准备: 微信小程序: WEB问题速查 小恐龙图标

1. 历届师兄师姐在今天课程中曾经问过的高频的经典的问题清单和解答: 小程序->首页->VUE->day03、day04

自己可能没有发现问题，但是想知道别的同学都问过哪些问题

2. 今日课程对应的重点案例的视频以及鄙视面试所需的扩展知识视频：小程序->在线->VUE->day03、04

(学有余力!!!的同学或自己认为自己基础好的同学必须看!!!!!!)：

今日扩展视频：

小程序->在线->VUE->day03

6.2 Vue组件化开发todo实现封装组件... ...删除（基础一般的同学不用看！）

7. 组件间传参 todo 父给子，子给父，兄弟间 ref $ref（基础一般的同学不用看！）

模块化开发（7个视频）（基础一般的同学不用看！）

小程序->在线->VUE->day04

1. SPA 单页面应用...路由嵌套（基础一般的同学不用看！）

3. 今日课程中包含的英语单词的翻译：小程序->单词->VUE->day03、day04

英文不太好的同学，可以提前看一下第二天需要用到哪些英文单词。

约定: 2件事:

1. 第三阶段禁止上课跟着敲笔记和敲代码！谁抄笔记抄的勤，谁抄代码抄的勤，谁学的越烂！

2. 每个完整功能的案例/视频，至少写三遍: (如果需要画图，必须亲手画图)

(1). 连代码+注释抄一遍 —— 明白这个例子是要做什么事儿

(2). 保留注释，自己试着看着注释，将代码填回来 —— 复习函数和对象

(3). 删除所有代码和注释，自己试着用自己的话，把注释写回来，再翻译为代码！—— 只有自己可以写出人话的注释，这个例子才算自己会！

如果没有问题的同学: （享受vip一对一答疑的时间还剩2+3天，端午节我不休息）

1. 上课时那句话没听懂，笔记中那句话看不懂，案例中那句话看不懂，立刻问！

2. 小程序问题清单中哪个问题或解答看不懂，立刻问！

3. 小程序视频中哪个视频看不懂，立刻问

4. 个人项目中哪个功能不会做，立刻问

5. 平时刷笔试题时，遇到不会的笔试题，立刻问

正课:

1. 组件化开发

2. SPA

3. 脚手架

一. 组件化开发

对应小程序视频:小程序->在线->vue->day03->6.1 创建单个组件 计数器...  
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的功能区域

2. 为什么: 重用

3. 何时: 今后只要页面中有一个区域会被反复使用，都要先封装为组件，再反复使用组件

4. 如何:

(1). 封装组件: 创建一个组件对象，添加到Vue大家庭中

Vue.component("组件名",{

//其实每个组件都是一个功能完整的小快递员，麻雀虽小五脏俱全！

//所以，一个组件的内容和new Vue()的内容几乎完全一样

~~el:"#app",~~

template:`HTML片段`, //模板，强调: 组件的HTML片段必须包含在一个唯一父元素内

~~data:{ 变量 },~~

data(){ //可反复调用，每次调用都返回一个新的保存数据的对象——老母鸡下蛋

return { //new Object()

变量: 初始值:

... : ...

}

},

//其余完全一样！

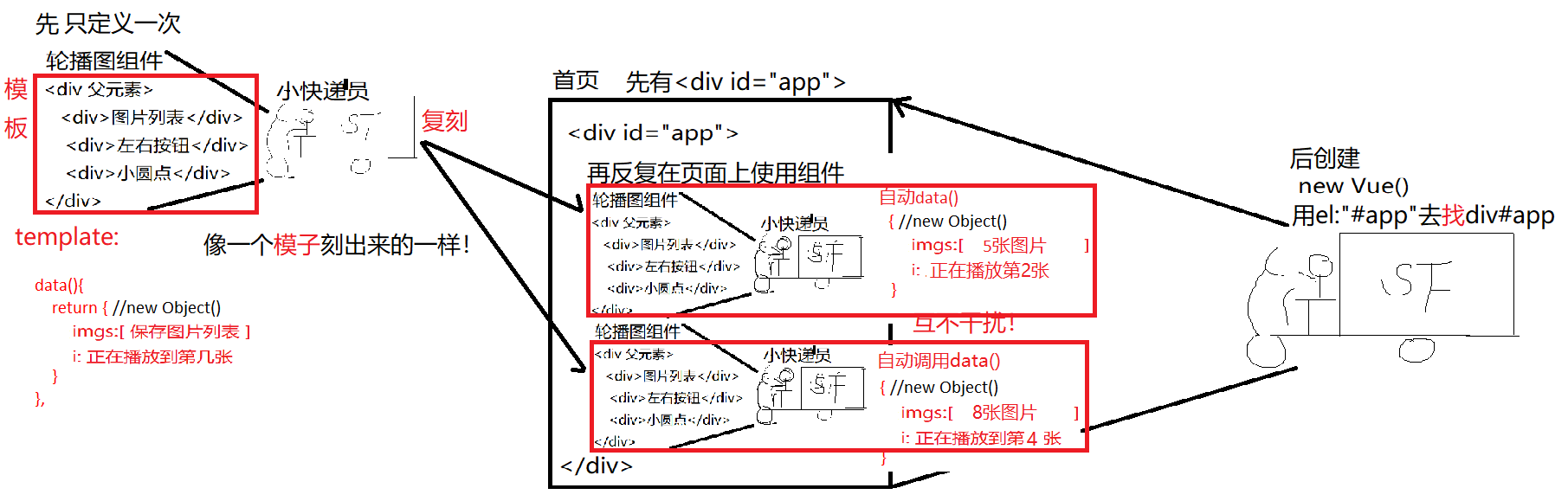
methods:{ 函数 },

computed:{ 计算属性 },

mounted(){ ... }

... ...

