我为大家准备了: 微信小程序——“WEB问题速查”——每天必须跟着看

1. 历届师兄师姐的问题清单: 小程序->首页->VUE->day03~day04 点击右侧列表中的问题，就可弹出窗口，查看问题的详情

2. 当天重点视频片段: 小程序->在线->VUE -> day03~day04

3. 当天的单词列表: 小程序->单词-> VUE -> day03~day04

如果tmooc上的视频有问题看不了，可以直接看小程序中的视频

小程序中内容到鄙视面试前看完就行！不用现在都看完！

关于提问: (剩2天)直接私信我是最快的解决途径——谁问的问题越多，谁的培训费交的越值

1. 上课时哪个问题没听明白，立刻问

2. 笔记中哪句话/哪个图不明白，立刻问

3. 小程序中问题清单里哪个问题看不懂，立刻问

4. 如果个人项目中功能不会做，立刻问

5. 刷笔试题时，遇到不会的笔试题，立刻问

正课:

1. axios

2. 生命周期

3. 组件

4. 组件化开发  
5. SPA

一. axios

1. 每种框架或函数库中都提供了专门的发送ajax请求的函数。用哪个框架，就要使用对应的哪种ajax函数。

比如: jquery: $.ajax() vue: axios.get() react: xxx angularJS: xxx 微信小程序:xxx

2. 什么是: axios是基于Promise技术的，专门发送ajax请求的函数库——几乎专用于vue，但是其他框架也可使用

3. 如何:

(1). 引入axios.js，获得一个axios对象

(2). 因为大部分服务器端接口的域名和端口号部分都是一致的，只有服务端接口地址的相对路径部分不一致。所以axios允许先统一配置所有请求的基础路径

axios.defaults.baseURL="服务器端接口的统一基础路径"

(3). 发送get请求:

axios.get("/接口相对路径",{

params:{ 请求参数1: 值1, 请求参数2: 值2, ... }

}) 自动翻译

//延迟!!!

.then(result=>{ ~~//function(result){~~

//成功收到响应结果后，自动执行.then中的箭头函数

//result并不直接是返回结果

//result.data 才是返回结果！

//且返回结果已经被自动转为js数组或对象了

})

结果: 服务器接口地址?请求参数1=值1&请求参数2=值2&...

(4). 发送post请求:

axios.post("/接口相对路径","请求参数1=值1&请求参数2=值2&...")

//延迟!!!

.then(result=>{

result.data才是服务器端响应结果

})

4. 示例: 使用axios发送三种请求

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <title>Document</title>    <!--axios无需vue可独立运行！-->    <script src="js/axios.min.js"></script>  </head>  <body>    <script>      axios.defaults.baseURL="http://xzserver.applinzi.com";        axios.get("/index")            .then(result=>{              console.log(result.data);            });      axios.get("/details",{params:{ lid:5 }})          .then(result=>{            console.log(result.data);          })      axios.post("/users/signin","uname=dingding&upwd=123456")          .then(result=>{            console.log(result.data);          })    </script>  </body>  </html>  运行效果： |

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。

7. 示例: 测试生命周期钩子函数的自动执行，以及发送axios请求

|  |
| --- |
| <!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="js/axios.min.js"></script>  </head>  <body>    <!--希望在页面上加载首页6个商品的列表-->    <div id="app">      <ul>        <li v-for="(p,i) of products" :key="i">          {{i+1}} | {{p.title}} | {{p.price}}        </li>      </ul>    </div>    <script>      new Vue({        el:"#app",        data:{          products:[]        },        //axios.get()放哪儿，就能让他自动执行？！        beforeCreate(){          console.log(`创建data对象之前触发beforeCreate`)        },        created(){          console.log(`创建data对象之后触发created`)        },        beforeMount(){          console.log(`创建虚拟DOM树之前触发beforeMount`)        },        mounted(){          console.log(`页面挂载完成后触发mounted`)          axios.get("http://xzserver.applinzi.com/index")            .then(result=>{              console.log(result.data);              //将result.data保存到vue中data中的products变量上              this.products=result.data;              //触发vue对象的update生命周期阶段！            });        },        //不是必经阶段！只有data中的变量值被更改时才触发。        beforeUpdate(){          console.log(`修改data中的products变量之前触发beforeUpdate`)        },        updated(){          console.log(`修改完data中的products变量之后触发updated`)        },      })      </script>  </body>  </html>  运行结果: |

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.component("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>

