提前准备:

1. 今天的历届师兄师姐问过的经典的高频的问题清单都已经整理在小程序->首页->VUE->day04里

2. 今天的课程中所需的单词列表都已经整理在小程序->单词->VUE->day04里

3. 今天的重点视频、扩展高频鄙视题视频已经整理在小程序->在线-> VUE ->day04里

如果不知道问什么: 谁问的越多！谁培训费花的越值！

1. 笔记中或画图中那句话或者那个位置不明白，立刻问！

2. 案例中哪行代码或注释不明白，立刻问！

3. 课下自己做项目，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！——公众号: 前端大全 定期总结大厂笔试题

5. 看小程序问题清单时，遇到看不懂的问题，立刻问！

正课:

1. SPA

2. 脚手架

3. 脚手架的文件夹结构

4. .vue文件

5. ES6的模块化开发

6. 懒加载

一. SPA: Single Page Application

单 页面 应用

1. 什么是单页面应用: 整个应用程序只有一个完整的HTML页面。其它所谓的"页面"，其实只是页面中一个片段/组件而已。所谓的"页面跳转"，其实是在一个HTML页面中切换不同的片段内容/组件而已。

2. 为什么: 多页面应用 vs 单页面应用

|  |  |  |
| --- | --- | --- |
|  | 多页面应用 | 单页面应用 |
| 请求次数 | 请求次数多，因为每更换一次页面都要向服务器重新发起请求 | 请求次数少，因为首次请求就将所有页面组件一次性下载到本地，之后及时切换"页面"，只不过是更换本地不同的组件展现而已，无需向服务器重新发送请求 |
| 加载效率 | 效率低，因为每更换页面，都要重建整棵DOM树 | 效率高, 因为每次更换页面不需要重建整棵DOM树，只更新原DOM树中部分组件节点即可。 |
| 公共资源重复使用 | 即使所有页面都需要的css和js文件，每次更换页面时都需要重新下载，增加了请求次数，且浪费了流量！ | 即使所有页面都需要的css和js文件，仅在首次加载第一个页面时请求一次，之后更换页面，因为引用公共文件的内容部分没有改变，所以不会重新请求这些多页面共用的css和js，就减少了请求次数，且节约了流量！ |
| 页面过渡动画 | 多页面应用，因为同一时刻，只能保存一张网页，所以无法实现页面过渡动画 | 单页面应用，因为所有组件已经在客户端内存中。主页面完全由可能同时加载两个组件，定义两个组件的移动动画效果。 |

3. 缺点: 首屏加载太慢！

(1). 因为: 即使用户不一定会看后续页面，第一次也都要把所有页面组件的内容都要下载下来——首次下载内容太多！

(2). 解决:

a. 异步延迟下载: （脚手架默认选择）

1). 打包时，不要把所有组件打包在一个大的文件中，应该每个组件分开打包为单独的文件。

2). 首屏加载时，只强制加载第一个组件的内容。后续组件由底层程序异步下载。

3). 优点: 不影响首屏加载速度，又能实现单页面应用的效果

4). 缺点: 因为即使用户不看其他组件的内容，也会下载其他组件，浪费网络流量。

b. 懒加载: (可通过配置脚手架实现)