})



(2). 在页面中使用组件：vue中的组件其实就是一个可重复使用的自定义HTML标签而已

<组件名></组件名>

强调: 因为HTML不区分大小写，所以如果组件名(也就是将来的自定义标签名)包含多个英文单词时，不要用驼峰命名！应该用-分割多个单词！

5. 原理:

(1). 当new Vue()扫描到不认识的自定义HTML标签时，会回vue家找有没有同名的组件对象

(2). 如果找到同名的组件对象，先把template中保存的HTML片段替换页面中组件名标签的位置

(3). 自动调用一次data()，为本次组件副本创建一个全新的模型对象

(4). 结果: 在当前组件这个小区域中就形成了一个缩微版的new Vue()对象和绑定关系。

6. 示例: 创建一个计数器组件，并反复使用:

1\_component.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>  <script src="js/vue.js"></script>  <script>  //想创建一个计数器组件，反复使用  Vue.component("my-counter",{  //组件要反复复制的HTML模板  //模板中将来哪里可能发生变化:  //本例中span的内容可能发生变化，所以用{{n}}，初始为0  //模板中将来哪里可能触发事件:  //本例中button可能触发事件  template:`<div>  <button @click="minus">-</button>  <span>{{n}}</span>  <button @click="add">+</button>  </div>`,  //因为页面上需要1个变量  data(){  return {  n:0  }  },  //因为页面上需要两个事件处理函数  methods:{  add(){  this.n++;  },  minus(){  if(this.n>0){  this.n--;  }  }  }  })  </script>  </head>  <body>  <div id="app">  <!--页面上反复使用三次计数器组件: -->  <my-counter></my-counter>  <my-counter></my-counter>  <my-counter></my-counter>  </div>  <script>  new Vue({  el:"#app"  })  </script>  </body>  </html>  运行结果: |

对应小程序视频:小程序->在线->VUE->day03 6.2 组件化开发...

7. 组件化开发:

(1). 什么是: 拿到一个网页之后，先将页面划分为多个组件区域，分头并行开发多个组件。最后再将所有组件拼成一个页面展现给用户。

(2). 为什么: 2个主要原因：

a. 可以多人并行开发一个页面——快！

b. 松耦合，一个人出错，不牵连大家！

(3). 何时: 今后所有使用前端框架开发的项目，全都采用组件化开发

(4). 如何:

a. 拿到页面先划分有几个组件: 2个原则

1). 区域划分

2). 是否重用

b. 在页面之外创建独立的组件.js文件，包含组件的HTML+CSS+JS+数据

c. 创建一个完整的.html文件，将外部的组件.js文件内容都引入到.html文件中，在html文件中指定位置使用组价标签，将多个组件合为一个页面.

(5). 问题: 有些子组件，只有放在规定的父组件内才有效，才有意义。一旦出了父组件就没有意义了！

(6). 解决: 其实: vue中的组件大致分为三大类:

a. 根组件root: 一个页面只有一个，监控整个页面的组件

new Vue({

... ...

})

b. 全局组件: 可在任何位置反复使用的页面独立功能区域——没有限制

Vue.component("组件名", { ... })

c. 子组件: 只能在某个指定的父组件内才能使用的小组件，2步:

1). 创建子组件对象:

var 子组件名={

//和Vue.component中的格式完全一样！

template:`HTML片段`,

data(){ return { 变量 } },

... ...

}

强调: 因为子组件名是一个js变量，所以，~~不能用-分割~~多个单词，必须用驼峰命名

2). 在父组件中添加components属性来限制子组件只能在自己范围内使用

Vue.component("父组件",{

template:`HTML片段`,

data(){ return { 变量 }},

... ...

components:{ 子组件对象名 , ... , ... }

})

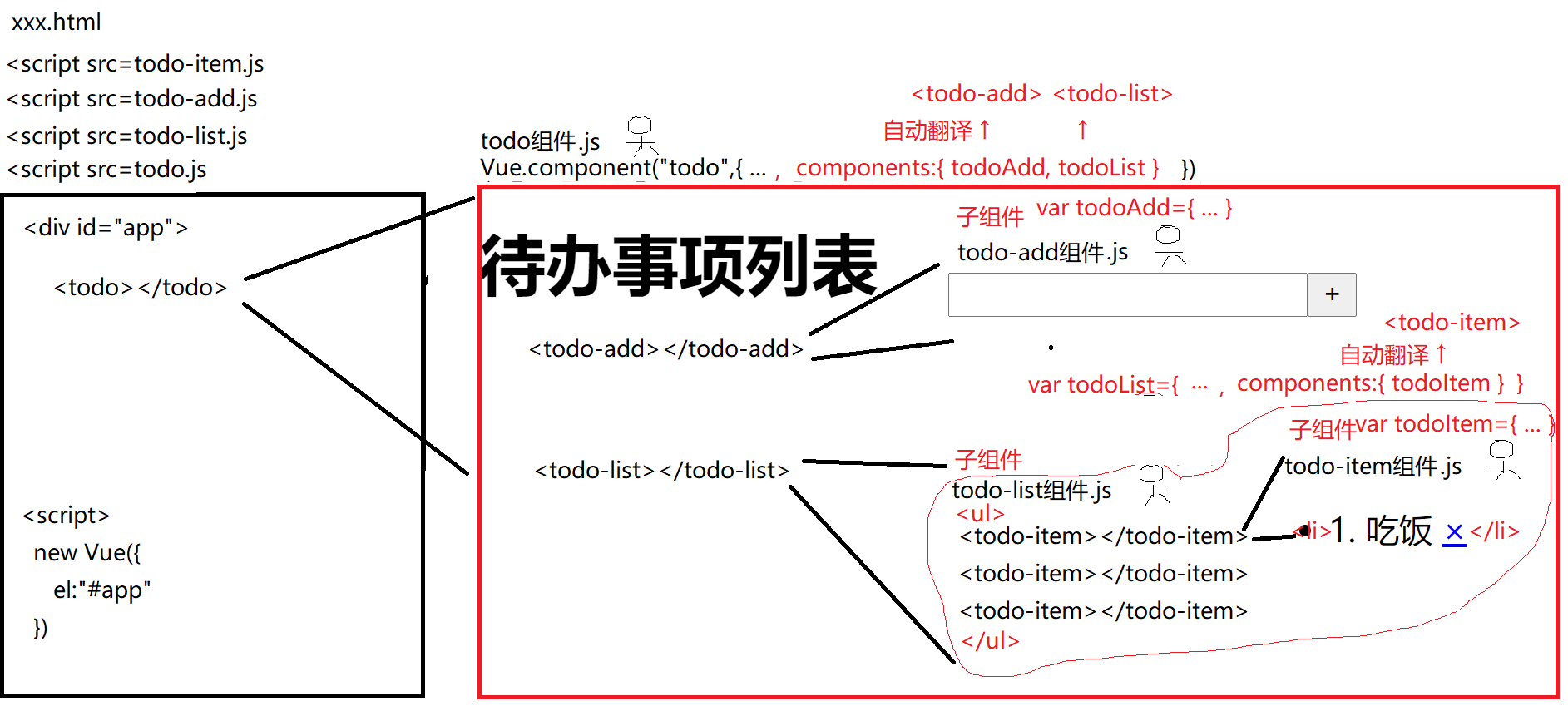
强调: 但是，父组件中components会将驼峰命名的子组件名自动翻译为-分割多个单词，所以在父组件template中使用子组件标签时，依然要使用-分割