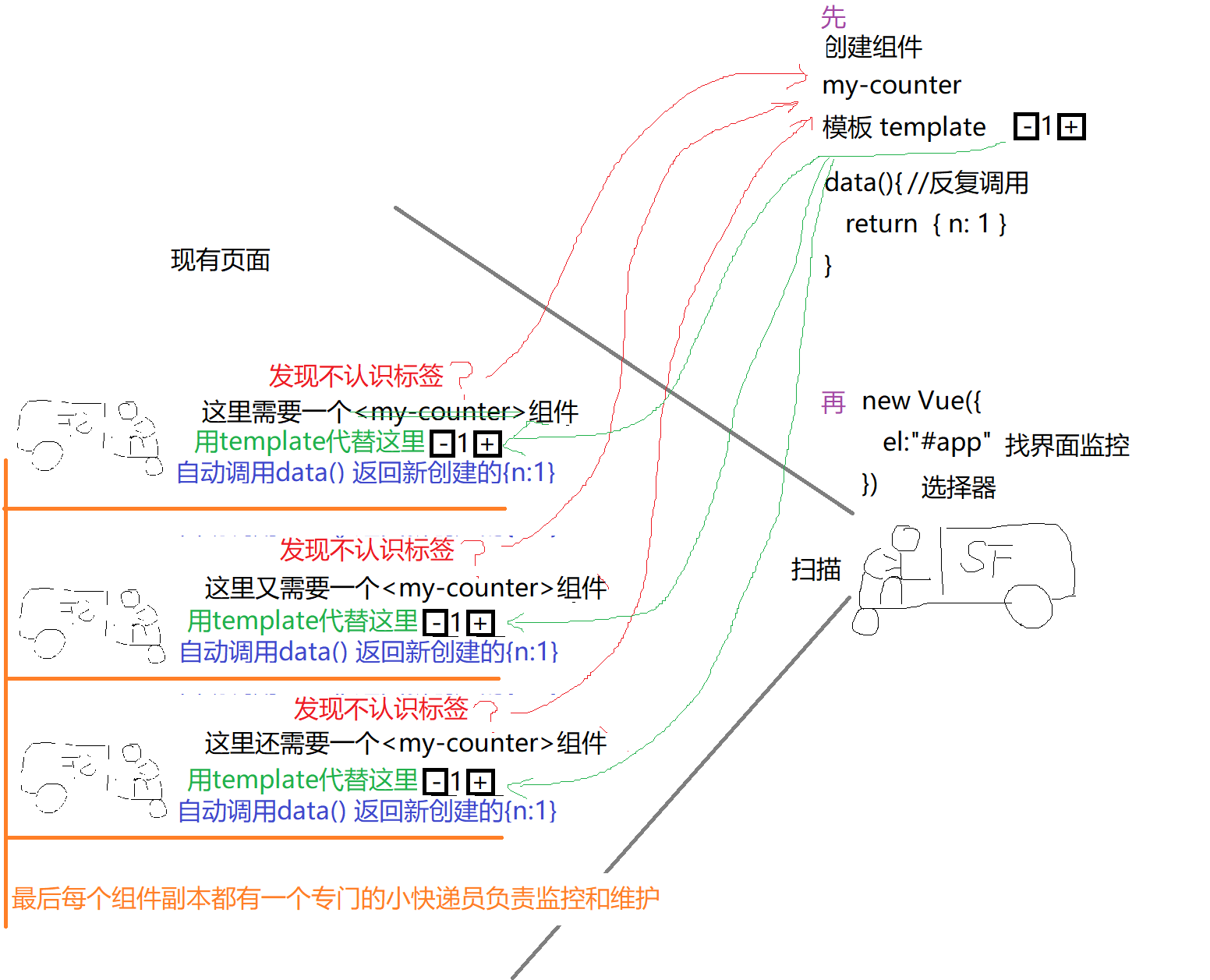
5. 原理:

(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。



6. 示例: 定义一个计数器组件，并在页面中反复使用

|  |
| --- |
| <!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>      //想定义一个计数器组件，点+-号，可更改数量      //1. 创建组件对象, 定义组件名称,多个英文单词用-分割      Vue.component("my-counter",{        //每个组件的制作过程和new Vue()制作过程几乎完全一样        //2. 定义界面模板:template        //要求唯一父元素包裹        //问: 模板中共有几处可能发生变化        //本例中: 只有span的内容可能发生变化,所以用{{n}}标记        //问: 模板中哪些元素可能触发事件        //本例中: 两个button可点击执行+-计算，所以用@click标记        template:`<div>          <button @click="minus">-</button>  <span>{{n}}</span>  <button @click="add">+</button>        </div>`,        //3. 创建模型对象        //问: template模板中共需要几个变量        //本例中: 共需要1个变量n        data(){ //将来会被自动反复调用          return { //反复创建模型对象副本——这里才相当于new Vue()中data对象            n:1          }        },        //其余和new Vue()就完全一样了        //问: 模板中共需要几个函数        //本例中共需要minus和add两个函数        methods:{          minus(){ if(this.n>1){this.n--} },          add(){ this.n++ }        }      })    </script>  </head>  <body>    <div id="app">      <ul>        <li><my-counter></my-counter></li>        <li><my-counter></my-counter></li>        <li><my-counter></my-counter></li>      </ul>    </div>    <script>      new Vue({        el:"#app"      })    </script>  </body>  </html>  运行效果： |

