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->day05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->day05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 ->day05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. 懒加载:

2. 全局组件:

一. 懒加载:

1. 问题: 单页面应用首屏加载时，默认会将所有页面组件的内容都下载到客户端，首屏加载非常慢！

(1). src文件夹/router文件夹/index.js中

//凡是import进来的组件，都会被打包进App.xxx.js文件中

import Home from "../views/Home.vue"

import About from "../views/About.vue"

const routes=[

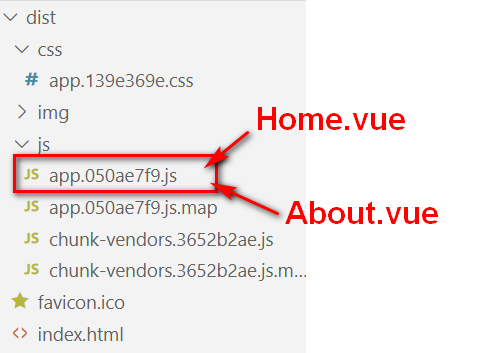
{ path:"/", component: Home},

{ path:"/about", component: About}

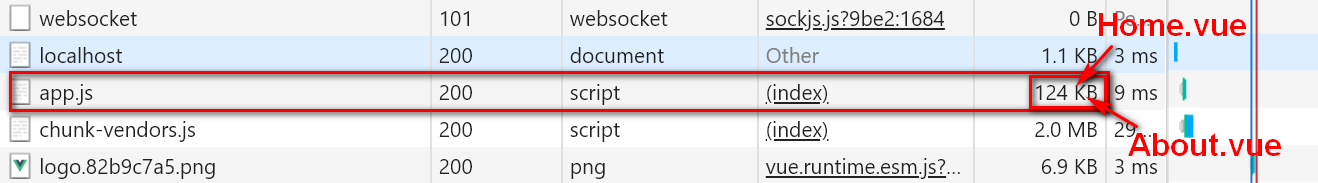
]

(2). 运行npm run build，看到dist文件夹/js文件夹下只有一个app.xxx.js文件

其中既包含了Home.vue的内容，又包含了About.vue的内容，较大，一次性下载较慢



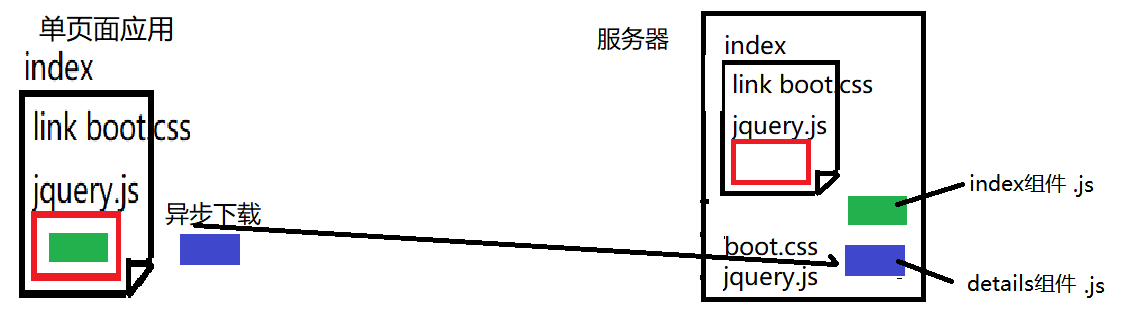
(3). 运行npm run serve，打开f12->network 刷新，看到下载列表中只有一个app.js文件，说明即使用户只看首页，也被迫把整个app.js都下载下来。较慢



2. 解决: 2种:

(1). 异步延迟加载: （默认）

a. 什么是: 将每个页面的组件内容单独打包为一个文件，首屏加载时，仅加载首屏页面一个组件的文件。当用户正在看首屏时，在低层使用异步请求，悄悄下载其他页面组件的文件.



b. 优点: 依然可减少向服务器发送请求的次数，依然可在客户端更换页面组件，提高首屏加载效率

c. 缺点: 偷跑流量

d. 如何:

1). src文件夹/router文件夹/index.js中

import Home from "../views/Home.vue"

//如果没有import进来的组件，就不会被打包进App.xxx.js文件中

//而是独立打包为一个js文件

~~import About from "../views/About.vue"~~

const routes=[

{ path:"/", component: Home},

~~{ path:"/about", component: About}~~

{

     path: '/about',

  // this generates a separate chunk (about.[hash].js) for this route

//这段代码用于专门为当前这个路由生成一个独立的about.js文件片段。

     // which is lazy-loaded when the route is visited.

     component: () => import(

       /\* webpackChunkName: "about" \*/

       // 打包时的 分段 名   为About.vue单独打包成的独立js文件起一个名字，比如这里，About.vue就会被打包成about.js

       '../views/About.vue'

     )