比如: components:{ todoAdd, todoList }

自动翻译 <todo-add> <todo-list>

(7). 坑: 因为创建父组件时就需要子组件对象了，所以引入时，比如先引入子组件，再引入父组件！

(8). 示例: 使用组件化开发的方式开发待办事项列表的页面（暂时不含数据）



2\_todo/todo-item.js

|  |
| --- |
| var todoItem={  template:`<li>  1. 吃饭 <a href="javascript:;">×</a>  </li>`  } |

2\_todo/todo-list.js

|  |
| --- |
| var todoList={  template:`<ul>  <todo-item></todo-item>  <todo-item></todo-item>  <todo-item></todo-item>  </ul>`,  //规定todoItem组件只能在当前todoList组件内才能使用，出了todoList组件外使用会报错  components:{ todoItem }  //自动翻译 <todo-item>  } |

2\_todo/todo-add.js

|  |
| --- |
| var todoAdd={  template:`<div>  <input><button>+</button>  </div>`  } |

2\_todo/todo.js

|  |
| --- |
| Vue.component(`todo`,{  template:`<div>  <h1>待办事项列表</h1>  <todo-add></todo-add>  <todo-list></todo-list>  </div>`,  //规定todoAdd和todoList两个组件只能在当前todo组件内才能使用，出了todo组件外使用会报错  components:{ todoAdd, todoList }  //自动翻译 <todo-add> <todo-list>  }) |

2\_todo/todo.html

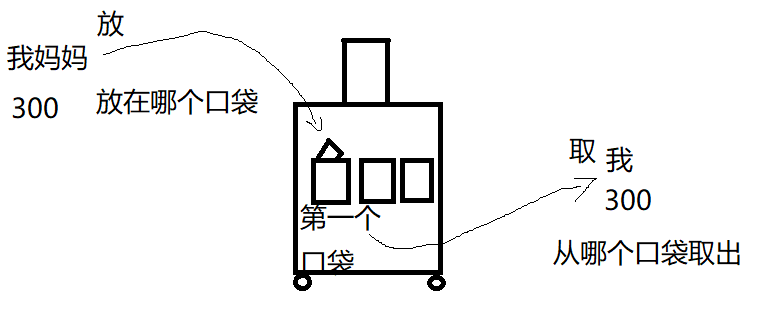
|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>  <meta charset="UTF-8">  <meta name="viewport" content="width=device-width, initial-scale=1.0">  <title>Document</title>  <script src="js/vue.js"></script>  <script src="todo-item.js">  //var todoItem={ ... }) //孙子  </script>  <script src="todo-add.js">  //var todoAdd={ ... }) //儿子  </script>  <script src="todo-list.js">  //var todoList={ ... }) //儿子  </script>  <script src="todo.js">  //Vue.component("todo",{ ... }) //父亲  </script>  </head>  <body>  <div id="app">  <todo></todo>  <!-- <todo-item></todo-item>  报错: <todo-item> - did you register the component correctly  说明: 出了todo，就找不到todo-item了！只能在todo内使用！-->  </div>  <script>  new Vue({  el:"#app"  })  </script>  </body>  </html>  运行结果： |

8. 组件间传参:

(1). 问题: vue中子组件也无权直接使用父组件中的数据变量

(2). 解决: 父给子传值！

(3). 如何: 2步:



a. 父组件中：父将自己的变量通过:绑定的方式交给子组件

父组件对象: {

data(){

变量: 值

},

template:`<div>

... ...

<子组件 :自定义属性名="变量"></子组件>

</div>`

强烈建议: 子组件自定义属性名尽量和父组件中变量名保持一致！

b. 子组件中: 子组件从父组件放变量的自定义属性中取出父组件给的变量值

子组件对象:{

props:[ "自定义属性名", ... ]

data(){

return { 自己的变量 }

},

template:`<div>

... ...

