小程序: WEB问题速查

1. 问题清单: 历届师兄师姐在今天课程中集中问过的高频的重点的问题！

那些自己发现不了问题，又迫切想知道其它同学都问了什么问题，的同学

小程序->首页->VUE->day03、day04

当老师在回答其它同学的问题时，你不要闲着！你可以打开问题清单浏览别人曾经问过的经典问题！

2. 高清无码小视频: 小程序->在线->VUE->day03、day04

2部分: 1. 课上重点知识和案例的回放！2. 重点高频笔试面试题讲解扩展视频

万一家里或中心网络断了！暂时看不了直播了！

因为某些事儿请假了，错过了当天的直播！

课下复习时，觉得TMOOC视频太大，浏览不方便或TMOOC不能看了！

——打开小程序中"在线"，就可下载观看当前所讲内容和当天所讲全部内容的回放。

基础比较好的，确实学有余力的同学，想拿高薪的同学，在我答疑时，不要闲着:

要看今天扩展视频:

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

3. 单词列表: 小程序->单词->VUE->day03、day04

第三阶段上课约定:

1. 上课千万不要跟着记笔记和写代码！

2. 凡是拼写错误，自己懒得找！我可以帮你找，但是找到后，错误的单词你要手抄20遍，并拍照！

出错: 先开f12，console，看错误提示！

3. 每个带注释的案例至少写三遍:

(1). 注释+代码抄一遍

(2). 保留注释，删除代码，自己试着把代码翻译回来！

(3). 删除注释和代码，自己试着用自己的话把注释写回来！

一个例子只有自己可以把人话的注释写回来！才算自己会！

问题: 一听就会一做就废！原因: 你自己无法用人话精确描述自己想干什么！

如何提问: （vip待遇的剩余时间2+1天）

1. 上课时，案例中，笔记中那句话哪行代码看不懂，立刻问！

2. 小程序中问题清单里哪个问题和回答看不懂！立刻问！

3. 个人项目中哪个功能不会做！立刻问！

4. 刷笔试题时，遇到不会的笔试题！立刻问！

谁问的问题越多！谁的培训费交的越值！

正课:

1. 计算属性

2. 过滤器

3. axios

4. 生命周期

5. 组件

6. 组件化开发

一. 计算属性:

对应小程序视频: 小程序->在线->VUE->day02 10 计算属性

1. 问题: 有时页面上要显示的值，后端并没有直接给出，但是却可以根据给出的其它变量值经过复杂的计算过程动态计算获得

2. 解决: 2种:

(1). **不好的办法**: 使用自定义函数: 在methods中定义一个自定义函数，能够根据data中其它变量的值，计算出想要的新值并以返回值方式返回！

a. methods:{

自定义函数(){

计算过程;

return 返回值

}

}

b. **<元素>{{自定义函数()}}</元素>**

c. 问题: 如果页面上需要反复多次显示计算结果，如果调用自定义函数，就会多次反复计算——没必要！效率低

(2). **好的办法**: 使用计算属性: 自己不保存值，每次使用属性时，都根据其他变量的值动态计算获得自己的属性值！

a. 定义计算属性:

new Vue({

el:"#app",

data:{...},

methods:{ ... },

**computed:{**

**属性名(){ //计算属性, 本质/形式却是一个函数！**

**计算过程;**

**return 返回值**

**}**

**}**

})

强调: 计算属性虽然叫属性，但是本质却是一个可反复执行的函数

b. 在界面中使用计算属性；

**<元素>{{计算属性名}}</元素>**

强调: **在绑定语法中使用计算属性时，一定不要加()**

c. 优点: Vue框架会缓存计算属性的结果。

1). 只要计算属性依赖的其它变量值不变，则无论反复使用多少次该计算属性，也只在首次使用时计算一次。之后每次使用其实都是从缓存中获取旧值——避免重复计算——效率高！

2). 除非计算属性依赖的其它变量值发生了变化，vue框架才会重新计算一次计算属性的值，并再次缓存起来，反复使用！

3. 总结:

(1). 今后如果更**偏重于**使用一个**计算结果值**时，首选**计算属性**

比如: 根据购物车中商品列表计算购物车的总价

(2). 今后如果仅**偏重于执行**一项**任务**，而**不关心返回值**时，首选**函数**:

比如: 添加/删除/修改等服务器端相关的操作

4. 示例: 使用计算属性，计算购物车总价:

1\_computed.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>  </head>  <body>    <div id="app">      <h3>总价: ¥{{total.toFixed(2)}}</h3>      <ul>        <!--用v-for遍历cart变量中保存的购物车中每个商品对象        强调: **循环变量p，获得的是cart数组中某一个商品对象，所以p.lid或p.lname才能访问每个商品的id和名称-->**        <li v-for="(p,i) of cart" :key="i">{{p.lid}} | {{p.lname}} | ¥{{p.price.toFixed(2)}} | {{p.count}} | 小计: ¥{{(p.price\*p.count).toFixed(2)}}</li>      </ul>      <!--调用methods中自定义的total函数，必须加()-->      <h3>总价: ¥{{total.toFixed(2)}}</h3>    </div>    <script>      new Vue({        el:"#app",        data:{          cart:**[**            {lid:1, lname:"华为",price:5588,count:2},            {lid:2, lname:"小米",price:3588,count:3},            {lid:3, lname:"苹果",price:8588,count:1},  **]**        },        methods:{//methods中其实可以保存任何函数，而不仅仅局限于事件处理函数          },        computed:{          //定义了一个计算属性，属性名为total          total(){//自定义函数            console.log(`计算了一次总价...`)            var total=0;            //遍历购物车中每个商品对象            for(var p of this.cart){              //每遍历一个对象，就将商品的单价\*数量求出这个商品的小计，累加到变量total上              total+=p.price\*p.count;            }            return total;          }        }      })    </script>  </body>  </html>  运行结果: |

二. 过滤器:

对应小程序视频: 小程序->在线->VUE->day03 3. 过滤器

1. 问题: 有些服务器端保存的数据，不能直接给人看！需要加工后，人才能看懂！

比如: 性别，服务器端保存的是1和0， 人能看懂的是男和女

日期和时间，服务器端保存的是毫秒数，人肯定是看不懂！必须转为年月日人才能看懂

2. 解决: 过滤器: 专门将变量的原始值加工后再显示出来的特殊的函数

3. 何时: 今后只要一个变量的原始值必须经过加工后才能给人看时，就用过滤器

4. 如何: 2步:

(1). 定义过滤器:

进

↓

**Vue.filter("过滤器名",function(oldVal){**

//加工

出← **return 加工后的新值**

**})**

说明: Vue.filter()是为整个vue大家庭添加一个过滤器的意思。我们创建一个过滤器，其实可以在项目中任意位置使用！

(2). 使用过滤器加工变量的原始值:

**<元素>{{变量 | 过滤器}}</元素>**

连接 显示出加工后的新值

5. 示例: 使用过滤器翻译性别的1和0为男和女

2\_filter.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>  </head>  <body>    <div id="app">      <h3>性别1:{{sex1**| sexFilter}**}</h3>      <h3>性别2:{{sex2 | sexFilter}}</h3>    </div>    <script>      //需要将性别的1和0翻译为男和女      Vue.filter("sexFilter",function(**oldVal**){        return oldVal==1?"男":"女"      });    </script>    <script>      new Vue({        el:"#app",        data:{          sex1:1,          sex2:0        }      })    </script>  </body>  </html>  运行结果:  **性别1:男**  **性别2:女** |

6. 过滤器也可以传入**自定义参数值**:

