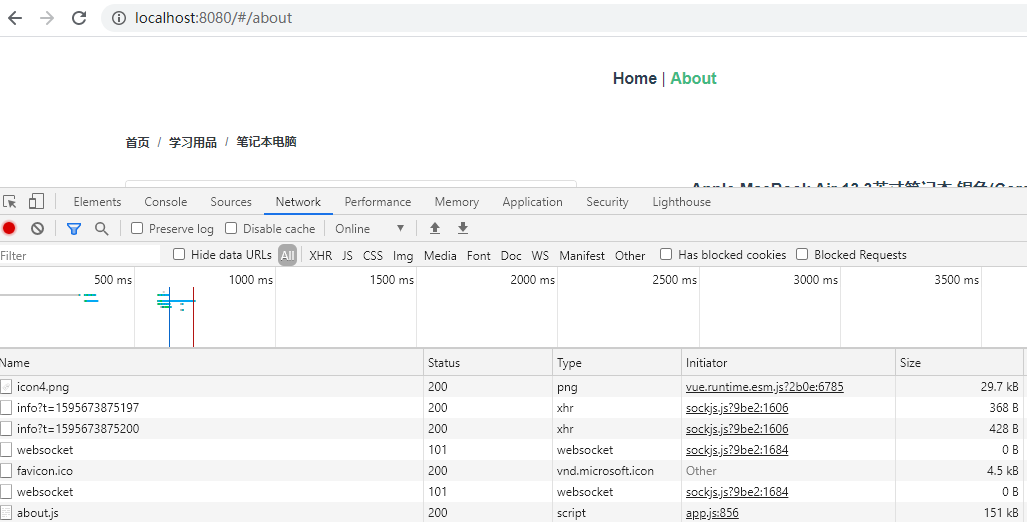
2). 框架会将about组件的内容额外新建一个js文件保存，文件名就是ChunkName注释中的名字

3). 只要用户不访问about地址，就永远不下载about组件的内容。除非用户要访问about地址了，才临时下载about的内容

d. 优点: 节约流量

e. 缺点: 加载about页面时，因为不在本地，需要临时下载，所以稍慢





总结:

7. vue生命周期4个阶段 8个钩子函数

beforeCreate(){ ... }

(1). 创建(create)

created(){ ... }

beforeMount(){ ... }

(2). 挂载(mount)

mounted(){ ... }

beforeUpdate(){ ... }

(3). 更新(update)

updated(){ ... }

beforeDestroy(){ ... }

(4). 销毁(destroy)

destroyed(){ ... }

8. 只要希望重用一块独立的功能区域就用组件:

(1). 定义组件

Vue.component(`组件标签名`,{

template:`HTML内容片段`,

data(){ return { 变量 } },

//其余和new Vue()完全相同

})

(2). 在HTML中使用自定义组件

<组件标签名/>或双标记也行

(3). 原理: new Vue()扫描到自定义组件标签，

a.组件的template中的HTML内容代替页面中<组件标签>位置。

b. 并为这个小区域专门创建一个缩微版的vue类型对象。

1). 调用组件的data()函数为当前组件副本创建一个专属数据对象副本。

2). 引入组件对象中的methods等其他内容到当前组件对象副本中

9. 组件化开发:

(1). 三种组件:

a. 根组件: new Vue()

b. 全局组件: Vue.component(...)

c. 子组件: 3步

1). var 子组件对象名={

内容必须符合组件的要求

}

子组件对象名必须是驼峰命名

2). 父组件对象中:{

... ...

components: { 子组件对象名, ... ,... }

}

子组件对象名必须是驼峰命名

3). 父组件template中用<子组件标签名/>引入子组件内容

components会将子组件对象名的驼峰命名自动翻译为-分隔

所以, 使用子组件标签时，要用-分隔多个单词

(2). 组件间传参: 父给子

a. 父组件给:

<子组件 :自定义属性名="父组件变量">

b. 子组件取:

props:["自定义属性名"]

结果: 在子组件内，props中的"自定义属性名"与子组件自己data中的变量用法完全相同！

10. SPA

(1). 3步:

a. 先创建唯一完整的HTML页面

1). 包含vue基本的页面结构

<div id="app"> new Vue({el:"#app"})

2). 引入所有必要的文件和组件

vue-router.js, 其它页面或组件文件, router.js路由器对象所在文件

3). <div id="app">中用<router-view/>为今后页面组件预留空位

b. 再为每个页面组件创建独立的文件。每个页面组件其实都是一个子组件

c. 创建router.js文件，创建路由器对象

1). 创建路由字典对象:

var routes=[

{path:"/", component:首页组件对象名},

{path:"/相对路径" , component: 其它页面组件对象名},

{path:"\*", component: 404页面组件对象 }

]

