正课

1. SPA vs 脚手架

2. ES6 模块化开发

3. axios

4. 组件生命周期

5. $router vs $route

6. 封装axios请求函数

一. SPA vs 脚手架

|  |  |  |
| --- | --- | --- |
|  | SPA | 脚手架 |
| 1. 创建唯一完整的html文件 | index.html <html>  <head>  <link>  <script>  </head>  <body>  <div id="app">  <router-view>  </div>  <script>  new Vue({  ... ...  })  </body> | public文件夹/  imgs/当前项目用到的所有图片  css/ 当前项目公用的第三方css，比如bootstrap.css  js/ 当前项目公用的第三方js，比如: jquery.js, bootstrap.js  index.html 唯一完整的html文件，只包含<div id="app">，不包含<router-view>和new Vue()  src/文件夹下  App.vue  <div id="app">  <router-view>  可能还包含所有网页公用的样式和公用的组件，比如页头.  main.js  new Vue({...})  总结: App.vue+main.js，放入index.html后，才相当于咱们做的完整的index.html |
| 2. 创建页面组件 | index.js var index={  template:`html片段`,  data(){ return {}},  methods:{ },  ... ... }  details.js  notFound.js | src/文件夹下  views/文件夹下  放所有页面组件  有几个页面，就要有几个组件  每个组件都是一个.vue文件。每个vue文件由三部分组成:  <template>  当前组件的HTML片段，要求只能有一个统一的父元素包裹。  </tempalte>  <script>  export default {  没有template，  组件结构:  data(){return {}},  methods:{...},  watch:{ ... },  computed:{ ... },  components:{ }  }  </script>  <style scoped>  仅当前组件使用的css  </style>  每个.vue文件，最后都会被编译为传统的html,css和js，才能被浏览器认识 |
| 3. 创建路由器 | 在index.html顶部用script src="index.js"引入页面组件对象  然后 router.js var routes=[  ...  ]; var router=  new VueRouter({routes}) | src/文件夹下  router/文件夹  index.js  先引入页面组件  import 页面组件 from "./views/页面组件.vue"  new VueRouter({[  路由字典  ]}) |
| 4. 将router加入new Vue() | 手动加入new Vue() | src/main.js 自动添加  new Vue({ router ,..}) |
| 5. 全局组件 | my-header.js  Vue.component() | src/文件夹下  components/  my-header.vue  每个组件的组成三部分，和页面组件的三部分是完全一样的 |
| 6. 自己编写的公用的css和js |  | src/文件夹下  assets/文件夹  css/ 自己编写的所有页面公用的css  js/ 自己编写的所有页面公用的js |

二. ES6的模块化开发:

1. 什么是: 将一个功能的代码，保存在一个模块中，通过引用的方式，使用。

2. 回顾:node中的模块化:

(1). node中每个.js文件，就是一个模块，但是模块内的代码是自有的。模块外默认用不了！

(2). 每个js文件中的代码(方法，属性)必须导出后，才能被其它模块使用:

module.exports={ 要导出的方法名和属性名 }

(3). 其它模块想用这个模块的成员，必须先引入，再使用:

var 变量=require("./模块所在文件的相对路径")

变量.成员

3. ES6中的模块化:

(1). ES6中也规定了模块化开发，但是浏览器都不支持！目前只有脚手架才支持。  
 (2). 和node一样，每个.js或.vue都是一个模块对象。每个模块对象，要想让别人使用，也必须先导出:

export default { js代码 }

(3). 其它模块要想使用这个模块的成员，也必须先引入后才能使用

import 变量名 from "./模块所在文件的相对路径"

4. 脚手架中的模块化:

(1). 每个页面或者组件都是一个.vue文件，每个.vue文件都是一个模块。要想让别人知道有这个模块，必须先导出才行:

views/index.vue中

<script>  
 export default {  
 data(){ ... },

methods:{ ... },

...   
 }  
 </script>

(2). 如果在路由器router.js文件中，想使用某个页面组件，必须先引入，再放到路由字典中:

import Index from "./views/index.vue"

new VueRouter({[

{path:"/", component:Index}

]})

(3). 全局组件:

a. 在定义全局组件时，和普通页面组件、子组件没有任何差别。其实也只是一个普通的对象模块。

components/MyHeader.js

<script>

export default {

... ...

