1. 什么是Vue:
2. 什么是Vue：第三方开发的基于MVVM设计模式的渐进式的纯前端的js框架
3. .第三方开发：下载cn.vyejs.org
4. 基于MVVM设计模式
5. 渐进式：可以逐步在项目中引入vue的功能。不必整个项目都通篇使用vue
6. 纯前端js:无需任何后端只适合JS，vue就可以在客户端浏览器中独立使用和运行
7. 框架:已经包含核心功能的半成品代码
8. 为什么使用框架:简单--避免大量的重复代码
9. 何时：今后只要以数据操作(增删改查)为主的前端项目都可用框架开发
10. 如何使用vue:
11. 下载：2种：
12. .只下载一个独立的vue.js文件，引入网页中使用---只适用于初学者

最高版本：2.6

开发版：具有完备的注释、代码格式和见名知意的变量名---可读性好，适用于开发和学习，体积大，下载慢，不便于生产环境。带有极好的错误提示

生产版：删除一切注释，简化了变量名，体积小，下载快，使用于生产环境，删除了大量错误提示，不便于调试

1. 下载vue项目的脚手架代码---企业中使用

最高版本：4.x

1. 我的第一个vue程序3步：

(1). 定义界面: 3个要求:

a. 整个界面必须包裹在唯一的一个父元素内，习惯上唯一父元素命名为id="app"

b. 找到整个界面中可能发生变化的位置，用{{变量名}}标记出来！

本例中: 因为span的内容在点击时可能发生数量变化，所以: span的内用改为{{n}}

c. 找到触发事件的元素，在元素开始标签中绑定事件处理函数: @事件名="函数名"

(2). 定义data对象和methods对象分别保存界面所需的所有变量和所有事件处理函数

var data={ 页面所需的变量: 初始值 }

var methods={

事件处理函数名(){

//要想操作data中的变量值，必须this.变量名才行！

}

}

问题: data中的变量和methods中的方法不会自己长腿跑到页面上指定位置去

(3). 创建一个new Vue()对象，将要操作的界面和data和methods包裹在一起。

结果: new Vue()会自动将data中的变量和methods中的方法送到界面中指定的位置上。

1. MVVM
2. 旧前端代码三大部分划分
3. .HTML只负责保存页面的静态内容
4. CSS只负责保存页面所需的静态css样式
5. JS负责所有页面上的增删改查操作！
6. 问题：
7. .HTML和CSS因为缺少程序必须的要素（变量、分支、循环）
8. 所有增删改查操作，都要JS程序来负责，导致大量重复的代码
9. MVVM设计模式:对前端代码重新花费：3部分
10. .界面：增强版的
11. 等于html+css
12. HTML增加了变量、分支、循环等功能
13. .模型对象（Model）：专门保存页面所需的变量和事件处理函数对象
14. .视图模型（ViewModel）对象：专门负责自动将变量和事件处理函数送到界面指定位置---将界面和模型对象绑定在了一起
15. MVVM最大的优点：因为ViewModel可以自动维护模型对象和界面保持一致，所以减少了大量重复的代码
16. Vue框架正是基于《MVVM设计模式实现的
17. Vue绑定的原理：其实就是实现new Vue的原理
18. .先创建一个vue类型的对象
19. 引入data对象：
20. .自动为data对象中每个变量都请了保镖

结果;任何位置访问data中的变量时，其实根本就不是直接访问data中的变量，而是仅仅访问的是data中的变量请的同名保镖而已

只要外界试图访问data中的变量时，注定会触发保镖的set()方法

1. .new Vue()还自动在每个变量的set()函数内植入了一个通知函数

结果：只要外界试图修改data中的变量时，注定会触发保镖中的set()方法中的通知函数。整个vue框架就知道一个变量的值被修改了

1. 引入并打散methods对象：结果：
2. .methods中所有的函数默认归属于new Vue()对象
3. 导致methods中的函数进入new Vue()后与data中的变量的保镖（访问器属性）同级了，

结果:要访问同级中的变量，必须加this.

1. .构建虚拟DOM树;
2. 什么是：通过扫描完整版DOM树创建出的一颗新的DOM树。
3. 过程：
4. .new Vue()会查找el属性指定的界面上的元素
5. 扫描找到的父元素及其子元素
6. 碰到”@事件名=函数名”，就自动翻译为on事件名=函数名-----自动绑定事件函数
7. 碰到所有“{{变量名}}”
8. 先将可能变化的元素放入到虚拟DOM树上
9. 并且首次替换元素内容中{{变量名}}为变量的初始值
10. .结果：每次只要data中的变量被修改：
11. 自动调用变量的同名保镖set()函数
12. set()中的内置的通知函数向vue框架发出通知，说该变量发生了变化
13. 框架会自动扫描虚拟DOM树：
14. 查找受到本次变量修改影响的所有DOM元素
15. 自动利用已经提前封装好的DOM增删改查操作，只更新页面上受影响的元素。其他不受该变量影响的元素，保持不变

|  |
| --- |
| 总结：1.Vue绑定原理：访问器属性+虚拟DOM树  访问器属性+观察者模式+虚拟DOM树  2.虚拟DOM树的优点:3点：  (1).内容少！仅保存可能变化的元素。不变的元素--一个都不包含  (2).遍历快  (3).自动执行DOM增删改查操作--避免大量重复代码  (4).效率高！每次变量更改时，只修改个别受影响的元素 |

1. 绑定语法
2. 什么是:Vue中，在html内标记哪里可能发生变化的特殊语法
3. 何时：今后只要html中一个元素的内容可能随变量自动变化，就要用绑定语法
4. 如何：<any>{{变量名}}</any>
5. 结果：

(1).在参加虚拟DOM树时，该元素就会自动被添加到虚拟DOM树中

(2).一旦new Vue()中的{{}}中同名的变量方式变化，则new Vue()对象会自动将该变量的新值更新到html中{{变量名}}位置替换---不需要我们做任何增删改查操作

5.{{}}中：和ES6的模板字符串${}要求是一样的

(1)可以放;变量、算数计算、三目、有返回值的函数调用，创建对象、访问数组元素-----所有有返回值的合法的js表达式

(2).不能放：分支、循环等程序结果，以及其他没有返回值的js表达式

一.指令:(directive)为html元素添加新功能的特殊属性---13个

1.v-bind:

(1).问题：如果元素的属性值可能随变量自动变化，则不能用{{}}标记

(2).解决：

a.标准：<any v-bind:属性名=”js表达式”>

b.原理：new Vue()在扫描页面时，只要发现v-bind，就会自动计算=“”中的表达式，并将计算结果自动设置为当前属性的属性值

c.简写：<any :属性名=“js表达式”>

d.强调：如果属性明前加了:,则=后“”中的不需要再写{{}},””就充当了{{}}的作用，原来{{}}中写什么,””里就些什么

(3). 总结: 今后只要元素的属性值可能随变量自动变化时，都用:属性名="js表达式"方式来绑定，不能用{{}}。

2.v-show：

(1).需求：控制一个元素的显示和隐藏

(2).如何：<any v-show=”返回bool类型的js表达式”>

(3).原理：

a.new Vue()扫描到v-show时，就会先计算=后的js表达式，得到一个bool值。

b. 如果执行结果为true则，当前元素什么都不干，默认显示！

c. 如果执行结果为false则，当前元素自动添加display:none，隐藏

3.v-if和v-else

(1).2个元素二选一

(2).如何：

<元素1 v-if=”返回bool值的表达式”>

<元素2 v-else

(3).强调;

a.v-if和v-else两个元素必须挨着，中间不能加其他元素

b. v-else后不要加任何=，和程序中if else的else后不加表达式道理一样。

(4). 原理:

a. new Vue()扫描到v-if时，先自动计算=后的表达式的值

b. 如果v-if后表达式的值为true，则保留v-if所在的元素1，删除v-else所在元素2

c. 如果v-if后表达式的值为false，则删除v-if所在的元素1，保留v-else所在元素2

(5). v-show vs v-if 差别: （高频笔试题）

a. v-show通过display:none方式控制显示隐藏，因为不修改DOM树，所以效率略高

b. v-if通过删除元素方式控制显示隐藏，所以要修改DOM树，效率略低。

4.v-else-if；

(1).多个元素多选一显示

(2).如何：

<元素1 v-if=”返回bool值的表达式”>

<元素2 v-else-if=”返回bool值的表达式”>

... ...