2). 创建路由器对象，并将路由字典对象转入路由器对象中  
 var router=new VueRouter({ routes })

3). 将router对象加入到new Vue()中

回到唯一完整的HTML页面中: new Vue({ el:"#app", router })

(2). 页头等全局组件：

a. 创建独立的文件保存页头组件的内容

b. 使用Vue.component("my-header",{ ... })将页头创建为全局组件

c. 在唯一完整的HTML页面中引入页头组件文件

d. 使用页头组件标签<my-header/>: 2种:

1). 如果所有页面都有统一的页头:

就放在唯一完整的html页面中<router-view>外部上方

2). 如果有的页面有页头，有的页面没有页头:

就只放在需要页头的组件中的template中

(3). 路由跳转: 2种:

a. html中: <router-link to="/相对路径">文本<router-link>

b. js中: this.$router.push("/相对路径")

(4). 路由传参:

a. 修改路由字典:

{path:"/相对路径/:自定义参数名", component:页面组件对象, props:true}

b. 跳转时:

<router-link to="/相对路径/参数值"

或

this.$router.push("/相对路径/参数值")

c. 下一个页面接:

1). props:[ '自定义参数名' ]

2). 可将"自定义参数名"用作绑定或程序中都行

11. 脚手架文件夹结构:

(1). 唯一完整的HTML页面: 一分为三:

a. public文件夹

1). 图片img文件夹放在public文件夹下

2). 第三方css的压缩版和第三方js的压缩版都放在public文件夹下

3). 唯一完整的HTML文件index.html中，head中引入第三方的css和js

b. src/App.vue

1). <template>下只包含公共的页头组件和<router-view>

2). <style>下包含所有网页都要用到的公共css样式，比如css重置代码

c. src/main.js

1). import引入App.vue，router，axios，以及其他全局组件

2). 将全局组件对象转为真正的全局组件: Vue.component( "组件标签名", 全局组件对象 )

3). 配置axios并放入原型对象中:

axios.defaults.baseURL="服务器端基础路径"

Vue.prototype.axios=axios;

(2). 为每个页面创建.vue组件文件，都放在src/views文件夹下。每个.vue文件中:

a. <template>标签内，包含这个页面的HTML内容

b. <script>export default{ ... }</script>中包含组件对象的js内容。

c. <style>标签内包含仅这个页面组件内才使用的css

d. <template>中的HTML内容以及<script>export default{...}</script>中的js内容，和前四天将的是完全一样的写法，绑定，指令，函数，生命周期，axios请求等都一样。前四天怎么用，这里就怎么用。

(3). 路由字典和路由器对象，在src/router/index.js文件中

a. 仅import首页组件对象，不要过早引入其它页面组件

b. 路由字典中首页组件: { path:"/", component:首页组件对象}

c. 其余页面组件都做成懒加载:

{

path: '/相对路径',

component: () => import(/\* webpackChunkName: "组件名" \*/ '../views/其它页面组件.vue')

}

(4). 全局组件都放在src/components文件夹下，每个全局组件.vue文件中。但是，全局组件必须在main.js引入，并用Vue.component()转化为真正的全局组件，才能在其它组件的HTML中使用。

(5). 运行时: 路由器router对象监视浏览器地址栏路径变化，并查找对应的页面组件内容，先代替App.vue中的<router-view>，然后main.js再将包含要加载的页面内容的App.vue组件内容，替换到唯一完整的index.html中空<div id="app">位置。

今日对应小程序视频列表:

小程序->在线->VUE->day04 1. SPA单页面应用

2. 使用vue/cli工具创建学子商城项目

3. vue脚手架文件夹结构

小程序->在线->VUE->day05 -1 懒加载

作业:

1. 复习今日问题清单: 小程序->在线->VUE->day04、day05

2. 看小程序视频使用vue脚手架完成学子商城首页、详情页

先下载: 小程序->在线->VUE->day05 0. VUE版学子商城 起始空项目 中

旧js项目: public.zip

vue脚手架起始项目: xzvue\_start\_with\_axios.zip

观看并跟着做: 小程序->在线->VUE->day05

1. 学子商城首页

2. 学子商城详情页 。。。（不带放大镜）

3. 学子商城详情页 。。。（放大镜效果）

家里环境xampp和nodejs能用，且第一阶段学的好的同学:

可下载0. VUE版学子商城 起始空项目 中xzserver.zip，用自己的服务器端代码

家里环境xampp和nodejs不能用，或者第一阶段学的不好的同学：

无需下载xzserver.zip

视频中所有axios请求的服务器端接口地址，都改为我的新浪云服务器地址，就不需要你再运行服务器端了。也不需要你了解任何服务器端知识了:

http://localhost:5050/index 改为 http://xzserver.applinzi.com/index

http://localhost:5050/details 改为 http://xzserver.applinzi.com/details?lid=xxx