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！

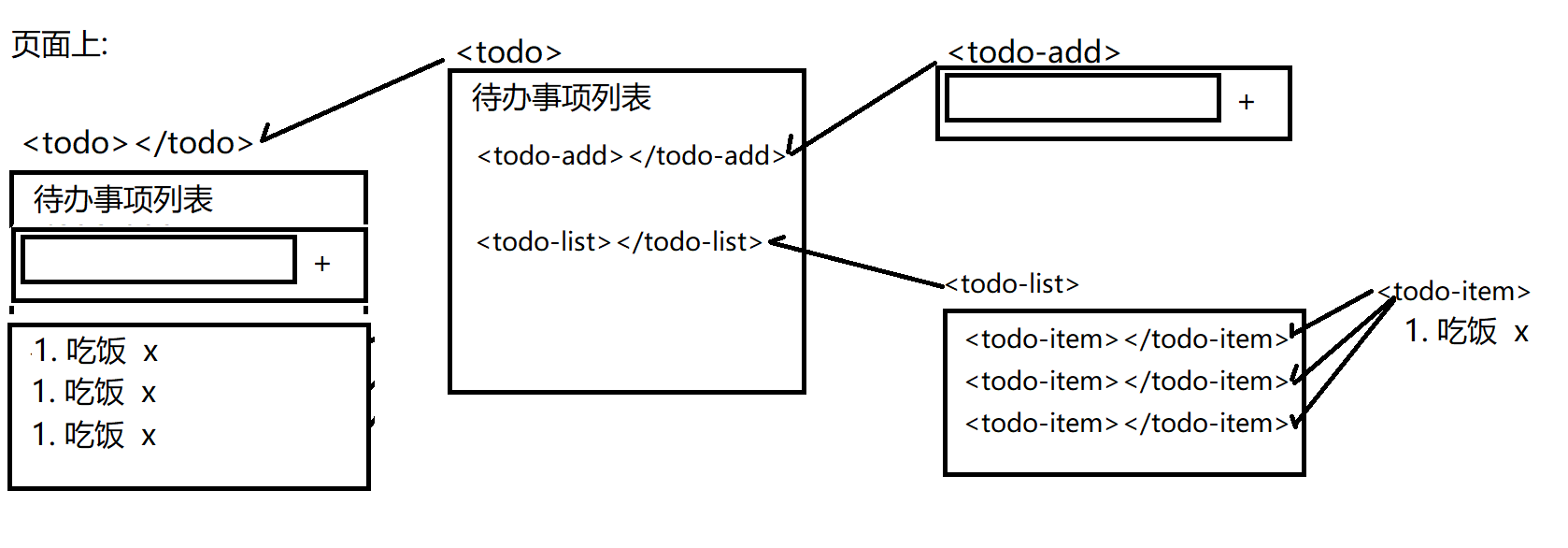
3. 何时: 今后几乎所有项目都采用组件化开发！

4. 如何:

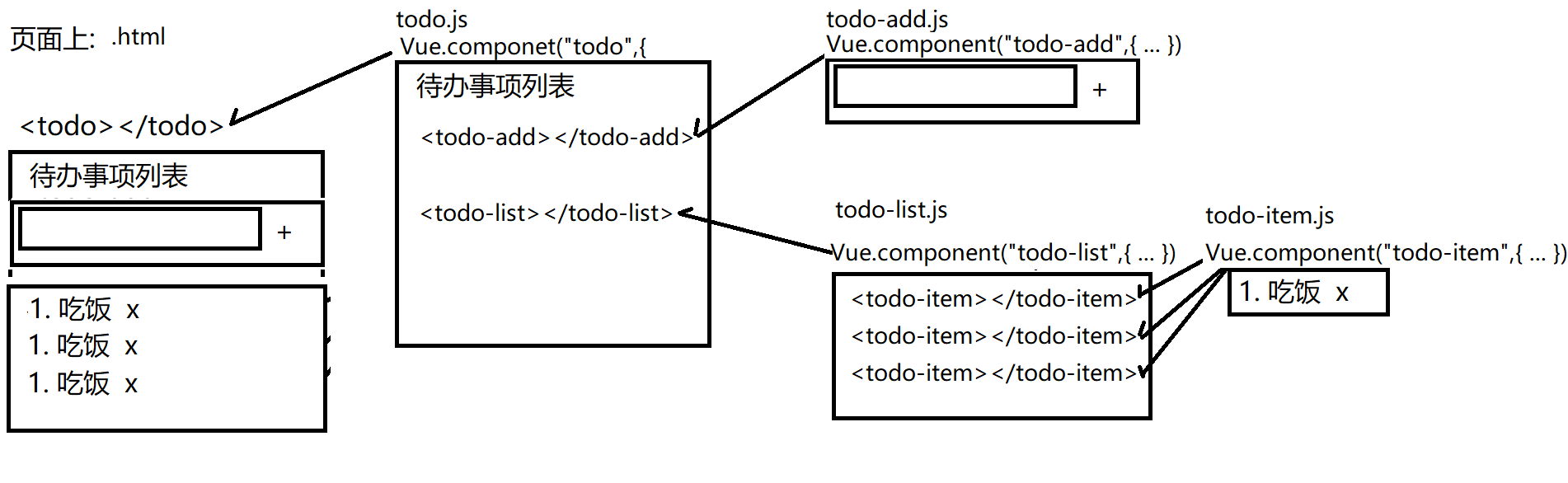
(1). 拿到页面图片，划分组件结构，2个依据

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, ... ... }

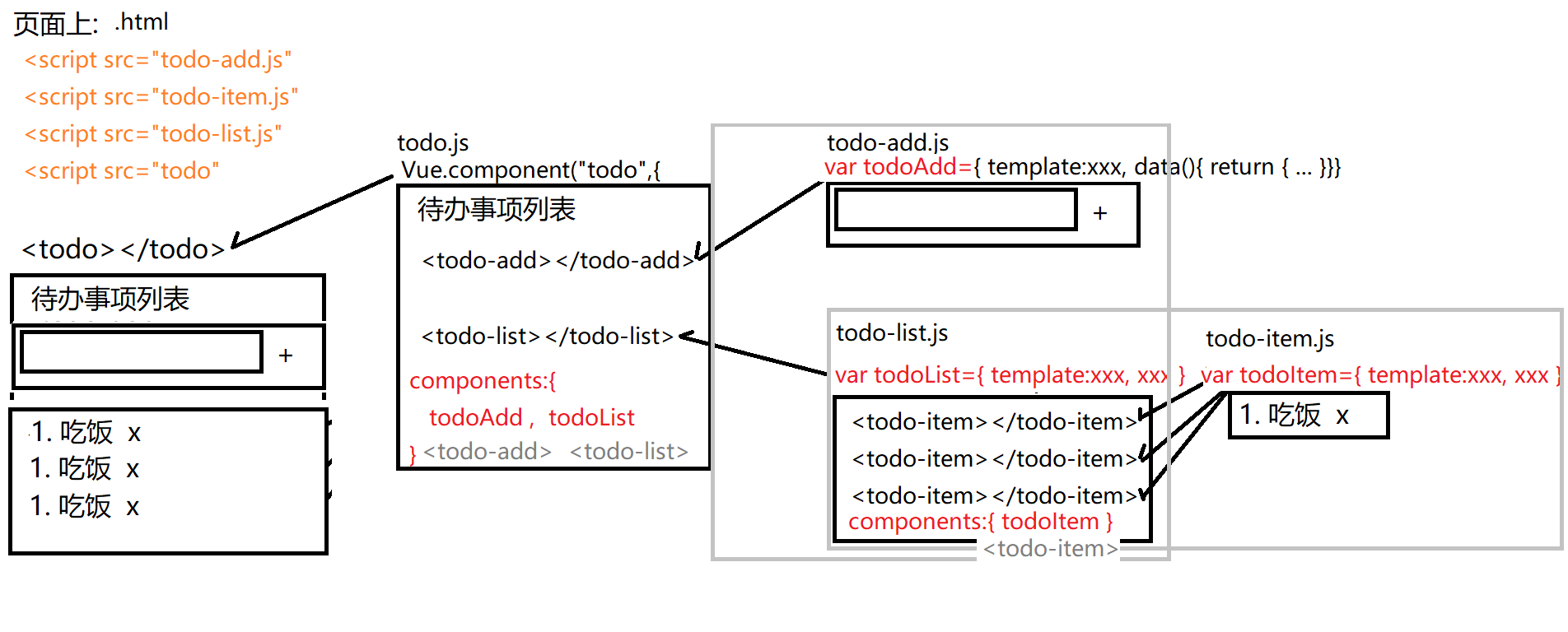
}

(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！



6. 示例: 不考虑数据情况下，将待办事项列表拆分为父子组件

4\_todo-add.js

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

4\_todo-item.js

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

4\_todo-list.js

|  |
| --- |
| var todoList={    template:`<ul>      <todo-item></todo-item>      <todo-item></todo-item>      <todo-item></todo-item>    </ul>`,    components:{ todoItem }  } |

4\_todo.js

|  |
| --- |
| Vue.component("todo",{    template:`<div>      <h1>待办事项列表</h1>      <todo-add></todo-add>      <todo-list></todo-list>    </div>`,    components:{ todoAdd, todoList }  }) |

4\_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="4\_todo-add.js">      //var todoAdd={ ... }    </script>    <script src="4\_todo-item.js">      //var todoItem={ ... }    </script>    <script src="4\_todo-list.js">      //var todoList={      //  ...      //  components:{ todoItem }      //}    </script>    <script src="4\_todo.js">      //Vue.component("todo",{      //  ...      //  components:{ todoAdd, todoList }      //})    </script>      </head>  <body>    <div id="app">      <todo></todo>    </div>    <script>      new Vue({        el:"#app"      })    </script>  </body>  </html>  运行效果: |