1). 打包时，不要把所有组件打包在一个大的文件中，应该每个组件分开打包为单独的文件。

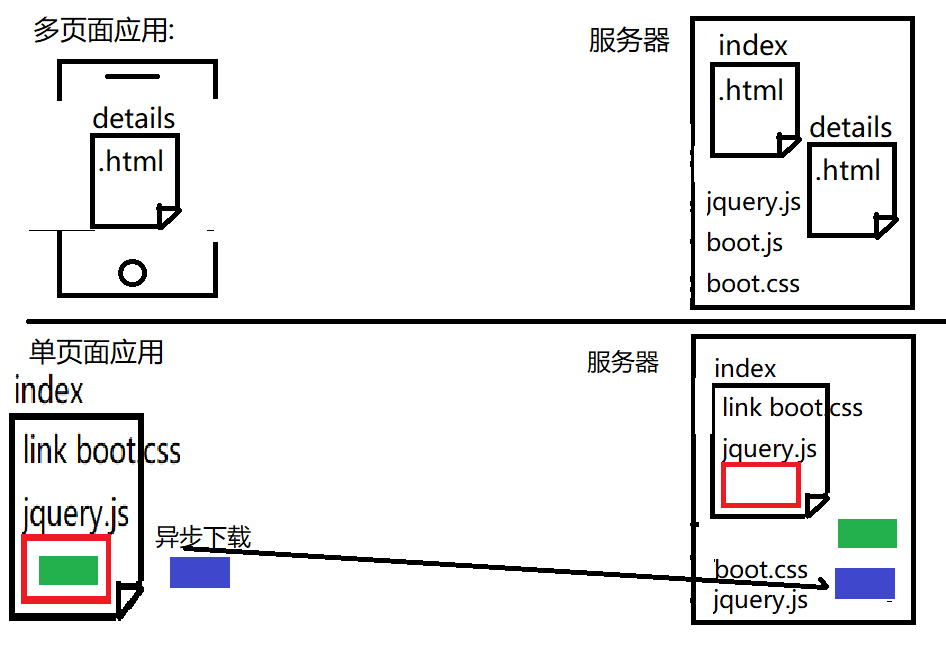
2). 首屏加载时，不会下载其它任何组件。

3). 只有当用户确实切换到其他某个组件时，才临时下载用户想看的组件

4). 优点: 省流量

5). 缺点: 下载速度比加载本地组件稍慢。

6). 但还是单页面应用！因为之后下载的只是页面中部分的组件内容而已，而不是完整的HTML页面。



4. 何时: 今后几乎所有前端框架，都使用的是单页面应用方式开发的！

5. 如何:

(1). 先创建一个完整的html页面，包含<div id="#app">和new Vue()

a. 还要引入<script src="js/vue-router.js">

b. 还要在<div id="app">内定义停车位: <router-view></router-view>

c. 还要在new Vue()中添加router成员

(2). 再为每个页面分别创建一个子组件对象:

(3). 创建路由器，定义路由字典数组

(4). 用live server运行唯一完整的index.html网页

(5). 结果:

a. 开局看到首页内容

b. 修改地址栏最后一个/后的地址为details，按回车，看到详情页内容

6. 全局组件:

(1). 定义一个全局组件

(2). 所有页面都需要的全局组件(页头)，应该放在唯一完整的html中<router-view>外部。放在<router-view>外部的组件，所有页面默认都能显示。

(3). 如果有些页面有，有些页面没有的组件，应该放在各个组件内。不应该放在<router-view>外部

7. NotFound组件:

(1). 其实所有网站都应该提供友好的404页面

(2). 不同: 每个普通页面都有一个唯一的地址与之对应，但是404页面应该是除了正确的地址之外，其余所有地址都应该可以跳到404页面

(3). 如何:

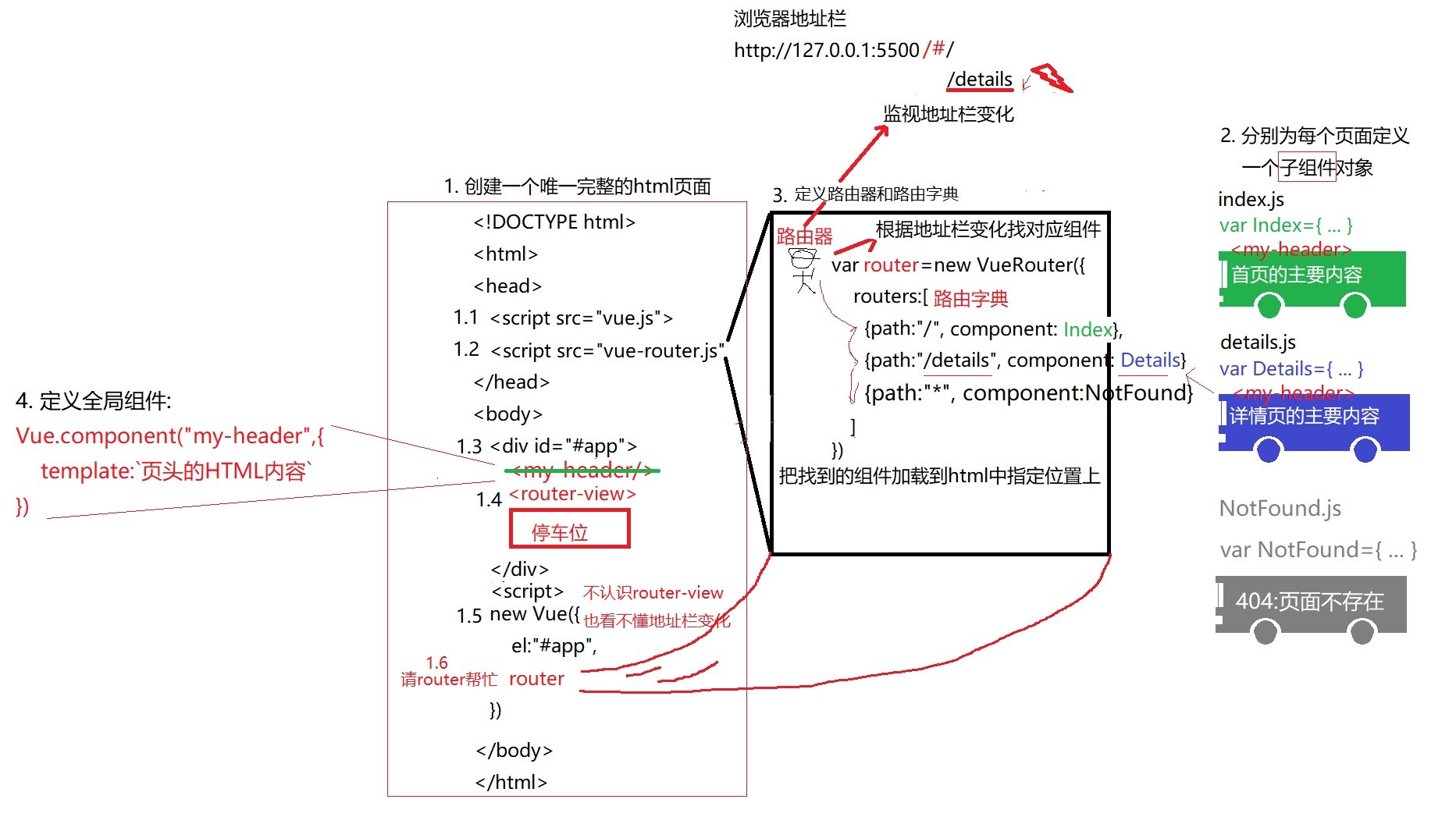
a. 定义404页面组件

b. 在routes数组中末尾添加: {path:"\*", component: 404组件}

其中:

1). \*代表所有其余路径的意思

2). 因为当地址栏发生变化时，路由器router是按从上向下的顺序匹配routes数组中每个路径的。如果\*凡在前边，则永远匹配，就无法跳转到其他页面了。



8. "页面"间跳转:

(1). HTML中: 用<router-link>代替a

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

b. 强调: to中的相对地址不要加#

c. 原理: router-link to会被自动翻译为<a href="#/xxx">

d. 链接激活样式:

1). 所有和当前地址栏中相对路径吻合的router-link元素上都会被自动加上class="router-link-exact-active"

2). 我们只要在css中定义同名的class选择器，就可以无代码定义链接样式的激活样式.

(2). Js中: this.$router.push("/相对路径")

9. "页面"间传递数据:

(1). 修改路由字典数组中的路由对象定义:

a. { path:"/相对路径/:变量名", component: 页面组件对象名, props:true }

b. 其中:

1). 将来传值都是通过地址栏里的url传递的。:变量名，就是传值时所使用的变量

2). props:true，意思是让地址栏中的变量名以及变量值自动成为下个页面的组件的props中的自定义属性。可将传过来的变量值当做普通的变量一样在下个页面中用于绑定。

3). 强调: 一旦一个路由地址要求携带参数，则再用不带参数的地址就进不去这个路由地址了！

(2). 下个页面组件中使用props接住传来的同名的变量值

props:[ "变量名" ]

(3). 跳转时:

a. HTML跳转: <router-link to="/相对路径/变量值"

强调: 不要加:

b. js中跳转: this.$router.push("/相对路径/变量值")