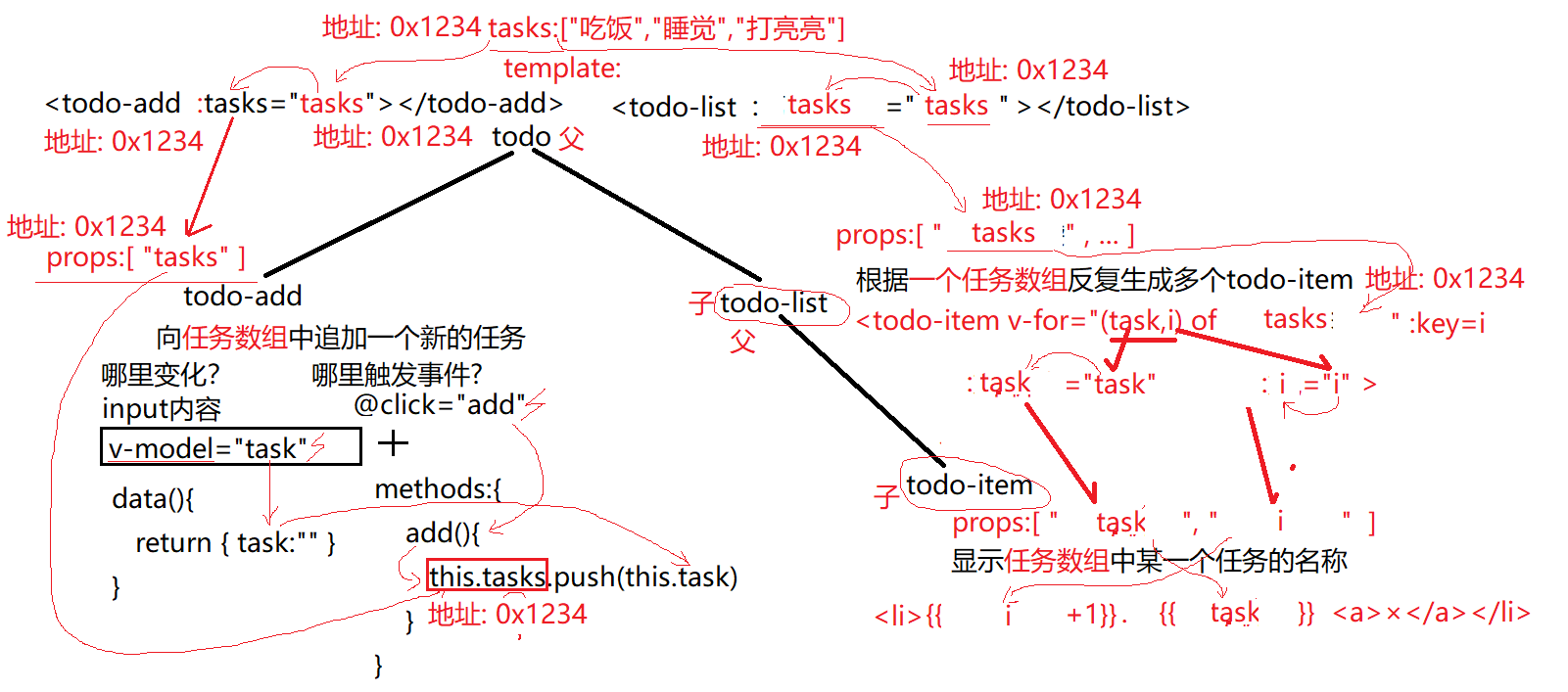
//结果: 子组件中即可使用data()中自己的变量，也可以使用props中父组件给的变量。且props中变量的用法和data中自己变量的用法完全一样！只不过来源不同而已！

... ...

</div>`

}

(4). 示例: 使用父给子传参实现todo案例中任务列表和添加功能。



3\_todo/todo-item.js

|  |
| --- |
| var todoItem={  //从父组件todoList给的两个自定义属性中取出父组件给的变量值  props:["task","i"],  //props中的变量和data中自己的变量用法一样，也可用于绑定和指令等。只不过props来源于外部，而data是内部定义的而已。  template:`<li>  {{i+1}}. {{task}} <a href="javascript:;">×</a>  </li>`  } |

3\_todo/todo-list.js

|  |
| --- |
| var todoList={  //从父组件给的自定义属性tasks中取出父组件给的tasks变量值  props:[ "tasks" ],  //props中的变量用法和自己data中变量用法完全一样，都可用于绑定、指令等。  template:`<ul>  <!--根据父组件中tasks数组中的元素个数，反复生成多个<todo-item>子组件-->  <todo-item v-for="(task,i) of tasks" :key="i" :task="task" :i="i"></todo-item>  </ul>`,  //规定todoItem组件只能在当前todoList组件内才能使用，出了todoList组件外使用会报错  components:{ todoItem }  //自动翻译 <todo-item>  } |

3\_todo/todo-add.js

|  |
| --- |
| var todoAdd={  props:["tasks"],  template:`<div>  <input v-model="task"><button @click="add">+</button>  </div>`,  data(){  return {  task:""  }  },  methods:{  add(){  this.tasks.push(this.task);  //在添加成功后，清空task变量值，也就自动清空了文本框中残留  this.task="";  }  }  } |

3\_todo/todo.js

|  |
| --- |
| Vue.component(`todo`,{  template:`<div>  <h1>待办事项列表</h1>  <todo-add :tasks="tasks"></todo-add>  <todo-list :tasks="tasks"></todo-list>  </div>`,  //因为三个子组件都需要访问一个任务数组  //所以在父组件中统一保存一个任务数组供所有子组件共同使用！  data(){  return {  tasks:["吃饭","睡觉","打亮亮"]  }  },  //规定todoAdd和todoList两个组件只能在当前todo组件内才能使用，出了todo组件外使用会报错  components:{ todoAdd, todoList }  //自动翻译 <todo-add> <todo-list>  }) |