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...`

}

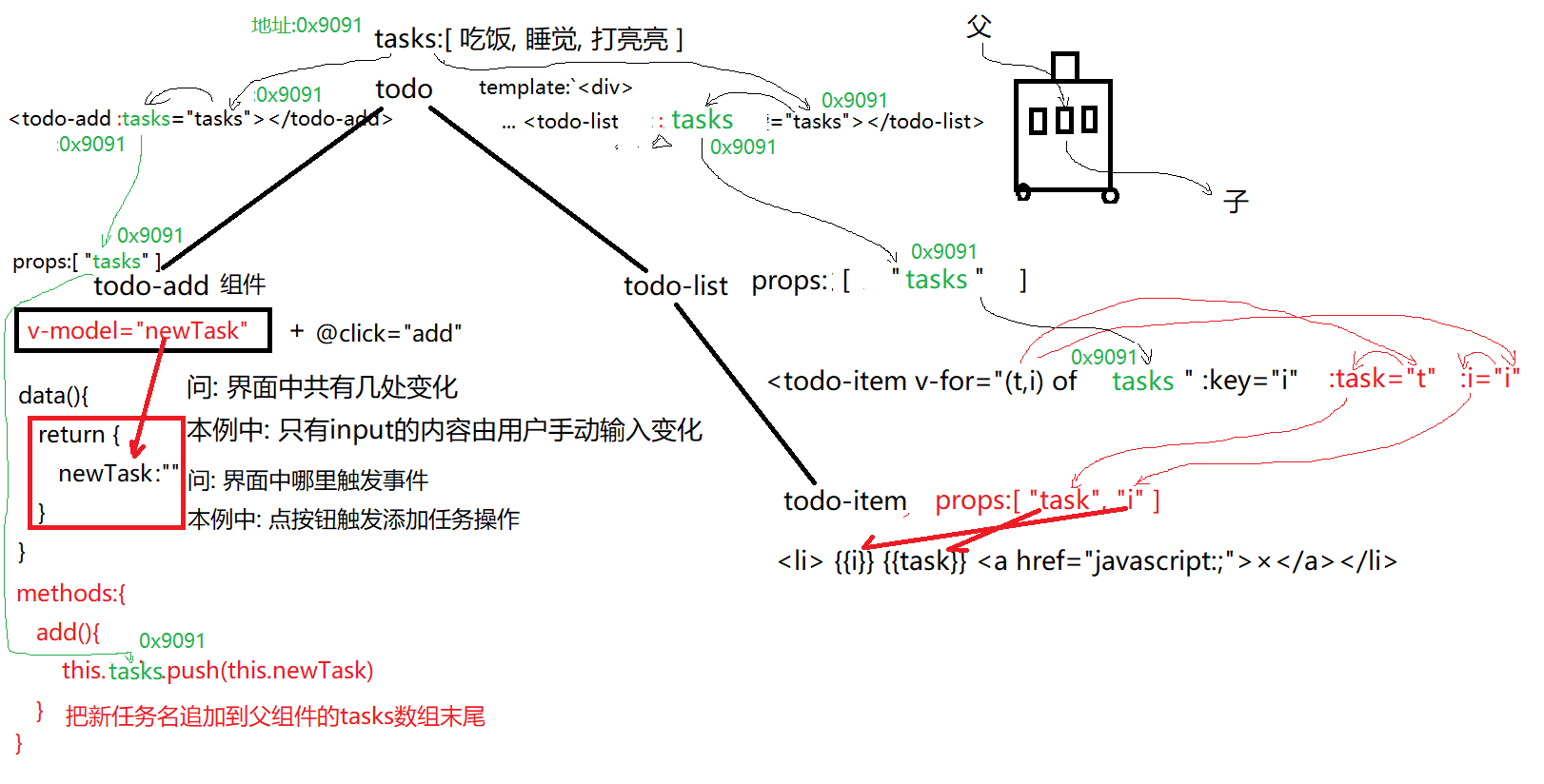
强调:

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！



8. 示例：实现待办事项列表的任务列表和添加功能

4\_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>`,    data(){      return {        tasks:["吃饭","睡觉","打亮亮"]      }    },    components:{ todoAdd, todoList }  }) |

4\_todo-add.js

|  |
| --- |
| var todoAdd={    props:["tasks"],    template:`<div>      <input v-model="newTask"><button @click="add">+</button>    </div>`,    data(){      return {        newTask:""      }    },    methods:{      add(){        this.tasks.push(this.newTask);        //添加完新任务后，清空文本框        this.newTask="";//双向绑定      }    }  } |

4\_todo-list

|  |
| --- |
| var todoList={    props:["tasks"],    template:`<ul>      <todo-item v-for="(t,i) of tasks" :key="i" :task="t" :i="i"></todo-item>    </ul>`,    components:{ todoItem }  } |

4\_todo-item.js

|  |
| --- |
| var todoItem={    props:["task","i"],    template:`<li>      {{i+1}} {{task}} <a href="javascript:;">×</a>    </li>`  } |

4\_todo.html

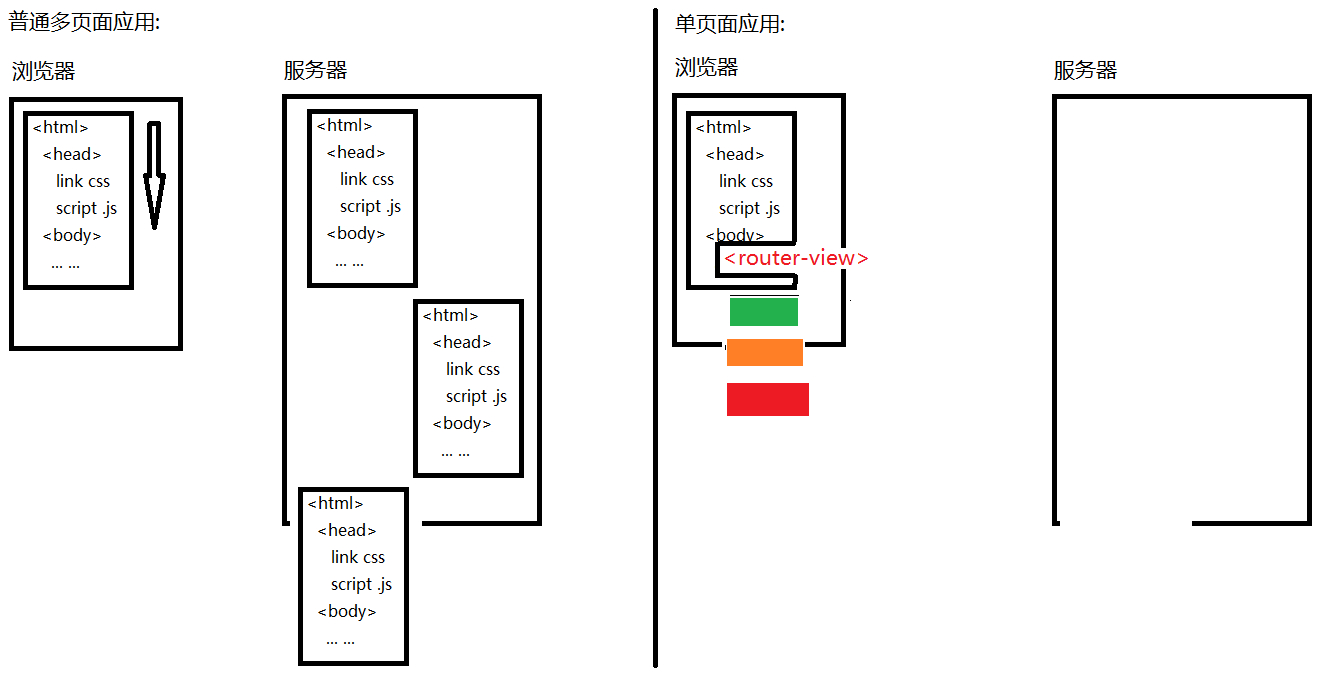
|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <title>Document</title>    <script src="js/vue.js"></script>    <script src="4\_todo-add.js">      //var todoAdd={ ... }    </script>    <script src="4\_todo-item.js">      //var todoItem={ ... }    </script>    <script src="4\_todo-list.js">      //var todoList={      //  ...      //  components:{ todoItem }      //}    </script>    <script src="4\_todo.js">      //Vue.component("todo",{      //  ...      //  components:{ todoAdd, todoList }      //})    </script>  </head>  <body>    <div id="app">      <todo></todo>    </div>    <script>      new Vue({        el:"#app"      })    </script>  </body>  </html>  运行效果: |