(1). 问题: 有时过滤器最终过滤出什么结果，希望也可以用参数值来控制

(2). 如何: 2步

预留

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

**return 加工后的新值**

**})**

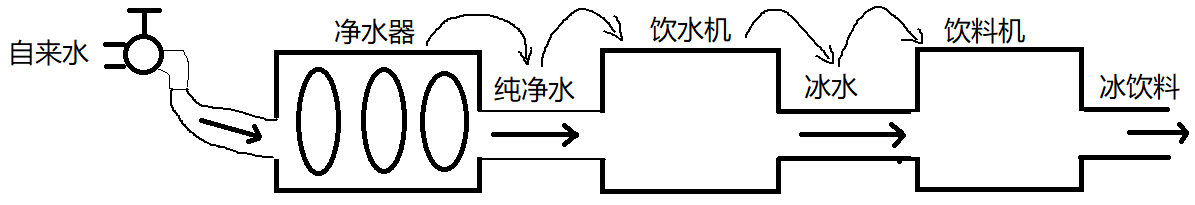
b. **<元素>{{变量 | 过滤器(自定义实参值)}}</元素>**

(3). 示例: 使用过滤器的参数控制转为中文性别或英文性别

3\_filter2.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>  </head>  <body>    <div id="app">      <!--有时想过滤出中文的男和女          有时想过滤出英文的Male和Female-->      <h3>性别1:{{sex1 | sexFilter}}</h3>      <h3>性别2:{{sex2 | sexFilter}}</h3>      <!--想转英文: -->      <h3>性别1:{{sex1 | sexFilter(**"en"**)}}</h3>      <h3>性别2:{{sex2 | sexFilter("en")}}</h3>    </div>    <script>      //定义一个自定义！自定义！自定义！形参变量lang，表示想转成的语言种类是哪种！      //如果想转成英文，就传入“en”——我们自己的规定！！！      //否则传入其余任何值都默认转为中文      Vue.filter("sexFilter",function(oldVal,**lang**){  **if(lang=="en")**{          return oldVal==1?"Male":"Female"        }else{          return oldVal==1?"男":"女"        }      });    </script>    <script>      new Vue({        el:"#app",        data:{          sex1:1,          sex2:0        }      })    </script>  </body>  </html>  运行结果: 性别1:男性别2:女性别1:Male性别2:Female |

7. 多个过滤器连用:



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

(2). 强调: **只有过滤器1有机会难道变量的原始值**。之后的其它过滤器拿到的都不是元素值，而是前一个相邻的过滤器返回的中间产物！

(3). 示例: 显示性别名称之后，追加显示性别图标

4\_filter3.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>  </head>  <body>    <div id="app">      <!--想无论什么时候显示性别，都在性别结尾加性别图标-->      <h3>性别1:{{**sex1 | sexFilter | sexIcon**}}</h3>      <h3>性别2:{{sex2 | sexFilter | sexIcon}}</h3>      <!--想转英文: -->      <h3>性别1:{{sex1 | sexFilter("en") | sexIcon}}</h3>      <h3>性别2:{{sex2 | sexFilter("en") | sexIcon}}</h3>    </div>    <script>      //定义一个自定义！自定义！自定义！形参变量lang，表示想转成的语言种类是哪种！      //如果想转成英文，就传入“en”——我们自己的规定！！！      //否则传入其余任何值都默认转为中文      Vue.filter("sexFilter",function(oldVal, lang){        if(lang=="en"){          return oldVal==1?"Male":"Female"        }else{          return oldVal==1?"男":"女"        }      });      //再加一个专门给性别结尾追加性别图标的过滤器  **Vue.filter("sexIcon",function(oldVal)**{        //oldVal共有几种情况?4种: 男，女，Male，Female        if(oldVal=="男"||oldVal=="Male"){          return **oldVal+"♂"**        }else{          return oldVal+"♀"        }      })    </script>    <script>      new Vue({        el:"#app",        data:{          sex1:1,          sex2:0        }      })    </script>  </body>  </html>  运行结果: 性别1:男♂性别2:女♀性别1:Male♂性别2:Female♀ |

三. axios

对应小程序视频: 小程序->在线->VUE->day03 4. axios

1. 今后所有的函数库或框架都提供了专门的发送ajax请求的函数。从此发送任何ajax请求，只需要一句话即可！

2. **vue中发送ajax请求，都用axios函数库**: 专门发送ajax请求的**基于Promise**的函数库。

3. 问题:

(1). axios不是vue官方开发的，但是确实vue官方强烈推荐使用的，所以需要先下载才能使用

(2). **axios不是vue专用的，即使没有vue，在其它平台比如nodejs服务器端也可以发送http请求**

4. 如何:

(0). **配置**axios对象的服务器端接口地址的**公共基础地址**部分

**axios.defaults.baseURL="服务器端基础地址"**

(1). **axios.get(**

**"服务器端接口的相对路径",**

**{**

**params: { 参数名: 参数值 }**

**}**

**).then(result=>{ /**/会在前一步请求结束后自动执行

~~//坑: result不直接是响应结果~~

**//result.data 才是真正的响应结果**

**})**

(2). **axios.post(**

**"服务器端接口的相对路径",**

**"变量名=变量值&变量名=变量值&..."**

**) .then(result=>{** //会在前一步请求结束后自动执行

~~//坑: result不直接是响应结果~~

//result.data 才是真正的响应结果

**})**

6. 问题: axios的.then()中的function(){}是回调函数，其中的this默认指window，而我们如果向操作data中的变量时，希望this->当前new Vue()对象！

解决: 今后在**new Vue()内使用axios时**，**.then()中必须使用箭头函数！**让回调函数内外this相通！都指向new Vue()对象

7. 示例: 使用axios向东哥新浪云服务器发送请求

5\_axios.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/axios.min.js"></script>  </head>  <body>    <script>      //0. axios可提前配置共用的服务器端基础路径      axios.defaults.baseURL="http://xzserver.applinzi.com";      //1. 去东哥的新浪云服务器请求学子商城首页6个商品      axios.get(        "/index"      ).then(function(result){        console.log(result.data);      })      //2. 去东哥的新浪云服务器请求学子商场5号商品的详细信息      axios.get(        "/details",        {          params:{            lid:5          }        }      ).then(function(result){        console.log(result.data);      })      //3. 去东哥的新浪云服务器请求登录      axios.post(        "/users/signin",        "uname=dingding&upwd=123456"      ).then(function(result){        console.log(result.data);      })    </script>  </body>  </html>  运行结果:  *(6) [{…}, {…}, {…}, {…}, {…}, {…}]*  0: {pid: 1, title: "Apple MacBook Air系列", details: "酷睿双核i5处理器|256GB SSD|8GB内存|英特尔HD显卡620含共享显卡内存", pic: "img/index/study\_computer\_img1.png", price: 5988, …}  1: {pid: 2, title: "小米Air 金属超轻薄", details: "酷睿双核i5处理器|512GB SSD|2GB内存|英特尔HD独立显卡", pic: "img/index/study\_computer\_img2.png", price: 3488, …}  2: {pid: 3, title: "联想E480C 轻薄系列", details: "酷睿双核i7处理器|256GB SSD|4GB内存|英特尔HD显卡680M", pic: "img/index/study\_computer\_img3.png", price: 5399, …}  3: {pid: 4, title: "华硕RX310 金属超极本", details: "酷睿双核i5处理器|512GB SSD|4GB内存|英特尔HD游戏级显卡", pic: "img/index/study\_computer\_img4.png", price: 4966, …}  4: {pid: 5, title: "联想小新700 电竞版游戏本", details: "酷睿双核i7处理器|1TGB SSD|8GB内存|英特尔HD显卡620含共享显卡内存", pic: "img/index/study\_computer\_img5.png", price: 6299, …}  5: {pid: 6, title: "戴尔灵越燃7000 轻薄窄边", details: "酷睿双核i5处理器|512GB SSD|2GB内存|英特尔HD显卡", pic: "img/index/study\_computer\_img3.png", price: 5199, …}  length: 6  \_\_proto\_\_: Array(0)  *{ok: 1, uname: "dingding"}*  *{product: {…}, specs: Array(4), pics: Array(7)}*  pics: (7) [{…}, {…}, {…}, {…}, {…}, {…}, {…}]  product: {lid: 5, family\_id: 2, title: "小米(MI)Air 13.3英寸全金属超轻薄笔记本(i5-6200U 8G 256G PCIE固态 940MX独显 FHD WIN10)银", subtitle: "【i5 独立显卡】全高清窄边框 8G内存 256G固态硬盘 支持SSD硬盘扩容 薄至14.8mm 轻至1.28kg！AKG扬声器！", price: 4999, …}  specs: (4) [{…}, {…}, {…}, {…}]  \_\_proto\_\_: Object |