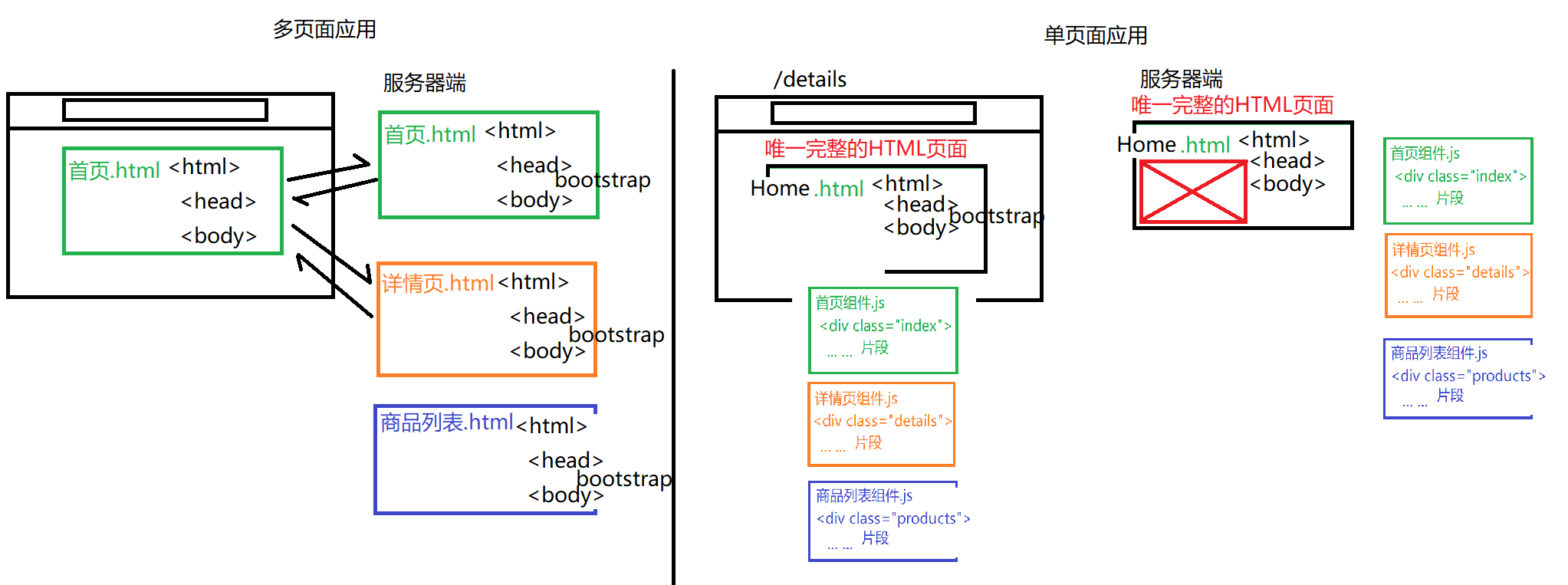
3\_todo/todo.html

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>  <meta charset="UTF-8">  <meta name="viewport" content="width=device-width, initial-scale=1.0">  <title>Document</title>  <script src="js/vue.js"></script>  <script src="todo-item.js">  //var todoItem={ ... }) //孙子  </script>  <script src="todo-add.js">  //var todoAdd={ ... }) //儿子  </script>  <script src="todo-list.js">  //var todoList={ ... }) //儿子  </script>  <script src="todo.js">  //Vue.component("todo",{ ... }) //父亲  </script>  </head>  <body>  <div id="app">  <todo></todo>  <!-- <todo-item></todo-item>  报错: <todo-item> - did you register the component correctly  说明: 出了todo，就找不到todo-item了！只能在todo内使用！-->  </div>  <script>  new Vue({  el:"#app"  })  </script>  </body>  </html>  运行结果： |

二. SPA: Single Page Application

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

1. 什么是单页面应用: 整个应用程序只有一个唯一完成的HTML页面。其它所谓的"页面"，其实都变成了组件。所谓的切换页面，其实切换的不是.html文件，而是一个html页面中的组件部分。



2. 为什么:

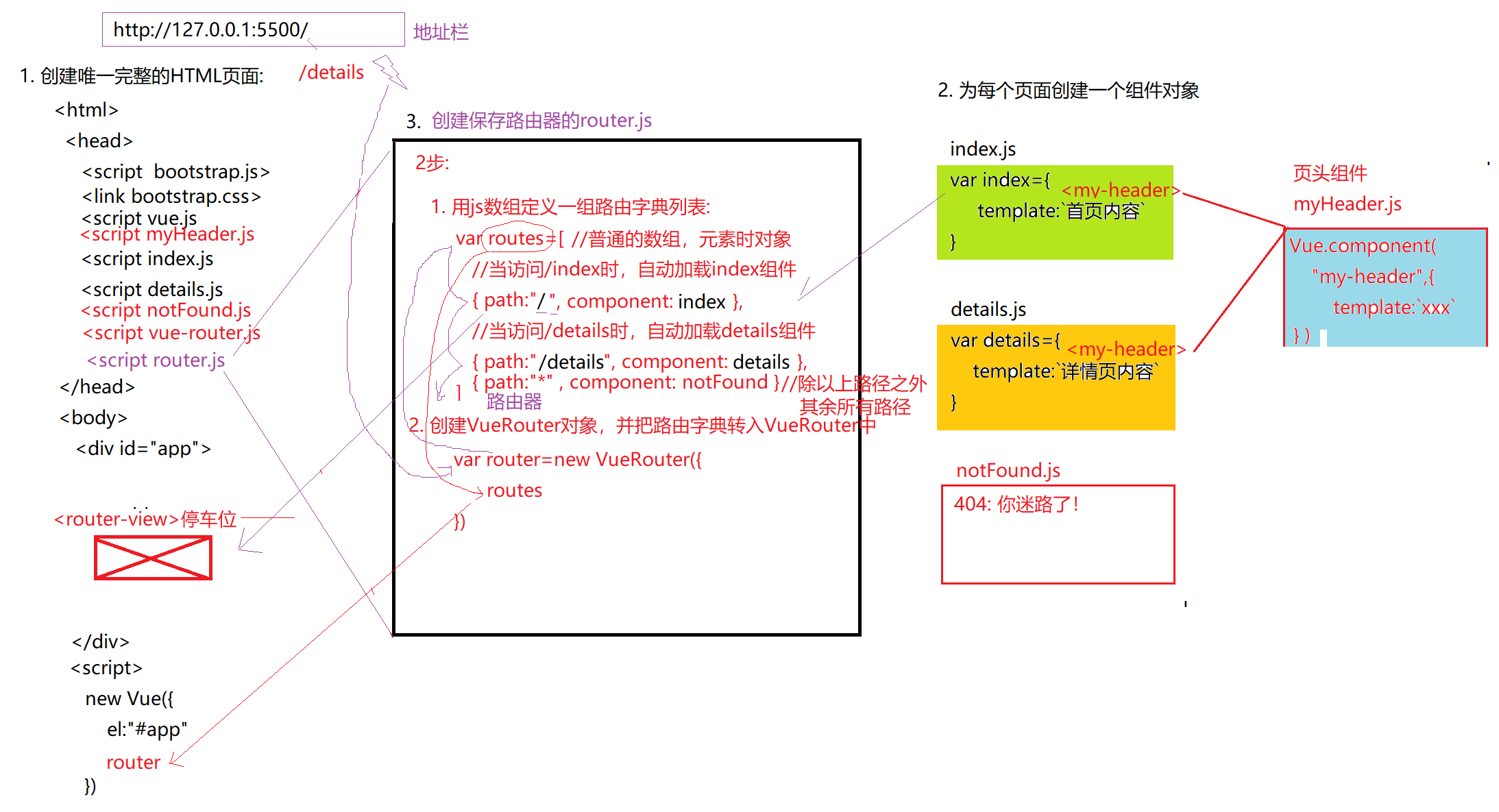
|  |  |  |
| --- | --- | --- |
|  | 多页面 | 单页面 |
| 请求次数 | 每切换一次页面，都要重新向服务器发送请求——请求次数多 | 只在首次加载时下载所有的组件和唯一完整的HTML页面，今后切换页面时，其实只是在客户端本地切换同的组件显示而已，不会反复向服务器发送请求——请求次数少 |
| 页面加载效率 | 每次切换页面都要废掉之前整棵DOM树，下载新的HTML重建整棵DOM树。反复下载+重建整棵DOM树，效率低！ | 每次切换页面时，整棵DOM无需重建，只更换局部的组件内容即可。且不用重新下载组件内容。无需下载+无需重建整棵DOM树，效率高！ |
| 公共资源:  bootstrap.css  jquery.js | 每次切换页面都要重新下载页面，页面中如果包含一些公共的资源，比如bootstrap和jquery等，也需要反复下载。——增大网络流量 | 因为公共资源只在唯一完整的HTML开头引用一次，所以只在首次加载时才下载一次。只有更换页面，因为.html文件不变，所以无需重复下载公共资源！ |
| 页面切换动画 | 多页面应用不可能实现页面切换。  因为页面切换动画过程中需要同时显示前一个页面的后一半和后一个页面的前一半。可是多页面应用不可能让两个页面同时停留在客户端！ | 因为所有组件对象都已经在客户端了。所以完全有可能在切换过程中，显示前一个组件的后一半和后一个组件的前一半。完全有可能实现页面切换动画 |

3. 单页面应用缺点: 首屏加载慢：

(1). 因为第一次请求时，就会把所有页面组件的内容都下载下来！

(2). 已经解决了：？

4. 如何: 3大步

s

(1). 先创建一个唯一完成的HTML页面

a. 是一个基本的支持vue框架的页面结构

b. 如果有公共组件比如bootstrap.css, jquery.js和bootstrap.js，也要在唯一完整的HTML中引入

c. 在<div id="app">内必须写一个<router-view>元素为将来的页面组件占位

d. 之后还要引入所有页面组件和vue-router.js组件，以及保存VueRouter对象的router.js文件

e. new Vue()中除了el:之外，还要添加router成员

new Vue({

el:"#app",

router

})

(2). 为每个"页面"分别创建一个组件对象保存各自页面的内容:

强调: 每个"页面"组件，必须采用子组件对象方式创建

var 组件名={

组件内容

}

(3). 创建路由器对象根据地址栏的变化，自动切换显示不同的"页面"组件: 2步

a. 创建单独的router.js，保存路由字典和路由器对象

b. 创建路由字典数组:

//路由字典: 保存一组路径与组件间对应关系的数组

var routes=[

{ path:"/相对路径", component: 路径对应的组件对象名},

...

]

c. 创建路由器对象并将路由字典放入路由器对象中

var router=new VueRouter({ routes })

5. 优化路由字典:

(1). 问题: 通常默认首页是不需要输入任何相对路径就可以访问的

解决: var routes=[

{path:"/", component: 首页组件}

...