}  
 </script>

b. 如果要让这个组件变成全局组件

1). 先将组件对象引入到main.js中:

main.js中

import MyHeader from "./components/MyHeader.vue"

2). 让组件对象变成全局组件

Vue.component("my-header",MyHeader)

MyHeader才会成为全局组件

三. Vue脚手架中的axios：

1. 安装: 在项目本地安装axios

右键单击package.json，选在终端或命令行打开，运行：

npm i -save axios

2. 配置: 在main.js中，在new Vue()之前

(1). 引入axios对象: import axios from "axios"

(2). 配置axios的基础路径: axios.defaults.baseURL="服务端接口基础地址"

(3). 配置axios携带cookie:

axios.defaults. withCredentials=true;

(4). 将axios对象，添加到Vue的原型对象中

Vue.prototype.axios=axios;

(5). 在所有组件中，如何调用axios:

this.axios.get(...)

四. 组件生命周期:

1. 什么是: 一个组件从创建，到加载完成的整个过程。

2. 何时: 只要希望在组件加载过程中，某个阶段自动执行一项任务时，就要用到生命周期。

类似于: 一个普通的HTML页面，加载过程会经历两个加载完成事件: DOMContentLoaded在仅DOM内容加载完就自动触发；window.onload在整个页面加载完才自动触发。

问题：如果希望一个VUE组件加载完成时，也能自动执行一个操作，应该怎么写？

3. 包括: 4个阶段:

(1). 创建阶段(create): 创建组件对象，创建data对象，但是，在这个阶段还未创建虚拟DOM树

可以操作data中的数据: 比如发送ajax请求

不可以执行DOM操作

(2). 挂载阶段(mount): 创建虚拟DOM树

既可以操作data中的数据，比如发送ajax请求

又可以执行DOM操作

================组件首次加载完成==============

(3). 更新阶段(update): 只要data中的数据被改变，就会自动触发更新阶段。

(4). 销毁阶段(destroy): 只有主动调用$destroy()方法销毁一个组件时才会自动触发——用的少

4. 每个阶段前后，都分别包含一个钩子函数: 共8个

|  |
| --- |
| 在组件加载过程中，指定执行的一种回调函数，称为钩子函数。不叫事件处理函数。 |

beforeCreate(){ ... 会在创建组件阶段前自动执行 ... }

(1). 创建阶段

created(){

... 会在创建组件阶段结束后自动执行 ...

比如: 发送ajax请求

}

beforeMount(){ ... 会在挂载阶段之前自动执行 ... }

(2). 挂载阶段

mounted(){

... 会在挂载完组件内容后自动执行 ...

比如: 发送ajax请求，还可以执行DOM操作

}

|  |
| --- |
| 如果路由跳转时，并未更换页面组件，而是在同一个页面组件中，仅更换部分值，则不会重复执行创建和挂载阶段的。导致放在created中和mounted中的axios请求，不会重复发送，也就无法自动获得新的查询结果。 |

beforeUpdate(){ ... 更新数据前自动执行 ...}

(3). 更新阶段:

updated(){ ... 在更新完数据后自动执行 ... }

beforeDestroy(){ ... 在销毁组件之前自动执行 ...}

(4). 销毁阶段:

destroyed(){ ... 在销毁组件后自动执行 ... }

五. $router vs $route

1. $router是路由器对象，专门执行跳转动作！

2. $route是保存地址栏中信息的对象。只要希望获得地址栏中的信息时，才适用$route。类似于bom中的location.href。

比如: 获得路由地址中的参数，不一定非要用props，还可以:

this.$route.params.lid

六. 封装axios请求函数:

1. 定义模块专门保存服务器端基础地址:

src/assets/js/config.js中:

var baseURL="http://localhost:5050"

export {

baseURL

}

2. 定义访问某个接口的函数:

src/assets/js/apis/index.js中：

(1). 先引入axios和config.js  
 import axios from "axios"

import {baseURL} from "../config.js"

|  |
| --- |
| 强调: import后不要为模块对象起别名，而是直接用解构语法取出对象中的成员。 |

(2). 在定义支持promise的函数，专门向一个接口发送请求:

function getIndex(){

return new Promise(

function(resolve,reject){

axios.get(baseURL+"/index")

.then(result=>{

resolve(result.data)

})

}

)

}

(3). 导出函数: export { getIndex }

3. 在组件中引入函数，并调用函数发送请求，获得响应结果继续操作

(1). 先引入包含函数的模块

import {getIndex} from '../assets/js/apis/index.js'

(2). 在组件代码中调用函数

getIndex().then(result=>{

//将result中的数据，放到data中

})

|  |
| --- |
| 强调: 1. 一定要用箭头函数！保持回调函数中的this和vue中this保持一致，都指向当前组件对象。  2. result已经时返回的结果了，不用再result.data。 |