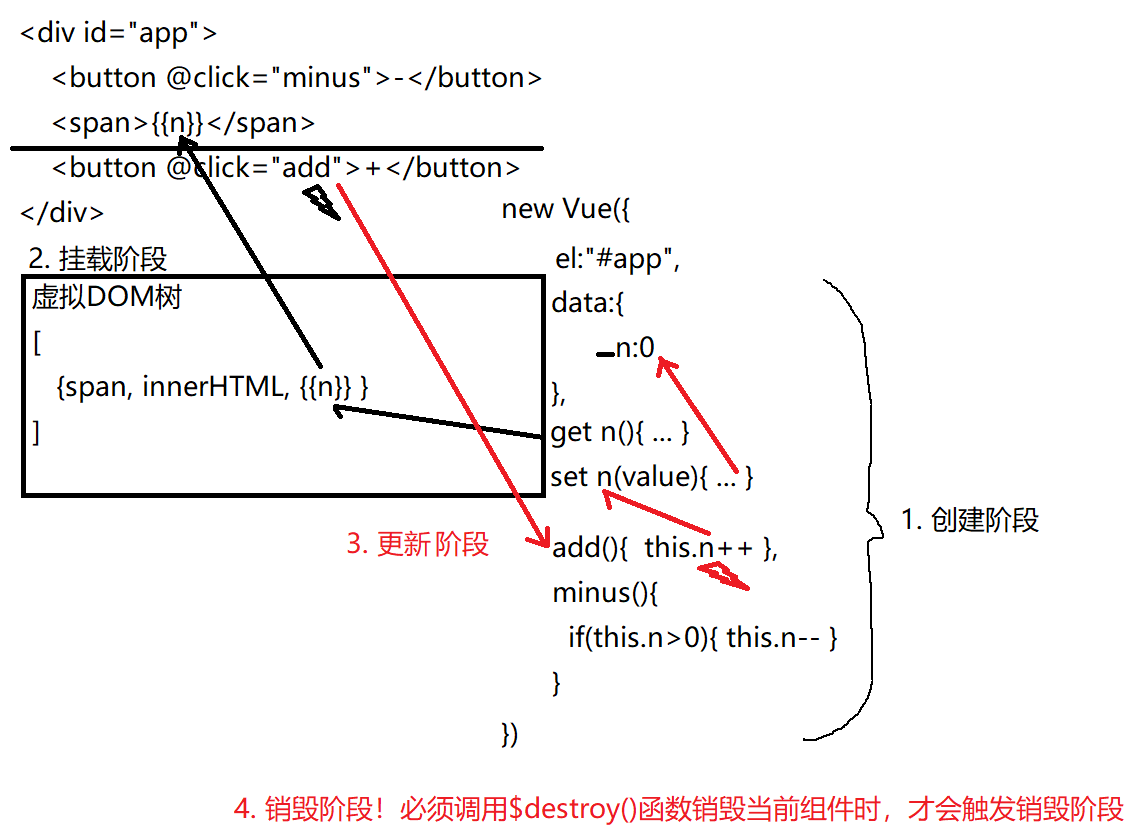
**四. 生命周期:**

对应小程序视频: 小程序->在线->VUE->day03 5. 生命周期

1. 问题: DOM和jq中，如果希望在页面加载完成后自动执行一段代码，这段代码应该放在domContentLoaded或load两个事件处理函数中！但是new Vue()没有domContentLoaded和load事件处理函数，那么我们希望在页面加载时自动执行一项操作(比如自动发送ajax请求)，应该写在哪儿呢？！

2. 答: 其实new Vue()和DOM中的网页一样，也有**加载过程**。在加载过程的不同阶段，也可以自动触发一些函数的执行——vue的生命周期

**3. vue的生命周期包括: 4个阶段！(2个必经阶段, 2个在特定情况下才触发)**



**(1). 创建(create):**

a. 创建new Vue()对象，以及data对象，以及data对象中的变量——可以操作data中的变量

b. 暂时没有扫描DOM树，也就没有虚拟DOM树——但是暂时无法操作DOM树上的元素

**(2). 挂载(mount):**

a. 已经有data对象——既可操作data中的变量

b. 扫描真实DOM树，生成虚拟DOM树，并**首次加载**数据到页面元素——又可操作DOM树上的元素

===================**以上必经阶段**===============================

**(3). 更新(update):** 只有new VUe中的程序修改了data中的变量时，并影响了页面中的内容，才会触发更新阶段！

**(4). 销毁(destroy):** 只有手动调用**$destroy()**函数，销毁当前组件时，才进入销毁阶段。

5. 问题: 生命周期阶段只是个名称，不是函数，也无法写代码

解决: 其实当new Vue()进入每个阶段前后，都会自动触发一组固定的函数！——钩子函数。如果我们将希望在某个阶段执行的操作代码，放入特定的**钩子函数**中，就可在new Vue()加载过程中，经历到某个阶段时，自动调用执行！