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({

routes:[ 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！

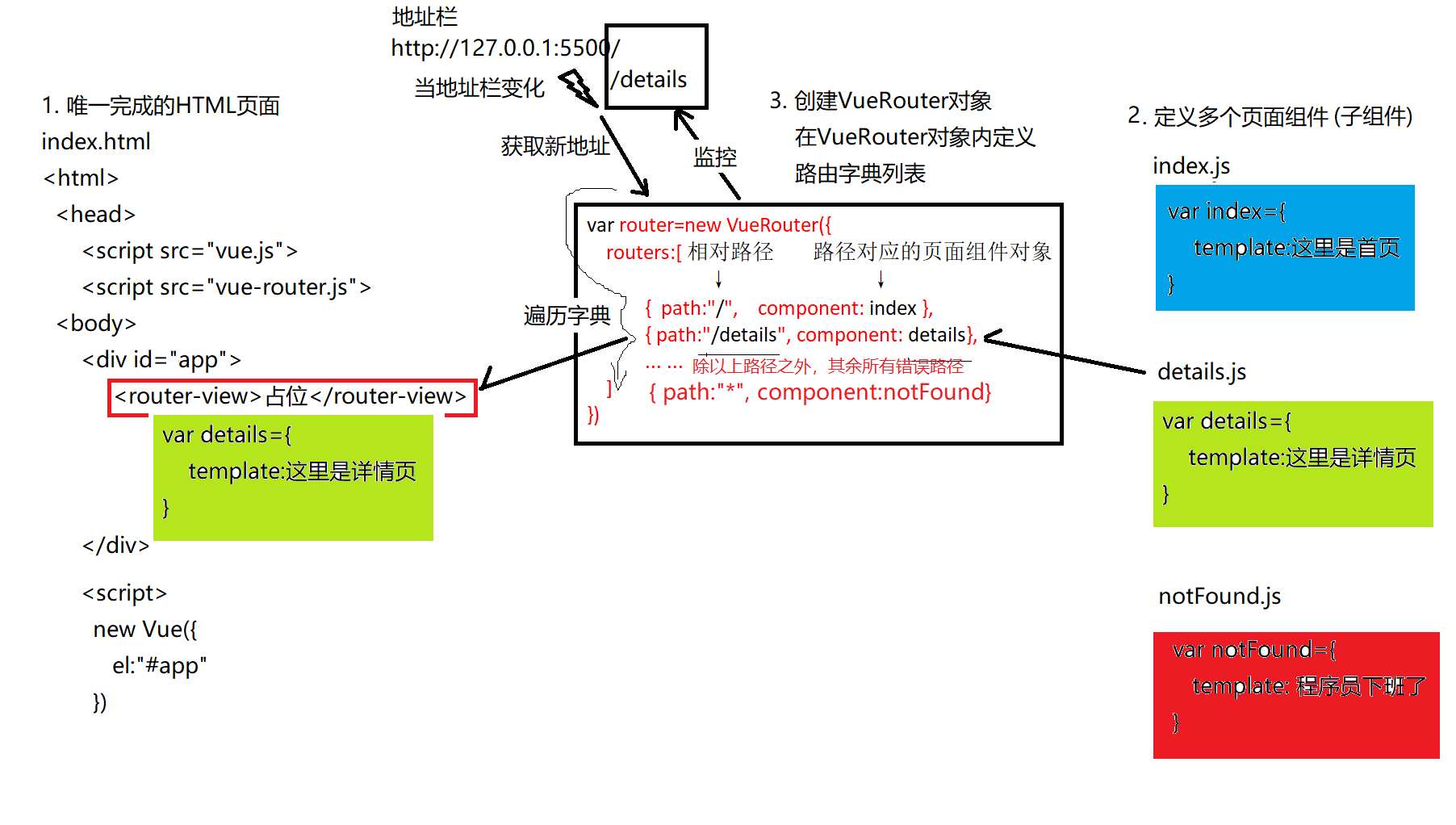
new Vue({

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。

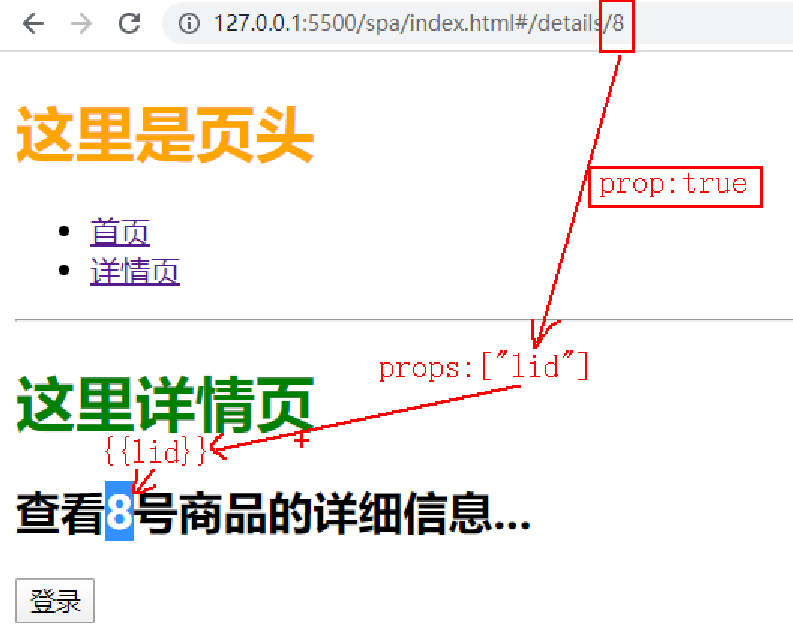
(2). 上一个页面跳转时: 相对路径之后，只拼参数值即可

/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。

总结:

1. MVVM:

(1). 界面(View) 增强版，支持变量、分支、循环等功能

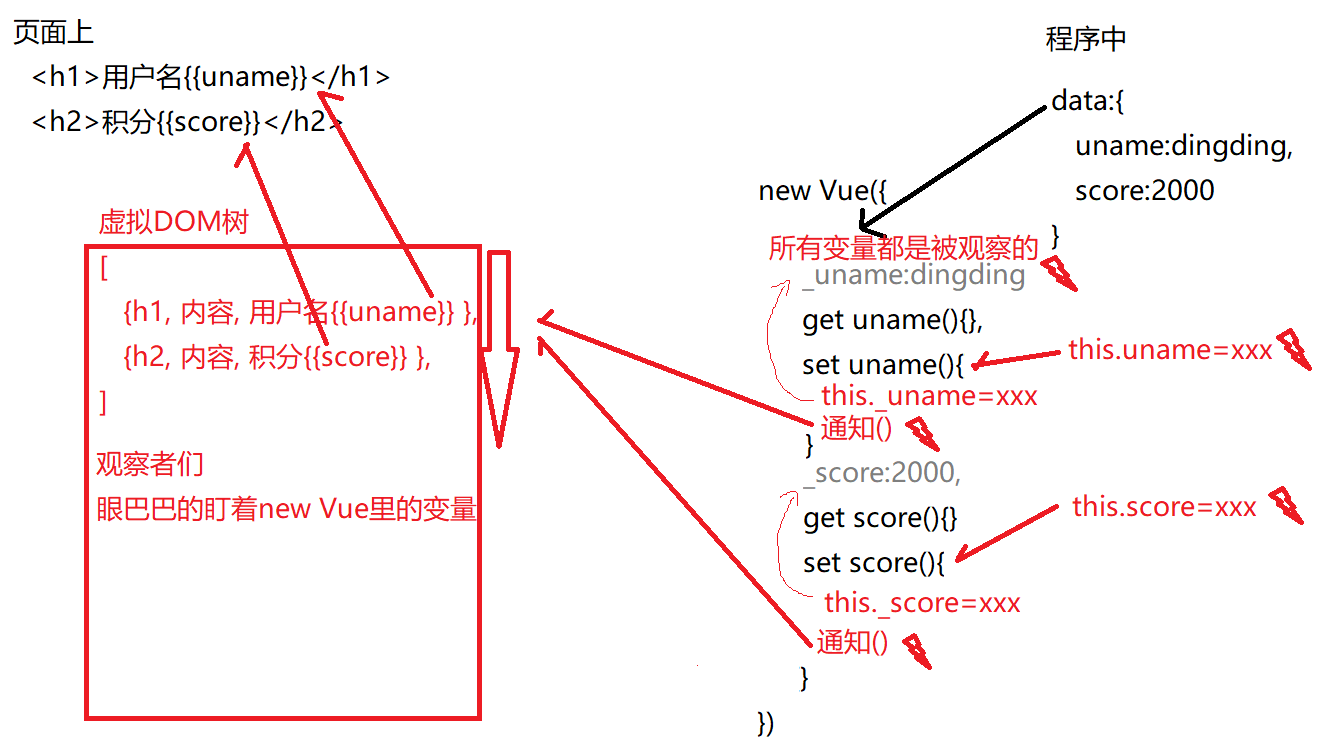
(2). 模型对象(Model) 保存界面所需的一切变量和函数:

(3). 视图模型(ViewModel) 快递员 模型对象中的变量值和函数自动送到界面中所需的位置。并始终保持同步！

2. Vue绑定原理:

(1). 简单: 访问器属性+虚拟DOM树

(2). 完整: 访问器属性+观察者模式+虚拟DOM树



3. 虚拟DOM树:

(1). 什么是: 仅保存可能变化的DOM元素的简化版DOM树  
 (2). 优点: 4个

a. 小: 仅保存可能变化的DOM元素

b. 快: 因为小，所以遍历快

c. 减少重复代码: 自动对受影响的元素执行增删改查操作

d. 更新效率高: 只更新受影响的元素。不受影响的元素，保持不变的！

4. Vue功能: 3步——招数

(1). 编写界面: 要求:

a. 整个界面必须包裹在一个唯一的父元素内，且父元素习惯上id="app"

b. 所有可能发生变化的地方，用{{变量名}}或指令方式标记

c. 所有触发事件的元素，用@事件名="函数名"标记

(2). 先创建new Vue()对象：

new Vue({

el:"#app", //选择器

... ...