10. 示例: 构建一个简化版的SPA应用

SPA.html ——唯一完整的HTML页面

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <meta http-equiv="X-UA-Compatible" content="ie=edge">    <link rel="stylesheet" href="SPA.css">    <title>Document</title>    <script src="js/vue.js"></script>    <script src="js/vue-router.js"></script>    <script src="Index.js"></script>    <script src="Details.js"></script>    <script src="NotFound.js"></script>    <script src="router.js"></script>    <script src="MyHeader.js"></script>  </head>  <body>  <div id="app">    <!-- <my-header></my-header> -->    <router-view></router-view>  </div>  <script>  var vm=new Vue({    el:"#app",    router  })  </script>  </body>  </html> |

SPA.css

|  |
| --- |
| .router-link-exact-active{    background-color:red;    color:#fff;    font-weight:bold;    padding:5px 10px;  } |

Index.js ——两个页面组件之一

|  |
| --- |
| var Index={    template:`<div class="index">      <my-header></my-header>      <h1 style="color:green">这里是首页</h1>      <ul>        <li><router-link to="/details/1">查看1号商品的详情</router-link></li>        <li><button @click="goToDetails(5)">查看5号商品的详情</button></li>      </ul>    </div>`,    methods:{      goToDetails(lid){        this.$router.push(`/details/${lid}`)                        // /details/5      }    }  } |

Details.js——两个页面组件之一

|  |
| --- |
| var Details={    template:`<div class="details">      <my-header></my-header>      <h1 style="color:blue">这里是详情页</h1>      <h2>显示{{lid}}号商品的详情</h2>    </div>`,    props:["lid"]  } |

NotFound.js——404组件

|  |
| --- |
| var NotFound={    template:`<div class="not-found">      <h1 style="color:gray">404:您所查找的页面不存在</h1>      <h2 style="color:gray">{{n}}s后自动返回首页</h2>    </div>`,    data(){      return {        n:3,        timer:0      }    },    mounted(){      //必须用箭头函数保持内外this一致      this.timer=setInterval(()=>{        this.n--;//每隔1s减1，        if(this.n==0){//减到0时，返回首页，停止定时器          clearInterval(this.timer);          this.n=3;//将n重置为3          this.$router.push("/");        }      },1000)    }  } |

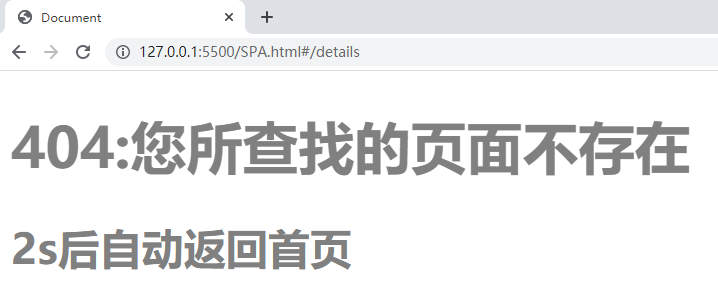
router.js —— 路由器对象和路由字典数组

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

MyHeader.js —— 页头的全局组件

|  |
| --- |
| Vue.component("my-header",{    template:`<div class="my-header">      <h1>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>`  }) |







二. 脚手架:

1. 什么是脚手架: 已经具有核心功能的标准化的半成品项目

2. 为什么:

(1). 标准化: 凡是用vue脚手架工具生成的脚手架代码结构完全一样！

(2). 半成品: 因为脚手架已经包含了大量核心代码，开发人员无需重复配置项目，只要专注于自己个性化定制的部分内容即可！省事！效率高！

3. 何时: 今后所有企业中的前端项目都是在脚手架代码基础上开发的

4. 如何: 2步:

(1). 先安装用于反复生成脚手架的命令行工具: (一只老母鸡) (不需要每个项目重复做)——在任意目录下都行:

npm i -g @vue/cli (如果家里网不行！不要试！)

(如果非要试，看小程序->在线->JSCORE->day01中的第三阶段软件安装手册)

成功: 在结尾看到+ @vue/cli@4.3.0

或 运行vue -V 看到版本号@vue/cli 4.2.3也算成功

(2). 再用脚手架命令行工具反复生成项目脚手架源代码(老母鸡下蛋)(每做一个新项目，都要重新做一遍)

在想要创建项目的文件夹目录下运行: vue create 项目名

结果: 在这个文件夹目录下从无到有创建出一套具有核心功能的vue脚手架项目源代码

5. vue create 项目名: 安装步骤:

(0). 如果本地安装的@vue/cli工具不是最新版，可能出现提示:

New version available 4.2.3 → 4.3.0

有新版本 可用

│ Run npm i -g @vue/cli to update!