<元素n v-else

(3). 强调:

a. v-if、v-else-if和v-else多个元素必须紧挨着，中间不能插入其他元素

b. v-else后不要加任何=，和程序中if else的else后不加表达式道理一样。

(4). 原理:

a. new Vue()扫描页面时，如果遇到v-if就先执行=后的表达式，获得bool值结果

b. 如果v-if表达式的值返回true，则保留v-if，删除其余v-else-if和v-else的元素

c. 如果v-if表达式的值返回false，则先后继续执行后续v-else-if的表达式。只要碰到一个v-else-if表达式的值为true，则仅保留着个v-else-if的元素，删除其余v-if v-else-if和v-else

d. 如果之前的v-if和v-else-if的表达式值都为false，才保留最后一个v-else元素，删除其余v-if和v-else-if的元素

5.v-for

(1).需求：根据数组中的内容反复生成多个相同结构的元素

(2).如何：

<要反复生成的元素 v-for=”(value,i) of 数组/对象”>

(3).强调：

a.反复生成的元素只需要写一个即可

b.v-for要写在反复生成的元素上，不能写在其父元素上

c.v-for of,既可以遍历数组，又可以遍历对象

(4). 原理:

a. new Vue()扫描到v-for，会自动遍历of后的数组或对象

b. 每遍历到数组中一个元素或对象中一个成员，都会将当前HTML元素自动创建一个新的副本。

c. 在要反复生成的元素身上或其子元素中，可以将v-for的两个变量value和i用于动态绑定内容。其中: value变量会自动获得当前正在变量的元素值或属性值。i会自动获得当前数组元素的下标或对象成员的属性名。

(5). 强调: 凡是用v-for时，必须同时绑定一个:key属性

a. 问题: 因为v-for每次反复生成的元素之间除了内容之外，单看元素是无差别的。即使将来只更改了数组中一个元素值，因为v-for无法区分反复生成的多个元素，就无法只精准更新一个元素。V-for只能采用笨办法，将v-for负责的所有元素重新生成一遍。——效率极低

b. 解决: 让v-for反复生成的每个元素都有一个唯一的标识。如果数组中某一个元素发生改变，v-for可凭借这唯一的标识找到一个页面元素，只更新一个页面元素即可。其余页面元素保持不变——效率高！

c. 如何: <要反复生成的元素 v-for="(值,下标) of 数组或对象" :key="下标"

(7). v-for还会数数！

a. <要反复生成的元素 v-for="i of 一个整数" :key="i">

b. 原理:

1). V-for从1开始循环，每次递增1，到of后给的整数值结束.

相当于for(var i=1;i<=一个整数;i++){ ... }

2). 每循环一次就将当前元素创建一个新的副本

3). 变量i会依次接住每次+1后的新值。变量i可用于元素内容中的绑定语法。

6.v-on事件绑定

(1).如何：

a.标准：<元素 v-on:事件名=“处理函数名(实参值)”>

b.简写：

1).<元素 @事件名=“处理函数名(实参值)”> 如果没有实参值，可以省略括号

c.页面所需的处理函数都应该集中定义在methods对象中

(2).Vue中如何获得时间对象：(Vue中如何获得鼠标点击的位置)

a.如果只获取事件对象，不需要传入其他值时:同DOM

<元素 @事件名="处理函数">

methods:{

//自动创建event对象

处理函数(e){

//e得到的事件对象和DOM中的事件对象完全一样！

}

}

b. 既想用事件对象，又要传入自定义实参值：（笔试题）

1). 错误做法1:

<元素 @事件名="处理函数(实参值)">

methods:{

处理函数(形参1, e){

//虽然实参可以传进来，但是e就无法获得了！

}

}

2). 错误做法2:

<元素 @事件名="处理函数(实参值)">

methods:{

//实参放在第一个位置，形参却定义在第二个位置！位置就不对！

处理函数(e, 形参1){

}

}

3). 正确: 手动使用vue提供的$event关键词代替e，手动传参

i. $event: vue自动封装的保存DOM事件对象的一个特殊关键词，可用于传参！

ii. 如何:

//当事件发生时, vue会自动封装"DOM事件对象e"

<元素 @事件名="处理函数($event, 实参值)">

methods:{

处理函数(e, 形参1){

//e得到的就是DOM中的事件对象，用法也和DOM中一样

}

}

强调: 处理函数中$event与实参值的位置没有固定顺序，但是必须和methods中处理函数定义中的形参列表顺序保持一致！

1. v-html:
2. .问题：如果要绑定的是一段html代码，则使用{{}}绑定，会保持html代码的原样，而不经过编译，直接显示到页面上
3. 解决：今后，只要绑定的变量是一段HTML代码，则必须使用v-html指令替代{{}}来绑定
4. 如何<元素 v-html=”变量或js表达式”></元素>
5. 优点：v-html会先将不看了中的HTML片段交给浏览器解析成可以给人展示的内容，
6. v-cloak和v-text:防止用户短暂看到{{}};
7. .问题：当用户网速慢时，如果js代码还未下载执行，则用户有可能短暂看到页面上的{{}}----尴尬
8. 解决：2个办法
9. v-cloak 指令：
10. .原理：在new Vue()下载执行前，先将所有带有{{}}的元素短暂隐藏。等到new Vue()下载执行完时，在自动将所有{{}}的元素显示出来
11. 2步：①现有网页中的style中定义一个属性选择器：[v-cloak]{display:none}.意为所有带有v-cloak属性的元素，暂时隐藏

②再有{{}}的元素上都加上v-cloak属性，不用加=

1. .结果：所有带有v-cloak的元素，都暂时隐藏，直到new Vue()下载完场并执行完成时，自动找到v-cloak的元素，删除v-cloak属性。于是元素便又会显示出来
2. v-text指令：
3. .原理：既然不想看到{{}}，就可以用一种特殊的指令代替{{}}，功能还一样，因为指令都是写在元素的属性中。即使没有加载完，用户也不可能看到元素的属性
4. 如何：1步：

i. <元素 v-text="变量或js表达式"> </元素>

ii. 原理: 当v-text的内容计算好之后，会自动将执行后的结果代替元素的内容

3). 问题: 如果元素的内容就是由固定的内容和变化的变量拼接起来的，则被迫使用模板字符串！<元素 v-text="`模板字符串`"></元素>

4). 简写: 如果v-text后跟的是一个模板字符串，则可以省略""，只保留模板字符串的反引号``即可！<元素 v-text=`模板字符串`></元素>

1. v-once:
2. .问题：有些绑定只在首次加载时，修改一次。之后其他地方的修改，都不会影响这里的显示结果。如果一个元素之后不需要在更新，但是却站着虚拟DOM树中的一个位置，会影响虚拟DOM树遍便利速度
3. 优化：如果只在首次加载时显示，之后其他地修改，这里都不会受影响的绑定，可以用v-once绑定

(3). 如何: <元素 v-once>{{变量或js表达式}}></元素>

(4). 原理: 只在首次渲染页面时，替换元素的绑定语法内容。但是不会将当前元素加入虚拟DOM树。结果: 只在首次加载一次，之后都不会再改变。

10. v-pre:

(1). 极端的问题: 如果元素的内容正文中刚好也包含{{}}，但是不想被vue编译，而是原样显示！

(2). 如何:<元素 v-pre> ... {{xxx}}...</元素>

一. 双向绑定:

1. 问题: 使用之前的12种指令，的确可以将new Vue()中的变量值更新到页面显示。但是都无法将页面中用户输入的新内容，自动反向更新会new Vue()中保存。

2. 单向绑定: 只能将new Vue()中模型对象中变量值的修改更新到页面上显示。如果页面中元素内容发生变化，无法自动将更改更新回new Vue()中的模型对象中保存

之前的12种指令，都只是单项绑定而已。

3. 解决: 今后只要希望自动获得用户在界面上输入或选择的新值，都只能用双向绑定

4. 双向绑定: 既能将new Vue()中模型对象中变量值的修改更新到页面上显示。又能将页面中元素内容发生的变化，自动更新回new Vue()中的模型对象中保存