})

(3). 为new Vue()添加data和methods模型对象

a. 界面中所需的所有变量都放在data对象中

b. 界面中所需的所有函数都放在methods对象中

c. methods中的函数，想操作data中的变量，必须加this.

5. 绑定语法和指令: ——招数

(1). 如果元素的内容可能随变量自动变化:

<元素>{{变量名}}</元素> 或 <元素 v-text="变量或js表达式"></元素>

(2).只要元素的属性值可能随变量自动变化: :属性名="js表达式"

(3). 控制一个元素显示隐藏: v-show="返回bool值的js表达式"

(4). 控制两个元素二选一显示: <元素1 v-if="返回bool值的js表达式">

<元素2 v-else>

(5). 控制多个元素多选一显示：<元素1 v-if="返回bool值的js表达式">

<元素2 v-else-if="返回bool值的js表达式">

... ...

<元素n v-else>

(6). 反复生成多个相同结构的元素：

<要反复生成的元素 v-for="(value,i) of 数组/对象/整数" :key="i">

(7). 绑定事件: <元素 @事件名="处理函数($event, 实参值)">

(8). 绑定原始HTML代码内容: <元素 v-html="变量"></元素>

(9). 防止用户短暂看到{{}}:

<元素 v-cloak>...</元素> 或 <元素 v-text="变量或js表达式"></元素>

(10). 今后只要一个绑定的值，在首次加载之后不再发生变化:

<元素 v-once>{{变量或js表达式}}</元素>

(11). 防止元素内容中的{{}}被new Vue()编译，保持{{}}原样显示:

<元素 v-pre>{{xxx}}</元素>

(12). 只要想获得用户在表单元素中输入的新值:

<元素 v-model ="变量名">

6. 绑定样式:

(1). 绑定内联样式: 只要仅单独修改一个css属性时，可首选绑定:style

<元素 style="固定不变的css属性" :style="变量">

new Vue({

data:{

变量: {

Css属性:"属性值",

... : ...

}

}

})

(2). 绑定class: 如果批量修改一个元素的多个css属性，首选绑定class

<元素 class="固定不变的class" :class="变量">

new Vue({

data:{

变量: {

Class1: true或false,

... : ...

}

}

})

7. 创建自定义指令: 只要希望下元素加载到页面上时，自动对元素执行一些初始化操作

(1).

Vue.directive("指令名",{

inserted(elem){

对当前带有指令的elem元素执行原生的初始化DOM操作

}

})

(2). 使用: <元素 v-指令名>

8. 计算属性: 只要想要的值没有直接给出，需要根据其他属性值经过复杂的计算才能获得

(1). 定义:

new Vue({

el:"#app",

data:{ },

methods:{ },

computed:{

计算属性名(){

根据其他属性值经过复杂计算得到一个结果

return 计算结果

}

}

})

(2). 使用: <元素>{{计算属性名}}</元素>

(3). 如何选择:

a. 如果更关心执行结果的数值时，首选computed

b. 如果只关心执行过程，不关心执行结果，首选methods

9. 过滤器: 只要变量的原始值不能直接给人看，需要经过加工才能显示给人看时

(1). 定义:

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

return 根据每次自动传入的变量原始值oldVal，加工后生成新值返回

})

(2). 使用:

<元素>{{变量 | 过滤器1(实参值) | 过滤器2 | ...}}</元素>

作业:

1. 复习今日问题清单(VUE->day03~day04)，预习明日问题清单(VUE->day04~day05)

2. 看小程序视频完成待办事项列表删除功能

小程序->在线->VUE->day03 6. 组件化开发 todo实现封装组件...

3. 看小程序视频预习使用vue/cli工具创建学子商城项目脚手架

小程序->在线->VUE->day04 2. 使用vue/cli工具创建学子商城项目脚手架vue create xzvue

补: 如果npm，安装出错，可尝试以下手段：

　　　　运行npm cache clean -f，清空缓存再试，

　　　　不行，就重新安装node，

　　　　还不行，就切换网络，试试用手机热点下载和安装。网络有问题也可能安装不成功，

比如，我在家用的是移动赠送的宽带，npm安装就出错，我改用手机热点联网，npm安装就成功。

补: 苹果本解决npm安装错误，还可尝试

#cd到项目目录 #然后依次执行下面的命令 rm -rf node\_modules rm package-lock.json npm cache clear --force npm install

以上解决方法都不行，只能重装操作系统。

4. 看小程序视频学习Vue脚手架文件夹结构

小程序->在线->VUE->day04 3. Vue脚手架文件夹结构

5. （学有余力）看小程序视频学习三种组件间传值的办法:

小程序->在线->VUE->day03 7 组件间传参 todo 父给子 子给父 兄弟间 ref $refs