运行 升级

建议: 如果本地的@vue/cli可用，就不要冒险升级！

(1). 第一步: Please pick a preset: (Use arrow keys)

请 选择一种预设 使用方向键

default (babel, eslint)

> Manually select features

(2). Check the features needed for your project: (Press <space> to select,

检查 以下功能 哪些是你的项目需要的 按 空格 选择

>(\*) Babel —— 将ES7,ES6等时髦的代码自动翻译为ES5的等效代码，让大部分浏览器都能接受

( ) TypeScript

( ) Progressive Web App (PWA) Support

(\*) Router —— 单页面应用的核心组件

(\*) Vuex —— 下个阶段学！

( ) CSS Pre-processors ——如果你的项目用了sass或less才选这项

( ) Linter / Formatter ——代码格式检查工具

( ) Unit Testing

( ) E2E Testing

(3). Use history mode for router?

是否使用history模式作为路由格式

(Requires proper server setup for index fallback in production) (Y/n) N

需要 提供服务器端支持 首页重定向 在生产环境中 不使用！

普通路由模式: http://127.0.0.1:5500/index.html/#/details

浏览器看到#/xxx时，是不会回发给服务器端。而是在客户端浏览器中就执行页面内跳转了

History模式: http://127.0.0.1:5500/index.html/details 去掉#

浏览器认为所有/xxx地址默认都是服务器端接口地址，都会回发到服务器，而不是请客户端的vue-router来解析。造成错误！

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

你更喜欢把Babel、ESLint等的配置放在那里？ 用方向键选择

In dedicated config files

放在各自独立的配置文件中

> In package.json

所有配置都统一放在一个package.json文件中

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

是否保存这套配置为将来项目的预设? N 不保存

(6). 等...

(7). 看到: 就算成功

🎉 Successfully created project xzvue.

👉 Get started with the following commands:

结果: 在当前目录下创建了一个执行项目名的文件夹，其中包含vue项目脚手架核心代码。

(如果自己开发项目或做实验，不用git，可删除.git文件夹)

(8). 首次运行半成品脚手架项目:

(~~从今天开始live server 彻底退出历史舞台！~~)

a. 用vs code打开刚创建的新项目文件夹

b. 右键点击package.json文件，选择"在终端/命令行中打开"

c. 在vs code弹出的对话框中，等待出现输入光标的提示后，再输入npm run serve

d. 结果: 做两件事:

1). Starting development server...

启动一个 开发 服务器 （简易服务器来运行前端网页）

2). Compiled successfully

编译 成功

i. 将代码中浏览器可能不认识的ES6，ES7的时髦代码转化为ES5的代码，让大部分浏览器可接受

ii. 压缩和打包代码！放入临时的开发服务器中，允许开发人员测试网页

e. 强调: 在vs code中打开npm run serve，今后每修改一下代码，只要一保存，就会自动重新编译运行。所以，~~不用每修改一次，都重复运行npm run serve~~

(9). 浏览当前项目中的"网页":

vs code中，按住ctrl点命令行中出现的链接地址: http://localhost:8080

结果: 在浏览器中看到脚手架代码默认自带的首页

点about可以切换首页和关于页面，说明包含两个页面以及一个公共的页头，同时还配置好了路由器。

三: 脚手架项目的文件夹结构: 脚手架项目的结构和SPA应用的结构完全一样

1. 也只有一个唯一完整的HTML页面，但是一个页面被拆成了三个文件:

public文件夹/index.html —— 唯一完整的HTML页面

src文件夹/App.vue —— <div id="app">

<router-view></router-view>

src文件夹/main.js —— new Vue({ ... })

运行时: index.html+App.vue+main.js，将共同组合成唯一完整的HTML页面

2. 也要为每个页面建立一个组件文件，在组件文件中集中定义一个组件的HTML+JS+CSS。但是所有页面组件的文件应该放在src文件夹/views文件夹中

脚手架中默认带了两个页面Home.vue和About.vue，可以自己试着用前三天vue的只是修改一下试试。

3. 也要有一个router对象和一个路由字典数组，脚手架已经默认建好了:

src文件夹/router文件夹/index.js中

4. 如果创建全局组件，需要将全局组件的文件放在src文件夹/components文件夹下

四. .vue文件:

1. 什么是: 包含一个组件的HTML+JS+CSS的vue专门的文件——只有vue脚手架认识，离了脚手架不能用！

2. 为什么: 为了把一个组件的HTML+JS+CSS都写在一个文件中，便于维护！

3. 何时: 今后在脚手架中只要创建一个组件(不管是页面组件，还是子组件，还是全局组件)，都要创建为.vue文件

4. 如何: 每个.vue文件中包含3部分

(要想有.vue文件内容提示，必须安装vscode vetur插件)

(1). <template>

<唯一父元素 class="组件class名">

组件的HTML内容

</唯一父元素>

</template>

(2). <script>//组件对象的js

export default {

~~template:` ... ` 作废了！提到前边<template>里了！因为这里没提示~~

data(){

return {

变量: xxx

}

}

}

</script>

(3). <style scoped> /\*scoped自动尽量屏蔽组件间样式冲突，但是不太好用\*/

仅当前组件内可用的css

</style>

五. ES6的模块化开发:

1. 脚手架中都是.vue文件或js文件，没地方写<script>引入

2. 所以脚手架采用的是模块化开发:

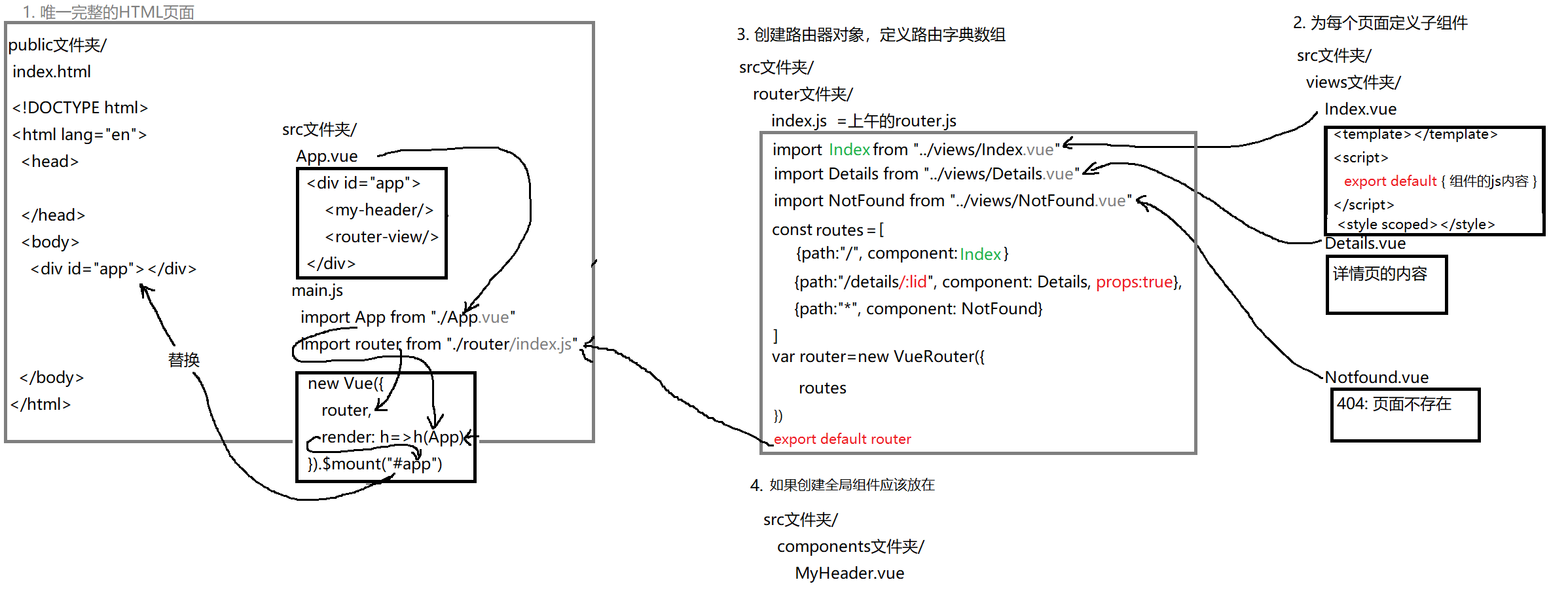
(1). 每个.vue文件默认都是一个模块对象

(2). 如果一个.vue文件或.js文件，想使用另一个.vue文件中的内容，必须通过:

import 别名 from ".vue文件相对路径"