5. 如何: <表单元素 v-model:可能变化的属性="变量">

6. 强调:双向绑定几乎只用在表单元素上

7. 原理: v-model包含两部分原理

(1). 从Model->View的绑定与之前讲解的绑定元素一样: 访问器属性+虚拟DOM树

(2). 但是从View->Model的绑定是新的原理：new Vue()扫描到包含v-model的表单元素时，就自动为表单元素绑定onchange=function(){}事件处理函数。意为当表单元素的值发生改变时，就自动触发。然后，在onchange事件处理函数内，包含自动获取当前元素值并更新回Vue中指定变量中保存的语句。

9. 问题: 不是所有表单元素修改时，更新的都是value属性。比如radio和checkbox元素更新时，更新的就不是value属性。而是value属性写死固定不变，而更新checked选中状态属性。

10. 解决: 用户修改表单元素时，那个属性变化，就用v-model绑定那个属性。比如，value变化，就用v-model:value=”变量”，checked变化，就用v-model=checked=”变量”

11. 其他类型的表单元素如何双向绑定:

(1). 单选按钮: <input type="radio">

a. 当用户选中与不选中时，修改的是checked属性，所以如果双向绑定时应该用v-model:checked="变量"。

b. 如何: <input type="radio" value="写死的值" v-model:checked="变量">

c. 原理:

1). 从Model->View绑定时，v-model先自动用变量的值和写死的value值做比较。如果变量值==value值，则当前radio选中。否则如果变量值!=value值，则当前radio不选中。

2). 从View->Model绑定时，当用户切换选中状态时，自动触发onchange事件，onchange事件中只将选中的radio的值自动更新回v-model绑定的变量中。

(2). 下拉选择框: <select>...</select>

a. 当用户在select内选择不同的option时，其实改的是整个select的value属性。

b. 如何: <select v-model:value="变量">

c. 原理:

1). 从Model->View绑定时，v-model将变量值自动和每个option的value比较。哪个option的value值与变量值相等，就自动选中这个option

2). 从View->Model绑定时，当用户选中一个option时，option的value先交给select，成为整个select的value，然后v-model再将select的新value，自动更新回data中的src变量中保存。

12. 简写: 其实以上例子中v-model后的:value和:checked全都可以省略！

今后双向绑定只写<表单元素 v-model="变量">

因为v-model会自定区分该绑定哪个元素！不用人为区分！

二．绑定样式：

1.绑定内联样式

(1).只修改个别css属性值时，首选修改内联样式：

(2).比如：做游戏时，控制主角的方向：top，left等

控制一个元素的显示隐藏：display:none;

1. .如何：
2. 不好的方法：因为style属性值也会一个字符串，所以完全可以当做一个普通的字符串属性来绑定：
3. .HTML中：<元素 :style=”变量”>
4. new Vue({

data:{

变量=“属性名1：属性值1，属性名2：属性值2，...:...”

}

})

1. .问题：不便于修改style中某一个css属性值 ，因为多个css属性混在一个字符串中，无法对其属性值进行加减
2. 好的做法:将整个style属性，看做一个对象来绑定
3. .HTML中：<元素 :style=”变量”>

2).new Vue({

data:{

变量:{

属性名1：属性值1

属性名2：属性值2

... : ...

}

}

})

3). 优点: 非常便于只修改其中某一个css属性！

vm.变量.css属性值="xxx" //new Vue()之外修改时

或

this.变量.css属性值="xxx" //new Vue()内部修改时

2.绑定class

(1).何时：批量修改多个css属性时，首选修改class属性

(2).比如：带样式的表单验证，提示信息在验证通过和验证失败之间来回切换

因为每种样式都包含多个css属性，所以用class批量修改是最划算的

1. .如何：
2. 不好的做法
3. .将class属性也当做普通的字符串来绑定
4. 如何：
5. HTML中：<元素 :class=”变量”>
6. new Vue({

data:{

变量=”class1:true,calss2:false,... : ...”

}

})

3).结果.不好修改其中的某一个class.

1. 好的做法：
2. .将class也看做一个对象来绑定
3. 如何；
4. HTML中：<元素 :class=”变量”>
5. new Vue({

data:{

变量：{ //自动翻译(只有值为true的class，才会被翻译进最终的class字符串中)

class1:true或false,

class2:true或false

}

}

})

1. 如果一个元素上既有固定不变的class，又有个别变化的class.应该：
2. .HTML:<元素 class=”固定不变的” :class=”变量”>
3. new Vue({

data:{

变量：{

可能变化的class:true或false,

... : ...

}

}

})

1. .结果：变量中值为true的class会现编译为一个字符串，然后和固定不变的class组合起来，组成新的class共同作用于元素上
2. 自定义指令
3. 如果vue中12中指令不够用，就可以自己添加vue指令
4. 何时：如果希望元素在首次加载时，对元素自动执行某些操作时，就可以用自定义指令

比如：让某个元素在页面加载时就自动获得焦点

1. 如何：

(1).Vue.directive(“指令名”,{ //定义时，不要加v-前缀

//将来当带有这个指令的DOM元素被渲染到页面上时，就会自动执行inserted回调函数

inserted(dom元素){

//形参dom元素会自动获得当前拥有该指令的DOM元素对象

//对当前拥有该指令的DOM元素执行原生的DOM操作

}

})

(2).使用时，在元素上使用自定义指令： 要加v-前缀

<元素 v-指令名>

1. 原理：
2. .new Vue()扫描界面时，发现有v-前缀的指令时，就会在原生的指令和自定义的指令中找该指令名，如果找到了，就执行inserted里面的回调函数
3. inserted()里面包含了所有的原生的DOM操作
4. 计算属性
5. 什么是：自己不保留属性值，需要根据其他属性值经过复杂的计算得到自己的属性值
6. 为什么：因为通常需要复杂计算出的结果，都不会保留在数据库里，自然后端也不会给我们提供现成的值直接使用。只能靠我们前端，利用其他属性值计算获得

比如：购物车的总价，计算过程复杂，又频繁变化，通常数据库里不保存，而是靠客户端动态计算获得

3. 何时: 只要界面上想要的值，后端没有直接提供，但是可以根据其他值计算获得，就用计算属性。

4. 如何: 2步

(1). 定义一个计算属性:

new Vue({

el:"#app",

data:{

可直接使用的属性

},

methods:{

事件处理函数

},

computed:{

计算属性名(){ //计算属性虽然用法和属性一样，但是本质却是一个函数！

//根据data中其他属性值，经过复杂的计算获得一个最终的属性值

return 计算结果

}

}

})

(2). 在HTML中使用计算属性:

比如: <元素>{{计算属性名}}</元素>

强调: 计算属性虽然本质是函数，但是使用时不要加()，和data中的普通属性完全一样

5. 结果: 运行时new Vue()会自动计算计算属性的结果，代替{{计算属性名}}的位置

6. computed vs methods：

(1). 用法:

a. methods中的函数在HTML中使用时必须加()调用

b. computed中的计算属性在HTML中使用时不用加()

(2). 效率: 当反复使用这个函数或计算属性时

a. methods中的函数，每加一次()，就重复执行一次！——没必要

b. computed中的计算属性结果，只在首次使用时计算一次，然后就被vue缓存起来。从第二次使用开始无需重复计算，就可直接使用缓存中的中。——效率更高！

7. 总结: 何时使用函数？何时使用计算属性？

(1). 如果更侧重于使用函数的执行结果数值时，首选计算属性

(2). 如果不关心执行结果，反而更侧重于执行一项操作（比如: 发送ajax请求），首选methods中的函数。

1. 过滤器
2. 什么是：专门对变量的原始值进行加工后在显示出来的一种特殊函数
3. 为什么：有些时候，变量的初始值人看不懂，需要先加工成可以看的，在显示出来，比如：性别的1和0翻译为男和女，时间的毫秒数，翻译为年月日，时:分:秒
4. 何时：今后只要显示的变量值不是直接可以看得，都需要加工后，就用过滤器
5. 如何：2步
6. .定义过滤器

Vue.filter(“过滤器名”,function(oldVal){

return 过滤后的新值

})