(2). 问题: 如果用户输入错误的路径，不能什么都不显示，应该显示404找不到网页

解决: 添加一个专门的404页面组件，并引入到页面中。并且在路由字典结尾添加:

{path:"\*", component: 404组件对象名}

\*表示除之前路径之外所有错误的路径，都导向404页面组件

6. 公共页头部分: 一般创建为全局组件，并添加到<router-view>之外，就不会被替换，而保证每个页面加载时都有！

问题: 有些页面(比如404)不想显示页头

解决: 页头不要放在唯一完整的HTML页面中，应该只放入那些需要页头的组件中！

7. 路由跳转:

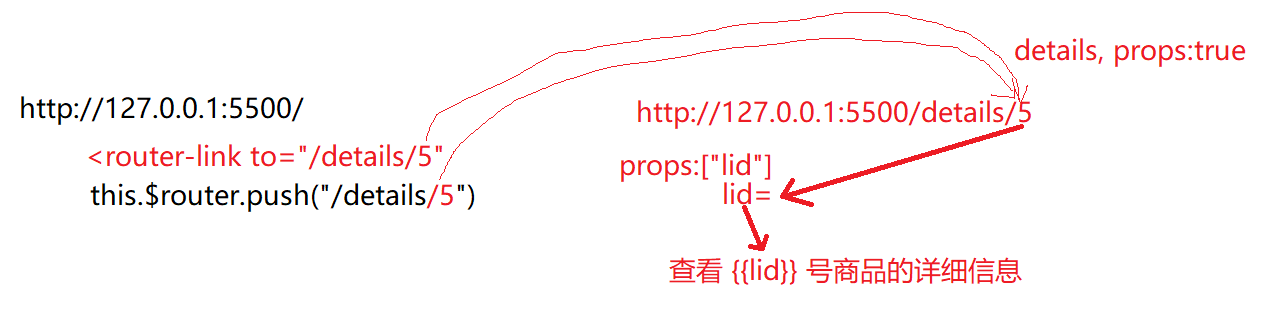
(1). 在HTML中写死的方式: ~~代替<a>~~

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

(2). 在js程序中也能用程序跳转:

固定套路: 没有为什么: this.$router.push("/相对路径")

8. 路由传值:



(1). 修改路由字典中的相对路径，允许一个路径携带参数

{ path:"/相对路径/:变量名", component:组件名, props:true }

props: 让路径中的变量值自动进入下一个页面组件，成为props中的一个变量

(2). 上个页面跳转时可使用: /相对路径/变量值

(3). 在下个页面中用props:[ "变量名" ] 获取传过来的变量值

(4). 结果在下个页面组件中可以用变量名用作绑定或js程序都行！

(5). 坑: 一旦一个路径写了变量，则再用不带变量值的路径，就无法进入了！

比如: path:"/details/:lid"

将来: /details/5 可以进入

/details/10 可以进入

/details 禁止入内！导向404

9. 示例: 制作一个单页面应用，包含首页和详情页

4\_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>    <script src="js/vue.js"></script>    <!--引入全局页头组件-->    <script src="myHeader.js"></script>    <!--引入所有页面组件-->    <script src="index.js">      //var index={ template:`首页内容` }    </script>    <script src="details.js">      //var details={ template:`详情页内容` }    </script>    <script src="notFound.js">      //var notFound={ template:`404:xxx` }    </script>    <!--引入SPA核心组件vue-router.js（官方）-->    <script src="js/vue-router.js"></script>    <!--引入我们自己创建的路由器和路由字典-->    <script src="router.js">      //var routes=[ 路由字典 ];      //var router=new VueRouter({ routes })    </script>  </head>  <body>    <div id="app">      <!-- <my-header></my-header> -->      <!--为将来要加载的页面组件占位-->      <router-view></router-view>    </div>    <script>      new Vue({        el:"#app",        //比如将引入的路由器对象添加new Vue()中，才能让new Vue()在扫描时认识<router-view>占位。        router      })    </script>  </body>  </html> |

index.js

|  |
| --- |
| var index={    template:`<main>      <my-header></my-header>      <h1 style="color:lightGreen">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="toDetails">查看10号商品的详情</button>    </main>`,    methods:{      toDetails(){        this.$router.push("/details/10");      }    }  } |

details.js

|  |
| --- |
| var details={    //接住地址栏中的lid变量的值（比如设置props为true才能这样用）    props:["lid"],    template:`<main>      <my-header></my-header>      <h1 style="color:orange">这里是{{lid}}号商品的详情</h1>    </main>`  } |

notFound.js

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

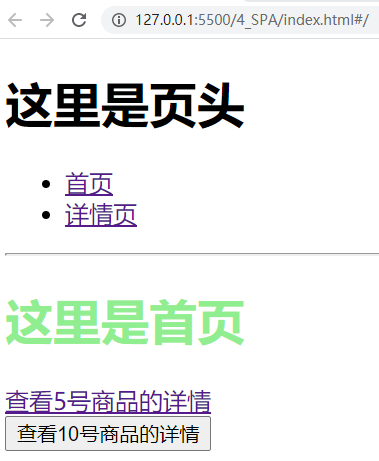
router.js

|  |
| --- |
| var routes=[    //默认首页    {path:"/", component:index},    //想让详情页可以带商品编号参数lid    {path:"/details/:lid", component:details, props:true},    //除以上路径之外，其余路径都导向404页面    {path:"\*", component:notFound}  ];  var router=new VueRouter({ routes }); |

myHeader.js

|  |
| --- |
| Vue.component("my-header",{    template:`<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>    </header>`  }) |

运行结果:







总结:

1. MVVM: 界面View+模型对象Model+视图模型ViewModel

2. Vue的绑定原理: 访问器属性+虚拟DOM树(+观察者模式)

3. 虚拟DOM树的优点:

(1). 小，仅包含可能变化的元素

(2). 遍历查找快

(3). 更新效率高，只更新受影响的元素

(4). 避免重复编码，已封装了DOM的增删改查操作

总结: 今后所有VUE功能三步:

1. 创建界面(增强)，界面中标记变量和事件处理函数

2. 创建new Vue()对象，用el:"选择器"，找到自己负责的页面区域的唯一父元素

3. 在new Vue()中添加data和methods，分别保存界面所需的所有变量和事件处理函数

总结:

1. 绑定语法(1种)和指令(13种):