将另一个.vue文件的内容引入到当前.vue文件内部，称为一个组件对象

(3). 如果.vue文件中包含js代码，则必须用export default{ ... }方式抛出，才能被别的组件引入



总结:

1. MVVM: 界面(View), 数据模型(Model), 视图模型(ViewModel)

2. VUE实现绑定的原理: 访问器属性+虚拟DOM树

(1). 什么是虚拟DOM树: 只保存可能发生变化的元素的简化版DOM树

(2). 好处: （为什么）

a. 小, 只保存可能发生变化的元素

b. 遍历快

c. 效率高: 只更新受影响的元素

d. 已封装DOM操作: 避免重复编码

3. new Vue({

el:"#app",

data:{ 变量 },

watch:{ ... },

computed:{ ... }, //调用时不加()，缓存结果，不会反复计算

methods:{ ... }, //调用时加()，不缓存结果，会反复计算

})

4. 组件的生命周期: 4个阶段:

beforeCreate()

创建(create)

created()

beforeMount()

挂载(mount)

mounted

beforeUpdate()

更新(update)

updated()

beforeDestroy()

销毁(destroy)

destroyed()

5. 绑定和指令:

(1). 元素内容可能发生变化: {{}} v-text

(2). 元素属性值可能发生变化: :

(3). 控制一个元素显示隐藏: v-show

(4). 两个元素二选一显示隐藏: v-if v-else

(5). 多个元素多选一显示隐藏: v-if v-else-if v-else

(6). 反复生成多个相同结构的元素: v-for :key 只更新一个元素即可，不必重复生成所有元素

(7). 事件绑定: @ $event

(8). 绑定HTML内容: v-html

(9). 防止用户短暂看到{{}}: v-cloak v-text

(10). 只在开始时绑定一次: v-once

(11). 阻止内容中的{{}}被编译: v-pre

(12). 如果想获得表单元素的内容: v-model

6. 绑定样式:

(1). 绑定内联样式:

<元素 style="不变的内联样式" :style="变量">

data:{

变量:{

Css属性名: "属性值",

... : ...

}

}

(2). 绑定class:

<元素 class="固定不变的class" :class="变量">

data:{

变量: {

Class名: true或false,

... : ...

}

}

7. 自定义指令:

Vue.directive("指令名",{

inserted(domElem){

... ...

}

})

8. 过滤器:

Vue.filter("过滤器名", function(oldVal){

return 加工后的新值

})

{{ 变量 | 过滤器(实参值) | ... }}

9. axios:

axios.defaults.baseURL="http://xzserver.applinzi.com"

(1). 发送get请求:

axios.get("服务器端接口地址",{

params:{ 参数1: 值1, ... : ... , ... }

}).then(result=>{

result.data

})

(2). 发送post请求:

axios.post("服务器端接口地址",{参数1: 值1, ... : ... , ... })

.then(result=>{

result.data

})

10. 创建全局组件:

Vue.component("组件名",{

template:`<唯一父元素>

... 组件的HTML内容 ...

</唯一父元素>`,

data(){

return {

变量: xxx

}

},

... ...,

components:{ 子组件对象名, 子组件对象名 }

})

使用时: <组件名></组件名>

11. 父子组件间传参: 2步

(1). 父组件的template中:

<子组件 :自定义属性="父组件的变量">

(2). 子组件中:

props:[ "自定义属性" ]

作业:

1. (必须)看小程序视频完成vue版学子商城首页:

小程序->在线->vue->day04 4. 学子商城首页

2. (扩展)继续看小程序视频学习高频笔试题: 模块化开发

小程序->在线->vue->day03 模块化(7个视频)

3. (扩展)看小程序视频学习vue路由高级用法: 路由嵌套

小程序->在线->vue->day04 1. SPA单页面应用 ...