1. .在页面上使用过滤器

比如：在绑定语法的{{}}中使用： <元素>{{变量 | 过滤器名}}</元素> |：充当连接的作用

1. 结果：
2. .new Vue()扫描到绑定语法中的 | 时，就回去vue家找指定名称的过滤器函数，
3. 如果找到，就自动调用过滤器函数:

a. 将|左边的变量的原始值自动传给过滤器函数的oldVal形参

b. 将过滤器执行的结果返回给new Vue()

(3). new Vue()会用过滤器返回的加工后的结果，代替当前{{}}语法位置，显示给用户看。

6. 过滤器可以传参:

(1). 定义过滤器: 预留

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

return 根据不同的自定义形参，过滤出不同的新值

})

(2). 在页面上使用过滤器:

比如: 在绑定语法的{{}}中使用:

<元素>{{变量 | 过滤器名(自定义实参值)}}</元素>

连接

7. 多个过滤器可以连用：

<元素>{{变量 | 过滤器1 | 过滤器2 | ...}}</元素>

强调: 过滤器连用时，每个过滤器既可能接住变量的原始值，又可能接住的是上一个过滤器加工后的半成品。

1. Axios
2. 每种框架和函数库中都提供了专门的发送ajax请求的函数。永安个框架，就要用对应的哪种ajax函数
3. 什么是：axios时基于promise技术的专门发送ajax请求的函数库----几乎专用于vue中，但是其他框架也可使用
4. 如何：
5. .引入axios.js,获得一个axios对象
6. 因为大部分服务器端接口的域名和端口号部分基本上是一致的，只有服务器端接口地址的相对路径部分不一致，

(2).发送get请求：

axios.get(“服务器端口接口地址”，{

params:{请求参数1：值1，请求参数2：值2，... : ...}

}).then(result=>{ //function(result){ //有延迟！！！

//成功接收响应结果后，自动执行.then中的箭头函数

//result并不直接返回结果

//result.data 才是返回结果

//且返回结果自动转为js数字或对象了

})

结果：服务器接口地址?请求参数1=值1&请求参数2=值2&...

(3).发送post请求：

axios.post(“服务器端接口地址”,”请求参数1=值1&请求参数2=值2&...”)

.then(result=>{ //有延迟

Result.data //才是服务器端响应的结果

})

二. 生命周期:

1. 问题: dom中和jquery中，都有页面加载完成事件可以在页面加载完成后自动执行一些页面初始化操作（发送ajax请求以及事件绑定）。但是dom中的window.onload和jquery中的$(function(){ ... })，都不属于vue。今后其实有一个规定: 所有vue的js代码，都要写在new Vue()对象内部。new Vue()对象之外不应该有任何代码！

2. 何时使用生命周期: 在Vue中也想在页面加载后自动执行一些初始化操作时，都要用生命周期来做。

3. 什么是生命周期: 指一个Vue对象从创建到渲染出来让人看到所经历的整个过程:

好像: 出生->幼儿园->小学->中学->大学->......

4. 一个Vue对象从创建到渲染出来让人看到，共可能经历4个阶段

(1). 创建(create): ——必经阶段

a. 创建new Vue对象，引入data对象，并请保镖

b. 暂时没有虚拟DOM树

(2). 挂载(mount): ——必经阶段

a. 生成虚拟DOM树

b. 首次渲染页面内容

(3). 更新(update): ——只有data中的变量被更改时，才触发更新阶段

(4). 销毁(destroy): ——只有调用$destroy()函数，销毁当前组件时，才触发销毁阶段

5. 如果希望在某个生命周期阶段自动执行一件事情，还必须使用生命周期钩子函数

(1). 什么是生命周期钩子函数: 在vue对象创建到每个阶段时，自动调用的一种回调函数。

(2). Vue对象加载过程中: 共经历8个生命周期钩子函数（每个生命周期阶段前后各有一个钩子函数）

beforeCreate(){ ... }

a. 创建阶段(create)

created(){ ... }

beforeMount(){ ... }

b. 挂载阶段(mount)

mounted(){ ... }

beforeUpdate(){ ... }

c. 更新阶段(update)

updated(){ ... }

beforeDestroy(){ ... }

d. 销毁阶段(destroy)

destroyed(){ ... }

6. 发送ajax请求的代码应该放在？

(1). 肯定不能放在update阶段！因为update阶段不是必经阶段，不一定总是执行。必须修改data中的变量才触发。如果不修改data中的变量是不会触发的。所以放在update中的代码，在页面首次加载时，无法自动执行！

(2).可以放在created()中，但是不保险:

a. created()执行时，data对象已经有了，axios请求回来的结果，就可以放入data对象中

b. 不保险: created()执行时，暂时还没有虚拟DOM树，可能无法完成一些DOM相关的操作。

(3). 最保险的做法: mounted()中！

a. 此时既有了data对象，可保存axios请求回来的数据，又有了虚拟DOM树，也可以执行一些DOM相关的操作。

三．组件

1.什么是：拥有专属的HTML+CSS+JS+数据的可重用的页面独立功能区域

2.为什么：重用

3.何时：今后只要页面中出现可能反复使用的功能区域，都要封装为组件，在反复使用组件

4.如何:2步

(1).定义组件

Vue.component(“组件名”,{

//和new Vue()中几乎完全一样！每个组件其实就是和new Vue()一个功能完全相同的小的分身而已。new Vue()负责整个页面，二组件负责自己的小范围而已

//没有了el,取而代之的时

template:”一段html代码片段”,

methods:{},

computed:{},

... : ...

})

三. 组件:

1. 什么是组件: 拥有专属的HTML+CSS+JS+数据的可重用的页面独立功能区域

2. 为什么: 重用！

3. 何时: 只要页面中出现可能反复使用的功能区域，都要封装为组件，再反复使用组件

4. 如何: 2步

(1). 定义组件:

Vue.componet("组件名",{ //强调:组件名不要用驼峰命名，多个英文单词要用-分割

//和new Vue()中几乎完全一样！每个组件其实就是new Vue()一个功能完全相同的小的分身而已。new Vue()负责监控整个页面，而组件负责监控自己负责的小范围区域而已。

//没有el了，取而代之的是一个

//组件的template中只能有一个唯一的父元素包裹

template: `HTML代码片段`, //因为这里是为将来反复使用这个组件定义的统一样式模板。将来使用这个组件时，反复创建的副本，和这里的HTML片段完全一样！

//data不再是一个对象，而变成一个可反复调用的函数

data(){ //将来每使用一次这个组件，就自动调用一次data()函数

return { //每次调用data()函数，都返回一个新创建的模型对象给当前这个组件的一个副本专属。——保证组件虽然可反复使用，但是数据是彼此隔离，互不影响的！

组件模板中所需的变量

}

},

//以下内容和new Vue()就完全一样了

methods:{ ... },

computed:{ ... },

//生命周期钩子函数

... ...

})

(2). 如何使用自定义的组件: 组件其实就是一个可反复使用的自定义HTML元素而已。组件名其实就是元素的标签名

<组件名></组件名>

5. 原理:

(1). new Vue()扫描页面时，碰到不认识的元素标签，就会去Vue家找有没有同名的自定义组件

(2). 如果找到同名的自定义组件，则先用组件对象的template HTML片段代替页面上组件元素的位置

(3). 自动调用一次data()函数，为本次组件的副本创建一个全新的专属的模型对象副本。——保证这个组件副本中操作模型变量，不影响其他组件副本中的数据

(4). 最后为这次组件副本创建一个小快递员对象，保持组件范围内模型对象与界面之间的同步。

四. 组件化开发:

1. 什么是组件化开发: 今后一个html页面其实都是由多个组件拼接起来的。今后拿到一个页面，第一件事应该是划分组件区域。然后再由多人分头编写一个页面的不同区域。最后再将每个人编写的组件拼接到一起形成一个HTML页面给人看！

2. 为什么:

(1). 便于大项目的分工协作。并行开发。效率高！

(2). 松耦合！组件与组件之间相对独立，互不影响。一个出错，不影响大家！

3. 何时: 今后几乎所有项目都采用组件化开发！

4. 如何:

(1). 拿到页面图片，划分组件结构，2个依据

a. 按功能

b. 按是否重用

(2). 划分了几个组件，就要创建几个独立的js文件，在独立的js文件中定义组件对象

(3). 将所有组件引入到.html页面中，由new Vue()经过反复扫描和替换过程，将所有组件内容最终拼接到一起给用户显示。

5. 子组件:

(1). 问题: Vue.component()创建的组件，称为全局组件。可在页面中任何位置随意使用。但是有些组件明显不能超出父组件的范围使用。一旦超出要么出错，要么没意义！

(2). 解决: 今后如果限制一个子组件只能在其父组件范围内使用，就应该定义子组件，而不应该使用全局组件

(3). 如何: 2步:

a. 定义子组件对象:

1). 子组件对象的内容应该和定义全局组件内容的要求完全一致！

2). 但是不能用Vue.component()创建，应该仅仅保存在一个普通的js变量里

3). 变量名应该是子组件名称的驼峰命名方式！

b. 必须在父组件对象中的components属性中，将子组件对象包含进来。

父组件对象: {

template:`...`,

data(){

return { ... }

},

... ...,

components:{ 驼峰命名的组件变量名, ... ... }

}

(4). 结果: 凡是出现在这个父组件components中的子组件对象，就只能在当前父组件内使用。出了当前父组件使用，会报错！

(5). 问题: 子组件定义时的变量名是驼峰命名，在父组件模板中使用子组件标签时，应该用驼峰？还是用-

(6). 答: 用-分割多个单词！因为components会自动将驼峰命名的组件变量名，翻译为-分割的名字。

(7). 强调：将多个子组件引入到页面时，子组件必须在父组件之前引入！

7. 组件间数据传递:

(1). 问题: vue组件化开发中，子组件无权直接使用父组件的变量！

(2). 解决: 采用专门的方法，让父组件将自己的一个变量传递给子组件才行

(3). 如何: (父给子) 2步:

a. 父亲放数据: 父组件在template中  
 <子组件 :自定义属性="父要给子的变量">

父亲把自己的一个变量值放进孩子身上一个指定名称的兜里

b. 孩子收数据: 子组件对象中:

子组件对象: {

props:[ "自定义属性" ], //孩子从父亲放东西的同名兜里取出父亲给的数据继续使用

template:`...子组件的模板中使用自定义属性名等效于使用父组件的变量值...`

}

强调:

1). 通常为了保证一个东西在程序中只有一个统一的名称，便于维护，避免歧义，都会将父组件变量名和自定义属性起相同的名字。但不是必须。

2). props中的属性名 vs data中的变量

i. 相同点: 都既可用于HTML中的绑定和指令，又可用于js中通过this.xxx访问

ii. 不同点: data中变量的值是组件内定义的; props中的属性值是外部给的！

五. SPA: Single Page Application

1. 什么是单页面应用: 整个应用程序只有一个完整的.html文件。其他所谓的"页面"，其实仅仅是组件而已

2. 为什么: 普通多页面应用 vs 单页面应用

|  |  |  |
| --- | --- | --- |
|  | 普通多页面应用 | 单页面应用 |
| 请求次数 | 每切换一次页面，都要重新请求服务器，重新下载页面——请求次数多，慢 | 在首次加载时就将唯一完整的HTML页面和其他"页面"组件片段，都下载到客户端。今后切换"页面"时，其实.html文件不动，只更新局部的组件内容即可。且不用向服务器重新发送请求。——请求次数少，快 |
| 页面加载效率 | 每切换一次页面，都要将旧的DOM树删除。下载新的页面后，重新建立完整DOM树——慢 | 因为每次切换页面，只更新局部内容，所以，不需要删除整棵DOM树，只更新局部的组件内容即可——快 |
| 多页面共享资源(boot,jq) | 每次切换页面都要把整个页面重新扫描一遍。扫描到link和script，会重新发送请求。——增大请求次数，浪费网络浏览量 | 完整的html文件只在首次加载一次，所以link和script，也只在首次下载一次。之后切换页面，link和script部分是不变的。所以，无需重复请求——减少请求次数，节约网络流量 |
| 页面切换动画 | 因为每次切换页面都是删除旧页面，下载新页面，导致前后两个页面不可能同时存在于客户端。——几乎无法完成页面切换动画。 | 因为单页面应用，所有所谓的页面都已经在客户端同时存在了。所以，完全可以实现页面切换动画 |

但是，传统大页面应用的缺点: 首屏加载极慢——因为首次加载时，就要把所有页面的组件都下载下来，东西多——慢

但是，已经被解决了：——懒加载

3. 今后几乎所有的框架都采用单页面应用的方式开发。

4. 如何:

(1). 先创建一个唯一完整的HTML文件要求:

a. 具有标准的vue网页结构: 引入vue.js，<div id="app">, new Vue({el:"#app"})

b. 必须引入vue-router.js

c. 在<div id="app">内必须用<router-view></router-view>为将来要加载的页面组件，占位

(2). 应用程序中要求有几个"页面"，就要创建几个子组件.js文件

var 对象名={

//内容同子组件完全一样！只不过当做页面用而已

}

强调: 页面组件.js创建完之后，也要先引入唯一完成的HTML中

(3). 创建一个VueRouter对象，在对象中定义一个路由字典列表

var router=new VueRouter({

routers:[ 相对路径 路径对应的页面组件对象

↓ ↓

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

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

... ...

]

})

强调: 创建完router对象后，一定要引入唯一完整的html页面中，还要将router对象加入new Vue()中。否则new Vue()不认识<router-view>标签！

new Vue({

el:"#app",

...,

router

})

5. 页面跳转:

(1). HTML中写死的:

a. 不好的做法: <a href="#/相对路径">

b. 好的做法: 用<router-link>代替a

<router-link to="/相对路径">xxx</router-link>

(2). Js中用程序跳转: 固定套路

this.$router.push("/相对路径")

6. 页面间传参:

(1). 修改路由字典:

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

Props:true 可将将来通过地址栏传的参数值，自动传给props中的同名变量。

(2). 上一个页面跳转时: 相对路径之后，只拼参数值即可

/相对路径/参数值

(3). 结果: 在接受参数值的页面中:

props:[ "变量名" ]

组件内部，可将变量名用于绑定语法。

强调: 一旦一个路径设置了/:变量名，则进入这个路径必须带参数值。否则，无法进入这个路径。导到404页面。

一. 脚手架:

1. 问题: 当项目中组件非常多时，都会用专门的文件夹来分类组织不同的组件文件。便于查找和维护。但是，人和人创建文件夹的习惯和命名差异很大！结果，一个人建立的项目结构和另一个人建立的项目结构，很可能完全不一样！给开发人员接手和维护新项目带来了巨大的障碍。

2. 解决: 今后前端的所有框架以及后端的nodejs express项目，都已经有了标准化的脚手架项目结构。今后公司中开发，都是基于脚手架的标准项目结构进行开发的。

3. 好处: 一个项目的结构和另一个项目的结构几乎完全相同。开发人员接手和维护新项目时，几乎是无缝衔接！

4. 什么是脚手架: 已经拥有半成品核心功能和标准化文件夹结构的项目源代码。

5. 为什么: 标准化和避免重复编码

6. 何时: 今后所有框架都是先创建脚手架，再在脚手架标准项目结构基础上添加个性化功能定制。

7. 如何:

(1). 创建脚手架项目: 2步

a. 先安装可反复创建脚手架项目的工具: 会下蛋的老母鸡——只安装一次就可反复使用

npm i -g @vue/cli

vue -V

看到版本号，说明安装成功

b. 再反复使用工具，反复创建不同项目的脚手架源代码: 老母鸡下的蛋——每做一个新项目都要重新创建一套标准的脚手架项目源代码

1). vue create 项目名 回车

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

选择 预设 箭头 键

default (babel, eslint)

默认

> Manually select features

手动 功能

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

(\*) Babel 将ES6,ES7代码翻译为所有浏览器都认识的传统的ES5以及以下代码

( ) TypeScript

( ) Progressive Web App (PWA) Support

(\*) Router VueRouter，是实现单页面应用的核心组件

( ) Vuex 第三阶段暂时不选。但是今后所有项目都要选

( ) CSS Pre-processors 如果项目中使用了sass或less，需要对css进行预编译，才选

() Linter / Formatter 学习阶段取消选择！linter是代码质量检查工具。

( ) Unit Testing

( ) E2E Testing

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

默认VueRouter采用"#/相对路径"，作为路由地址。但是有人讨厌#，想去掉，就可以开启history模式。一旦开启history，VueRouter采用"/相对路径"作为路由地址。

不要开启history模式，因为需要服务器端首页重定向功能的支持——专业的服务器端技术。

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

期望 放 配置

In dedicated config files 放在多个不同的配置文件中

> package.json 集中放在一个package.json文件中

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

保存这些为 预设 为 今后的项目

因为每个项目的配置可能各不相同，所以保存一套预设，意义不大！

7). 看到successfully说明在硬盘上创建脚手架项目源代码成功！

🎉 Successfully created project xzvue.