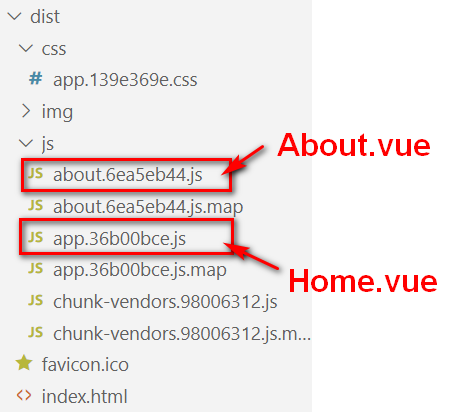
   }

]

2). 运行npm run build:

i. dist文件夹/js文件夹下:

有一个app.xxx.js文件和一个about.xxx.js文件。说明Home.vue和About.vue已经被独立打包为单独的文件。

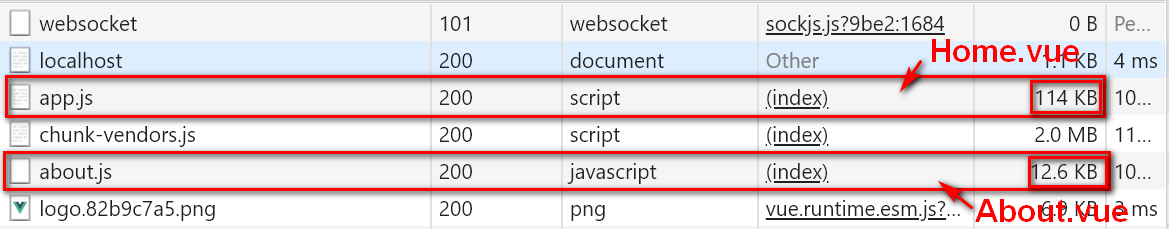


下载时就有可能先下载Home.vue先看着，然后再异步下载About.vue文件。加快首屏加载速度。

ii. dist文件夹/index.html开头: 必须下载并显示app.xxx.js，但是异步获取about，暂时不用显示。



3). 运行npm run serve，打开f12->network 刷新，看到下载列表中先有一个app.js（只包含首页内容），之后还异步下载了about.js（包含About.vue页面内容），说明用户先看到首页，但是低层异步下载了关于页面about.js备用。



(2). 彻底懒加载: (手动配置)

a. 如果用户不请求下一个页面组件，则不会提前下载其他页面组件

b. 优点: 节约流量

c. 缺点: 可能会稍微影响下一个页面组件的加载速度

d. 如何:

1). 在vue脚手架项目的根目录下新建一个vue.config.js文件，添加以下内容

module.exports={

chainWebpack:config=>{

config.plugins.delete("prefetch")

//删除index.html开头的带有prefetch属性的link，不要异步下载暂时不需要的页面组件文件

}

}

2). 路由中要保持独立打包about.js的路由写法不变！

c. 结果:

1). npm run build：

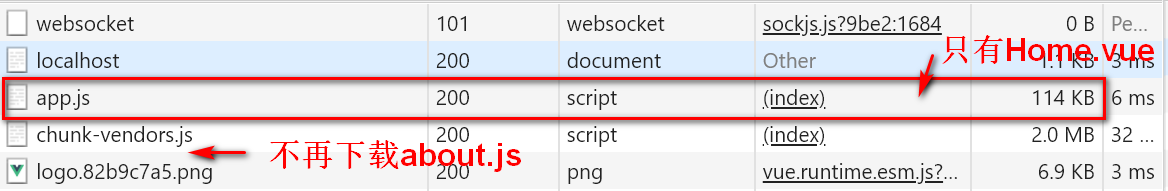
i. dist文件夹/js文件夹/还是两个文件: app.xxx.js和about.xxx.js 不变

ii. index.html中: 少了<link about.xxx.js prefetch>说明不会提前下载其他页面组件

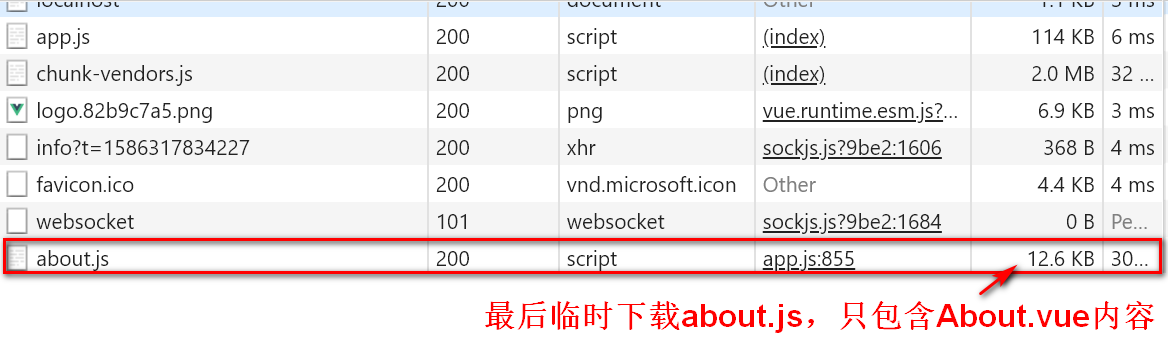


2). F12->network:

i. 当加载首屏时: 只下载app.js，不下载about.js



ii. 当点about链接，要看about页面内容时，才会临时下载about.js



补充: 脚手架项目开发完毕要交付给客户前，都要打包:

npm run build

结果: 将所有必须的组件内容都会被打包进一个js文件中，并压缩js代码。

在项目的dist文件夹下可查看打包后生成的HTML+CSS+JS代码。

二. 全局组件:

1. 在src文件夹/components文件夹下新建组件.vue文件:

2. 在组件.vue文件中编写组件的HTML内容、css内容和js内容

问题: 此时，这个组件只是一个普通的子组件，还不是全局组件

3. 让这个组件变成全局组件: 在src文件夹/main.js中new Vue()之前

(1). 引入全局组件.vue: import MyHeader from "./components/myHeader.vue"

(2). 将引入的组件对象变成全局组件: Vue.component("组件名",组件对象)

比如: 想把MyHeader变成全局组件: Vue.component("my-header", MyHeader)

4. 结果: 在当前脚手架中任何位置都可使用<my-header>标签引入全局组件

总结:

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

2. Vue绑定的原理: 访问器属性+虚拟DOM树

3. 虚拟DOM树4个优点:

(1). 小, 只保存可能发生变化的元素

(2). 遍历快

(3). 效率高，只更新受影响的元素

(4). 避免重复编码，已封装了DOM操作

4. new Vue({

el:"#app",

data:{ ... },

methods:{ ... },

computed:{ ... },

watch:{ ... }

})

5. 组件生命周期: 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()

6. 绑定与指令: 13种:

(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 v-if v-else

(5). 多个元素多选一显示 v-if v-else-if v-else

(6). 反复生成多个相同结构的元素: v-for :key

(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). 只在首次加载时绑定一次: v-once

(11). 防止内容中的{{}}被编译: v-pre

(12). 获取表单元素的值 v-model

7. 自定义指令:

Vue.directive("指令名",{

inserted(domElem){

DOM操作

}

})

8. 过滤器:

Vue.filter("过滤器名",function(oldVal,自定义形参,...){

return 加工后的新值

})

{{变量 | 过滤器名(实参值,...) | ... }}

9. 绑定样式:

(1). 绑定内联样式:

<元素 style="固定不变的内联样式" :style="变量">

data:{

变量:{

Css属性: "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,

... : ...

}

}

10. 创建一个全局组件

Vue.component("组件名",{

template:`<唯一父元素 class="组件名">

组件的HTML内容

</唯一父元素>`,

data(){

return {

变量

}

},

... ...,

components:{ 子组件 }

})

11. 父子组件传参: 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" ]

结果: 子组件中接住的自定义属性，和data中的变量用法完全一样

12. 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, 参数2:值2, ... })

.then(result=>{

result.data

})

13. SPA：

(1). 3步

a. 定义唯一完整的HTML文件

b. 为每个页面创建子组件对象

c. 创建路由器和路由字典

(2). 页面间跳转:

a. html中写死的跳转: <router-link to="/相对路径">文本</router-link>

b. js中用程序跳转: this.$router.push("/相对路径")

(3). 页面间传值:

a. 路由字典中: { path:"/相对路径/:变量", component: 组件对象, props:true }

b. 下个页面中: props:[ "变量" ]

c. 上个页面中想向下个页面传值:

<router-link to="/相对路径/变量值">文本</router-link>

this.$router.push("/相对路径/变量值")

14. 脚手架文件夹结构:

(1). 唯一完整的页面:

a. public文件夹/index.html

<div id="app"></div>

b. src文件夹/App.vue

<div id="app">

<router-view></router-view>

</div>

c. src文件夹/main.js

new Vue({

router,

render:h=>h(App)

}).$mount("#app")

(2). 每个页面的子组件都要放在src文件夹/views文件夹/

(3). 路由器对象和路由字典数组放在src文件夹/router文件夹/index.js中

(4). 全局组件和子组件应该放在src文件夹/components文件夹/

15. 每个组件都是一个.vue文件，三部分:

(1). <template> 当前组件的HTML内容

(2). <script> export default { ... } 当前组件的组件对象js内容

(3). <style scoped> 仅当前组件内可用的css样式

16. ES6模块化开发:

(1). 引入另个一文件中的内容: import 别名 from ".vue文件或.js文件的相对路径"

(2). 如果.vue文件或.js文件中包含想抛出的组件对象或js对象，则必须加export defaults修饰