6. new Vue共包含8个生命周期钩子函数:

在进入创建阶段前会自动触发: **beforeCreate(){ ... }**

(1). 创建(create)

在离开创建阶段后会自动触发: **created(){ ... }**

在进入挂载阶段前会自动触发: **beforeMount(){ ... }**

(2). 挂载(mount)

在离开挂载阶段后会自动触发: **mounted(){ ... }**

在进入更新阶段前会自动触发: **beforeUpdate(){ ... }**

(3). 更新(update)

在离开更新阶段后会自动触发: **updated(){ ... }**

在进入销毁阶段前会自动触发: **beforeDestroy(){ ... }**

(4). 销毁(destroy)

在离开销毁阶段后会自动触发: **destroyed(){ ... }**

7. 用的绝对多的！只有一个！**mounted(){ ... }**

因为此时，new Vue()中既有data对象，又有虚拟DOM树，即可操作data中的变量，又可操作DOM树上的元素。

比如: 今后，希望在页面加载时就自动发送的ajax请求代码都应该写在mounted()中！

8. 示例: 在vue生命周期钩子函数中发送axios请求，并修改data中的变量

6\_lifecycle.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="js/axios.min.js"></script>  </head>  <body>    <div id="app">      <ul>        <li v-for="(p,i) of products" :key="i">          {{p.pid}} | {{p.title}} | {{p.details}}        </li>      </ul>    </div>    <script>      //今后，凡是和vue有关的代码，都必须写在new VUe()内部！      //写在new Vue()外部的代码是不受监控的      //强调: 以下8个函数绝对不要都写出来！而是用到哪个，就只写哪个！      var vm=new Vue({        el:"#app",        data:{  **products:[]**//定义商品列表变量，准备接受服务器端返回的商品列表        },        beforeCreate(){          console.log(`创建前...`)        },        created(){          console.log(`创建后...`)        },        beforeMount(){          console.log(`挂载前...`)        },        mounted(){          //this->new Vue()          //这个时候既有data，又有虚拟DOM树了，什么都不缺了！          console.log(`挂载后...`);          axios.get(            "http://xzserver.applinzi.com/index"          ).then(result=>{            console.log(result.data);            //希望将结果放到data中的products变量上保存  **this.products=result.data;**            //应该能触发更新阶段！          })        },        beforeUpdate(){          console.log(`更新前...`)        },        updated(){          console.log(`更新后...`)        },        beforeDestroy(){          console.log(`销毁前...`)        },        destroyed(){          console.log(`销毁后...`)        }      })  //手工在控制台中输入: vm.$destroy()可看到销毁阶段    </script>  </body>  </html>  运行结果: |

五. **组件**

对应小程序视频: 小程序->在线->VUE->day03 6.1 创建单个组件

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

2. 为什么: 重用！

3. 何时: 今后只要页面中一个功能，会被反复使用，都要封装为组件。

4. 如何创建一个组件:

//向vue大家庭中添加一个自定义组件

**Vue.component("组件名",{**

//每个组件对象，其实都是一个缩微版的new Vue()