👉 Get started with the following commands:

$ cd xzvue

$ npm run serve

(2). 首次运行脚手架项目: 用vscode运行

a. 用vscode打开新生成的项目文件夹xzvue

b. 右键选择package.json文件，选"在终端中打开"，看到打开了一个新终端窗口

c. 当终端窗口中出现: xxx\xzvue>，就输入: npm run serve

1). 自动启动了一个类似于live server和apache的临时服务器，用来保存项目的所有文件，供人们通过浏览器输入地址访问项目中的网页。

2). 将项目源代码编译为所有浏览器都认识的HTML+CSS+JS代码。

d. 看到: DONE Compiled successfully in 2982ms 编译成功

e. 按住Ctrl，点下方链接: Local: http://localhost:8080/，就打开浏览器查看默认的页面了

f. 在打开的浏览器中看到脚手架项目自带的首页

g. 其实脚手架已经包含了单页面应用的核心功能:

1). 包含两个页面: Home和About

2). 公共页头组件

3). 在页头点Home和About，可在两个页面间切换，说明已经创建了VueRouter路由！

8. 改造脚手架项目:

(1). 如果自己开发，不用git，可删除.git文件夹

(2). 脚手架默认不带axios，所以需要安装axios

a. 右键点击package.json，选择"在终端中打开"

b. 看到文字: xxx\xzvue> 输入 npm i -save axios

c. 看到+ axios@0.19.2，说明安装成功。打开node\_modules文件夹，可看到axios文件夹

(3). 试着修改脚手架默认首页的内容:

a. 打开src/views/Home.vue

b. <template>内是脚手架首页的HTML内容

1). 删除<HelloWorld msg="Welcome to Your Vue.js App"/>

2). 换成自己的内容: <h1>我的第一个脚手架项目</h1>

3). 删除<template>下方<script>元素及其内容。

4). 删除src/components/HelloWorld.vue组件文件

c. 如果刚才没有关闭npm run serve，则每修改一次脚手架中源代码，只要Ctrl+S保存修改后，都会自动重新编译，不用反复运行npm run serve。就可以在浏览器中查看到新的修改后的效果。除非不小心把npm run serve关了。才需要重新运行npm run serve。

9. 脚手架项目文件夹结构:

(1). 回顾: SPA应用程序的结构: 4部分:

a. 唯一完整的HTML页面

1). 引用:

i. 引入公共的bootstrap.css、jquery.js和bootstrap.js

ii. 引入Vue单页面应用必须的js文件: vue.js 和 vue-router.js

iii. 引入所有页面组件、全局组件

iv. 引入路由器对象.js

2). 唯一父元素:

<div id="app">

<router-view>停车位</router-view>

</div>

3). new Vue({

el:"#app",

router

})

b. 定义页面组件，习惯上所有页面组件.js放在一个专门的文件夹中, 习惯上名为views

c. 定义VueRouter对象，并定义路由字典，通常放在router.js中

var router=new VueRouter({

routes:[

{path:"/", component:组件名},

{path:"/details/:lid", component:组件名, props:true},

{path:"\*", component:组件名}

]

})

d. 多个页面共用的页头等组件通常定义为全局组件，习惯上全局组件或个别子组件.js放在一个专门的文件夹中，习惯上名为components

(2). 脚手架文件夹结构: 大致和SPA应用结构几乎完全一样。只不过个别地方划分的更细致而已

a. 唯一完整的HTML页面: 被拆为三部分

1). public/index.html ——几乎不需要我们修改

2). 公共的资源文件bootstrap.css应该放在public/css文件夹，并引入到index.html中

3). 公共的js文件jquery.js和bootstrap.js应该放在public/js文件夹，并引入到index.html中

4). 项目中用到的所有图片，应该放在public/imgs文件夹中

5). 不要在唯一完整的HTML页面中引入vue和vue-router.js了。因为脚手架采用模块化开发，稍后告诉大家vue和vue-router.js在哪里引入。

6). 页面组件和router对象.js不要在唯一完整的HTML页面中引入了。因为脚手架采用模块化开发，页面组件和router对象有专门的路径来保存

7). new Vue()也不要放在唯一完整的HTML页面中了

到此，今后index.html页面再不需要做任何更改！

8). 问题: <router-view>去哪儿了？

src/App.vue文件中保存了:

<div id="app">  
 <router-view></router-view>

</div>

运行时会用App.vue中的HTML内容代替index.html中的空<div id="app"></div>

9). 问题: new Vue()去哪儿了？

src/main.js

import Vue from 'vue' //以模块化方式引入vue.js

import App from './App.vue' //引入App.vue文件中的内容

import router from './router' //以模块化方式引入了new VueRouter()对象和路由字典

Vue.config.productionTip = false

new Vue({ //创建监控整个页面的大的Vue对象（快递员）

router,

render: h => h(App) //加载App中的HTML内容

}).$mount('#app')

挂载到index.html中代替<div id="app">位置

10). npm run serve 先执行main.js，引入App.vue中的内容，并用App.vue中的内容代替了index.html中空的<div id="app"></div>

b. 页面组件:

1). 脚手架规定所有页面组件都应该放在src/views文件夹下

2). 每个页面组件应该定义为一个xxx.vue文件，不是.js文件

a. .vue中可同时包含一个组件的HTML+JS+CSS

b. .js中只能包含js和简单的HTML，无法包含组件的css内容。

3). 每个.vue文件都可以包含三部分内容:

a. <template> <!--专门保存当前页面的HTML内容-->

<唯一父元素>

当前页面的HTML内容

</唯一父元素>

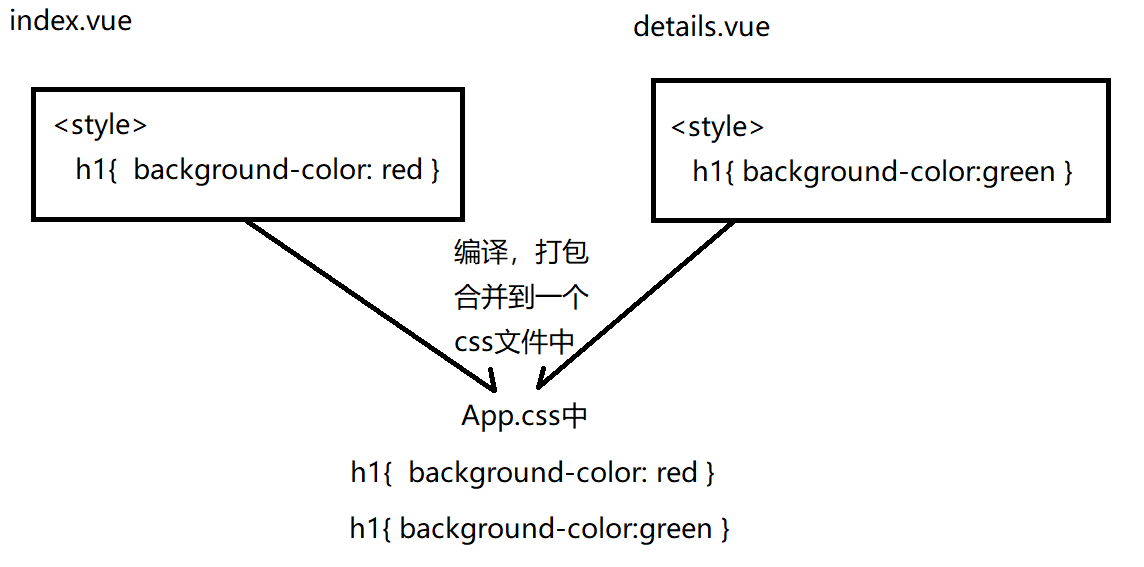
</template>

b. <style scoped>

只写当前页面用到的css样式

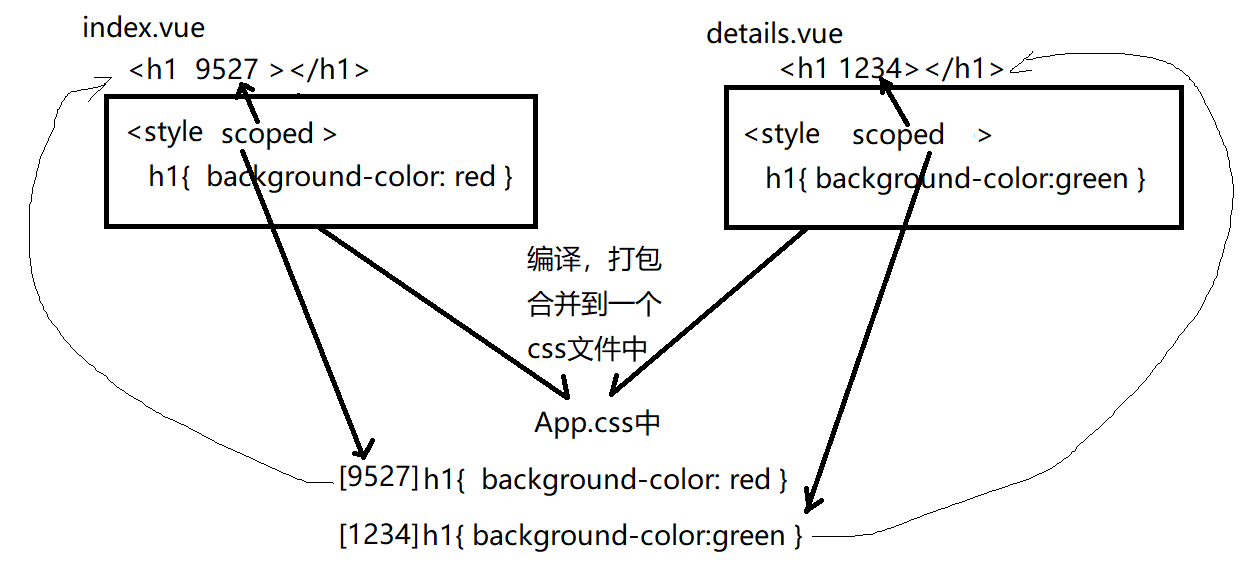
</style>

问题: 因为将来所有.vue文件中的<style>都会被编译到一个.css文件中。极有可能造成不同.vue中的选择器和样式发生冲突！



解决: 今后所有<style>都要写为: <style scoped>

scoped会自动保证不同组件的样式只局限于当前组件，不影响其他组件。



c. <script>

当前页面组件的组件对象js内容

</script>

c. 创建路由器对象和路由字典: src/router/index.js

import Vue from 'vue' //引入vue.js

import VueRouter from 'vue-router' //引入vue-router.js文件

import Home from '../views/Home.vue' //引入views文件夹下的首页.vue文件内容

Vue.use(VueRouter) //引入vue-router.js文件中的VueRouter对象

const routes = [

{ path: '/', component: Home }, //首屏加载的页面

{//其它页面只有在用到时才懒加载

path: '/about',

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

}

]

const router = new VueRouter({

routes //routes:routers

})

export default router //模块化开发: 抛出router对象，供其它文件引用

//如果没有这句话，其它文件，引用不了router对象。

d. 全局组件:

1). 集中创建在src/components文件夹下

2). 每个全局组件一个.vue文件

3). 在定义组件时，全局组件和页面组件的内容没有任何差别，也都是包含三部分

a. <template>组件的HTML内容</template>

b. <style scoped>组件的css内容</style>

c. <script>组件对象的js内容</script>

4). 必须在main.js中，引入components下的全局组件.vue文件，然后再用Vue.component()函数，将引入的组件内容变成全局组件:

import MyHeader from "./components/MyHeader"

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

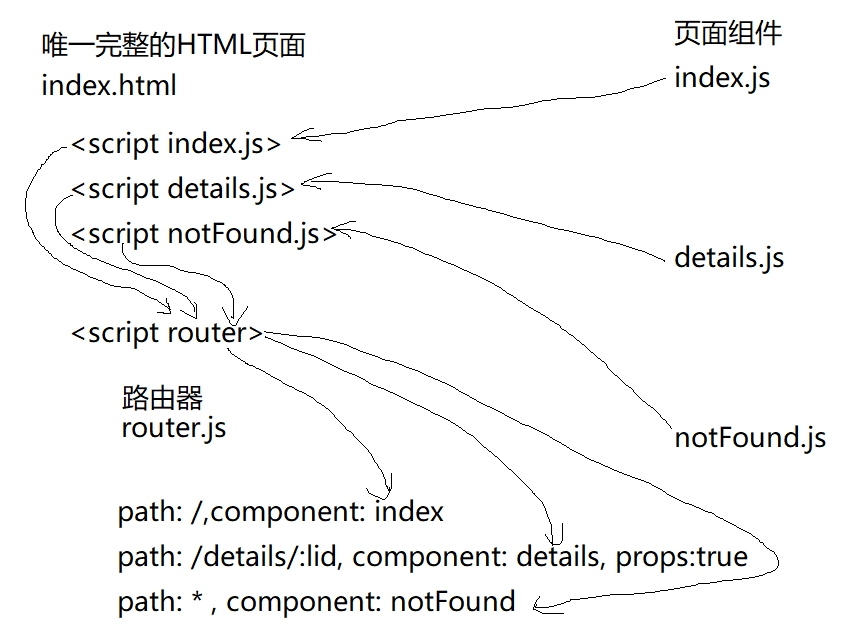
5). 结果在src/views文件夹下的页面.vue中可用

<my-header></my-header>引入全局组件页头

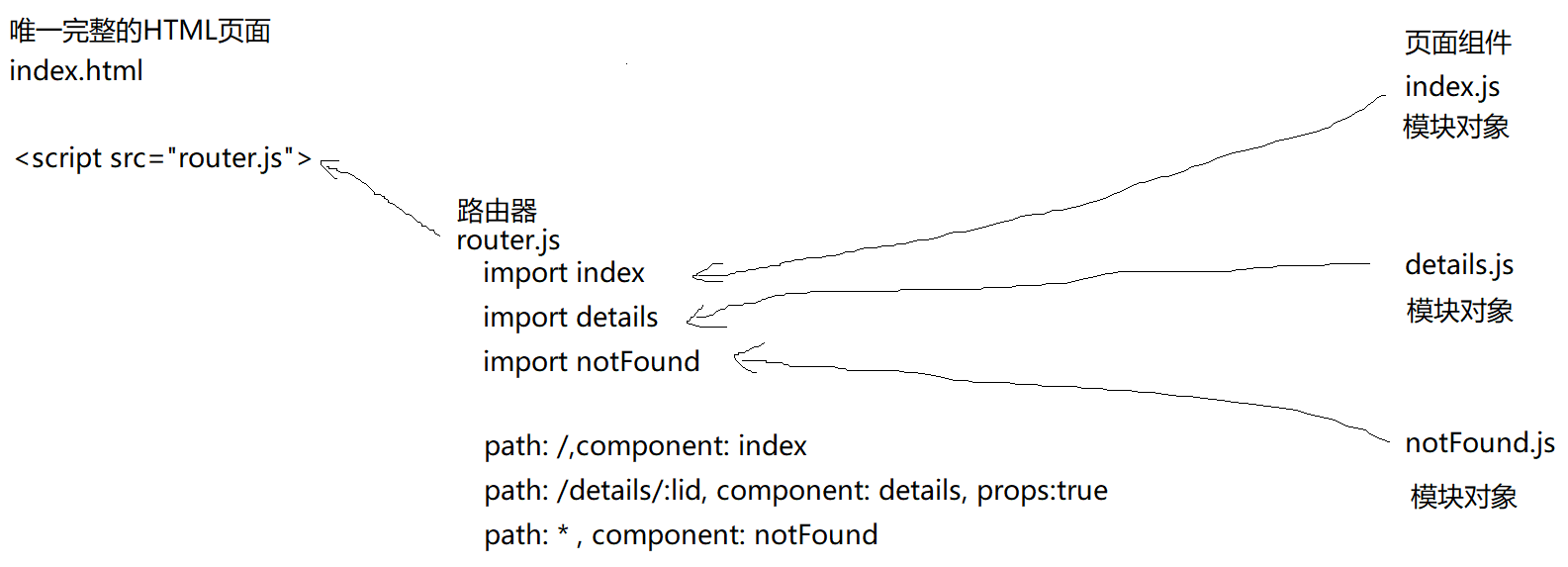
二. 模块化开发:

1. 什么是: 将每个页面和每个组件都看作是一个模块对象。

2. 问题: 没有模块化开发时，网页中一个对象要使用其他组件对象，必须先将所有组件对象都引入.html中集中起来。再一个页面内，挑选哪个对象使用哪个组件。——不直观，不便于理解，不便于维护



3. 模块化开发好处: 哪个模块想使用另一个模块中的内容，可直接引用！不需要全都引入到html文件中，再使用。——直观，便于理解，便于维护



4. 如何：2步:

(1). 组件对象需要创建js对象，并将js对象抛出到外部，供别人选择引用

export default {//new Object()

对象的js成员

}

(2). 其他组件可使用以下方式，引入另一个组件抛出的组件对象

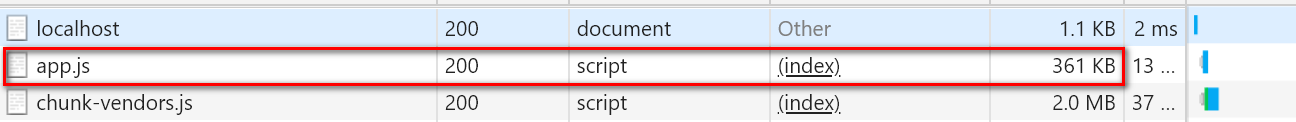
import 自定义变量名 from "另一个组件文件的相对路径"

(3). 结果: 在import所在的组件中，可通过自定义变量来使用引入的另一个组件的内容

5. 特例: 如果一个组件文件中没有js代码，则不需要export default，文件本身默认就是一个模块对象，就可以被别人import导入！

三. 懒加载:

1. 问题: 默认脚手架会将所有以import方式引入的组件的HTML+CSS+JS等编译到一个app.js文件中——这个文件会很大！且首屏下载慢！



2. 解决: 懒加载

3. 2种:

(1). 在编译时，将不同的组件编译到不同的js文件中，首屏加载时，先请求首屏组件的js，然后在低层悄悄异步下载其余组件的js——后续组件不影响首屏加载速度

a. 如何: 2步都是在src/router/index.js中修改

1). 除首页组件之外，不要用import方式过早引入其余页面组件

2). 除首页之外，其余路由字典中的对象，一律采用懒加载

{

path: '/details/:lid',

//只有当用户访问到/details/:lid路径时，才临时执行箭头函数，才临时执行import(...)

///\*\*/注释不能删除，它为将来分块打包时，当前组件所在的js文件起文件名。

//比如: webpackChunkName: "details"

//结果: Details.vue会被编译为一个details.js文件

component: () => import(/\* webpackChunkName: "details" \*/ '../views/Details.vue'),

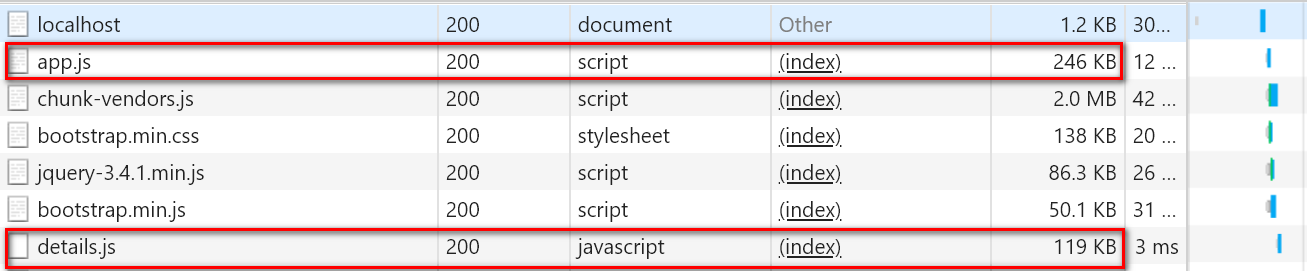
props:true

}

b. 结果:

1). 首屏index.vue和details.vue分别打包为两个js文件

2). 首屏先下载app.js，然后再低层异步下载details.js，不影响首屏加载速度



c. 问题: 不管用户是否会访问详情页，都始终悄悄下载details.js文件——偷跑流量

(2). 在编译时，将不同的组件编译到不同的js文件中，首屏加载时，先请求首屏组件的js，但是如果用户暂时不访问其他组件时，就不提前下载其他组件的js文件

a. 如何:

1). 前两步和第一种方法相同！

2). 但是需要多一步: 在脚手架项目的根目录创建一个文件vue.config.js，在其中添加如下代码:

module.exports={

chainWebpack:config=>{

config.plugins.delete("prefetch") //取消预加载

}

}

b. 结果:

1). 首屏加载时只加载app.js，只包含首页组件内容

2). 只有用户访问/details/5时，才临时下载details.js文件

如果用户不访问/details/5时，是不会提前下载details.js文件的。——省流量

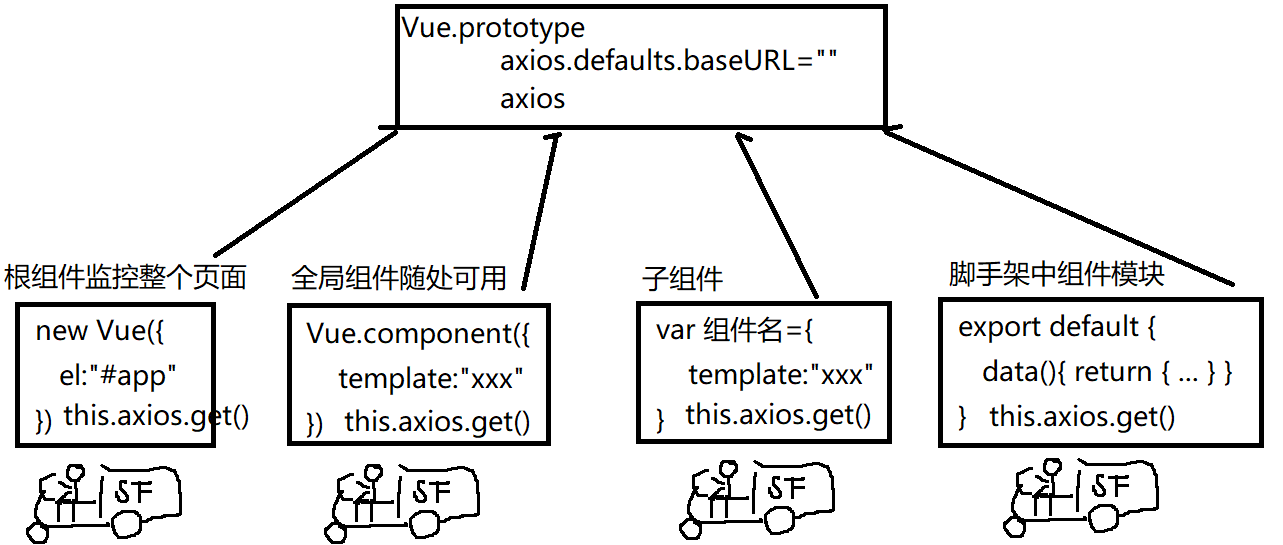
c. 问题: 其他组件加载可能稍慢

四. 脚手架中axios的使用:

1. 问题: 笨办法，可以在每个组件中都引入axios，繁琐！

2. 解决: 其实只在main.js中引入一次axios，然后将axios放入Vue的原型对象中。所有组件对象都可以使用this.axios方式来使用统一的axios对象。

3. 原理: 因为所有组件都是Vue类型的子对象，都继承自Vue的原型对象。所以，在vue的原型对象中添加一个axios，则所有子对象不用重复创建，就可直接使用。



4. 如何:

1). main.js中:

import axios from "axios"

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

Vue.prototype.axiso=axios;

2). 其余任何组件内:

this.axios.get("/相对路径").then(result=>{ ... result.data... })