(1). 只要元素的内容随变量自动变化: {{}} v-text

(2). 只要元素的属性值随变量自动变化: :

(3). 控制一个元素显示隐藏: v-show （display:none）

(4). 控制两个元素二选一显示: v-if v-else （删除元素）

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

(6). 反复创建多个相同结果的元素: v-for :key="下标"

(7). 绑定事件: @ $event

(8). 防止用户短暂看到{{}}: v-cloak v-text

(9). 只要绑定的内容是HTML片段: v-html

(10). 只要希望一个元素只在首次加载时绑定一次，之后都不会自动更新: v-once

(11). 只要不希望元素内容中的{{}}被vue解析，而是保持原样显示: v-pre

(12). 只要希望随时获得页面上用户在表单元素中修改的新值: v-model

2. 绑定样式:

(1). 如果希望精确修改某一个css属性的值时，绑定style内联样式:

a. 为每个要修改的css属性分别设置变量

<元素 style="固定不变的css属性 " :style="{ css属性1: 变量1, css属性2: 变量2, ... }"

data:{

变量1: css属性值1,

变量2: css属性值2

}

b. 为整个style属性设置一个变量对象，其中包含多个css属性和属性值

<元素 style="固定不变的css属性 " :style="xxxStyle"

data:{

xxxStyle:{

css属性1: css属性值1,

css属性2: css属性值2

}

}

(2). 如果希望批量修改一个元素的多个css属性的值时，就要绑定class属性:

a. 为每个class分别设置条件

<元素 class="不变的class " :class="{class1: bool表达式或变量, class2:bool表达式或变量 }"

bool表达式或变量的值为true就应用，false就不应用

b. 为整个class属性设置一个变量对象，其中包含多个备选的class名极其对应判断条件

<元素 class="不变的class " :class="xxxClass"

data:{

xxxClass:{

class1: true或false, //true就应用，false就不应用

class2: true或false //true就应用，false就不应用

}

}

3. 自定义指令: 2步

(1). 希望在页面首次加载时就自动对元素执行一些初始化DOM操作，添加自定义指令:

Vue.directive("指令名",{

inserted(domElem){

对domElem执行DOM操作

}

})

(2). 使用自定义指令: <元素 v-自定义指令名>

4. 计算属性: 如果一个变量值不是现成的，需要根据其他变量的值经过复杂计算才能获得

(1). 定义:

new Vue({

... ,

computed:{

自定义属性名(){

计算过程

return返回值

} //Vue会缓存自定义属性的计算结果，反复使用，避免重复计算！

}

})

(2). 使用计算属性: <元素>{{自定义属性名}}</元素>

5. 过滤器: 如果变量的原始值不能直接给用户看，需要加工后才能给用户看时

(1)不带参过滤器:

a. 添加过滤器:

Vue.filter("过滤器名", function(oldVal){

return 根据不同的oldVal加工出不动的新值

})

b. 使用过滤器: <元素>{{ 变量 | 过滤器 }}</元素>

(2). 带参数过滤器：

a. 添加过滤器: 1 2

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

return 根据不同的oldVal加工出不动的新值

})

b. 使用过滤器: <元素>{{ 变量 | 过滤器(自定义实参) }}</元素>

(3). 多个过滤器连用: <元素>{{ 变量 | 过滤器1 | 过滤器2 | ...</元素>

6. axios: 只要在vue框架中发送ajax请求

(0). axios.default.baseURL="http://所有服务器端接口统一的域名部分"

(1). axios.get("/接口名", { params:{ 参数:值, 参数:值, ... } })

.then(result=>{

this.xxx=result.data ...

})

(2). axios.post("/接口名", "参数1=值1&参数2=值2&...")

.then(result=>{

this.xxx=result.data ...

})

7. vue生命周期: 4个阶段 8个钩子函数

beforeCreate() 既没有data和变量，又没有虚拟DOM树和页面元素

(1). 创建阶段 create

created() 虽然有data和变量，但是没有虚拟DOM树和页面元素

beforeMount() 虽然有data和变量，但是没有虚拟DOM树和页面元素

(2). 挂载阶段 mount

mounted(){ //此时既有了data对象和变量，又有了虚拟DOM树和页面元素

axios.get(...).then(result=>{

... ...

})

}

beforeUpdate()

(3). 更新阶段 update

updated()

beforeDestroy()

(4). 销毁阶段 destroy

destroyed()

总结: this 7种:

1. obj.fun() this->obj

2. new Fun() this->new正在创建的子对象

3. 类型名.prototype.共有方法=function(){ ... } this->将来调用这个共有方法的.前的子对象

4. fun() 和 (function(){ })() 和回调函数中的this->window

5. 事件处理函数中的this->当前正在触发事件的这个元素对象

6. $jquery对象.click(function(){ this->将来点击的那个DOM元素对象})

7. new Vue()中的几乎所有this，都指向当前new Vue()对象。（回调函数要用箭头函数）

今日对应小程序视频:

小程序->在线->vue->day03 6.1 创建单个组件 计数器...

6.2 组件化开发...

小程序->在线->VUE->day04 1. SPA 单页面应用

作业:

1. 复习今日小程序问题清单(小程序->首页-> VUE ->day03、day04)，预习明日小程序问题清单(小程序->首页->VUE->day05)

2. 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

3. 复习SPA

小程序->在线->VUE->day04 1. SPA 单页面应用

4. 回看TMOOC第一阶段NODEJS第一天、第二天 模块相关内容——重要

5. 看小程序视频尝试自己创建vue脚手架项目

小程序->在线->VUE->day04 2. 使用vue/cli 工具创建学子商城脚手架项目

6. 看小程序视频预习vue脚手架结构：并修改脚手架内容为自定义内容:

小程序->在线->VUE->day04 3. vue脚手架文件夹结构