//所以几乎new Vue()中有什么，组件对象中，就有什么

//但是稍微有一点儿差别: 2个

~~el:"#app",~~

**template:`组件的HTML内容模板`,** //将来在页面中使用了几次组件，这个模板就会在页面中反复创建几次一模一样的元素！

~~data:{ ... },~~

**data(){** //将来在页面中使用了几次组件，vue就会自动反复调用几次data函数，返回几个模型对象的副本。——保证每个组件副本都有专属的数据模型对象，互不干扰！

**return {**//new Object()

**每个组件需要的变量和初始值**

**}**

**},**

=====以下都是一样的=====

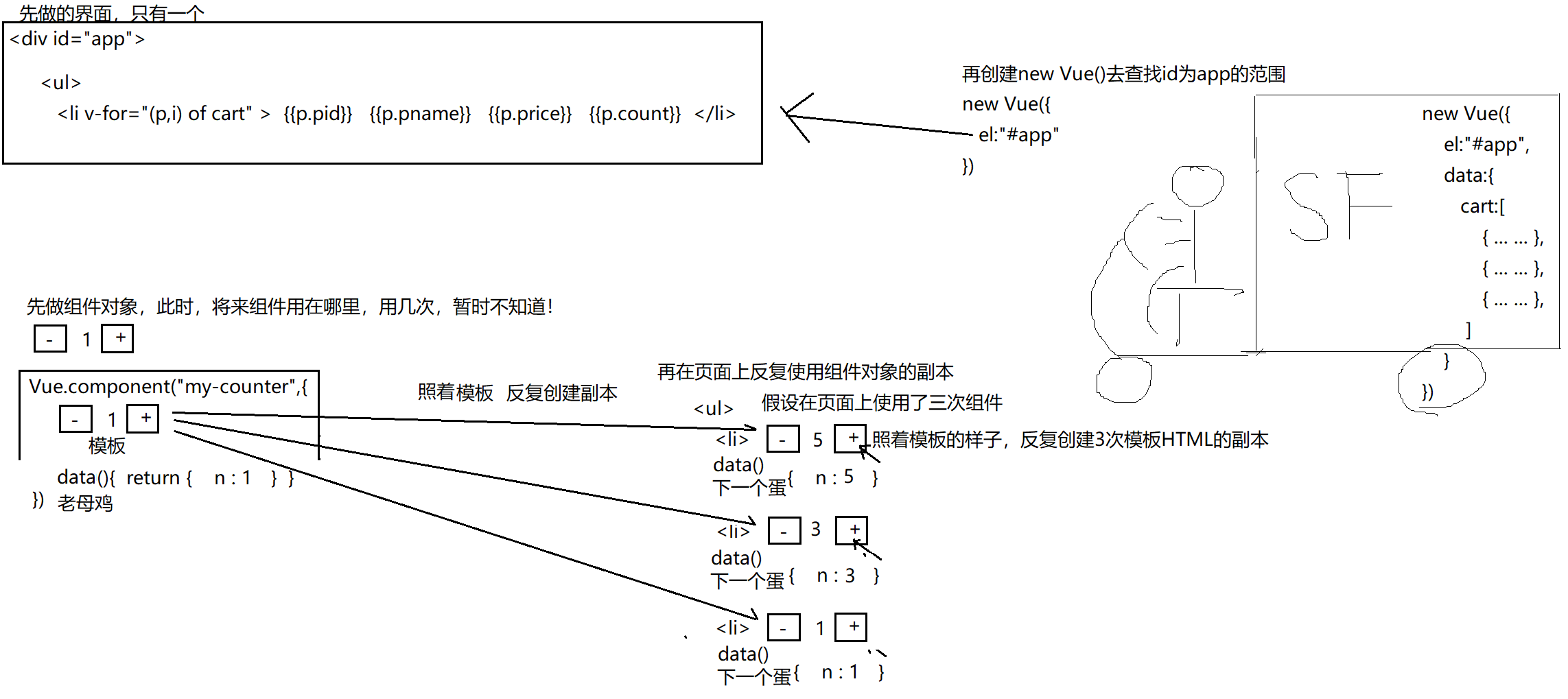
**methods:{ ... },**

**computed:{ ... },**

**mounted(){ ... },**

**... ...**

**})**



5. 如何使用组件:

(1). vue中的组件在使用上，其实就是一个可重用的**自定义HTML标签**而已！组件名叫什么，标签名就叫什么

(2). 强调: 因为组件名就是将来的标签名，又因为HTML语言不区分大小写！所以如果组件名使用驼峰命名，到了HTML中是区分不出来的！所以，**组件名**中如果包含多个英文单词，应该**使用-分隔**，而不应该用驼峰命名！

比如: 在HTML中<myCounter>和<MYCOUNTER>和<mycounter>其实是同一个标签！

6. 原理:

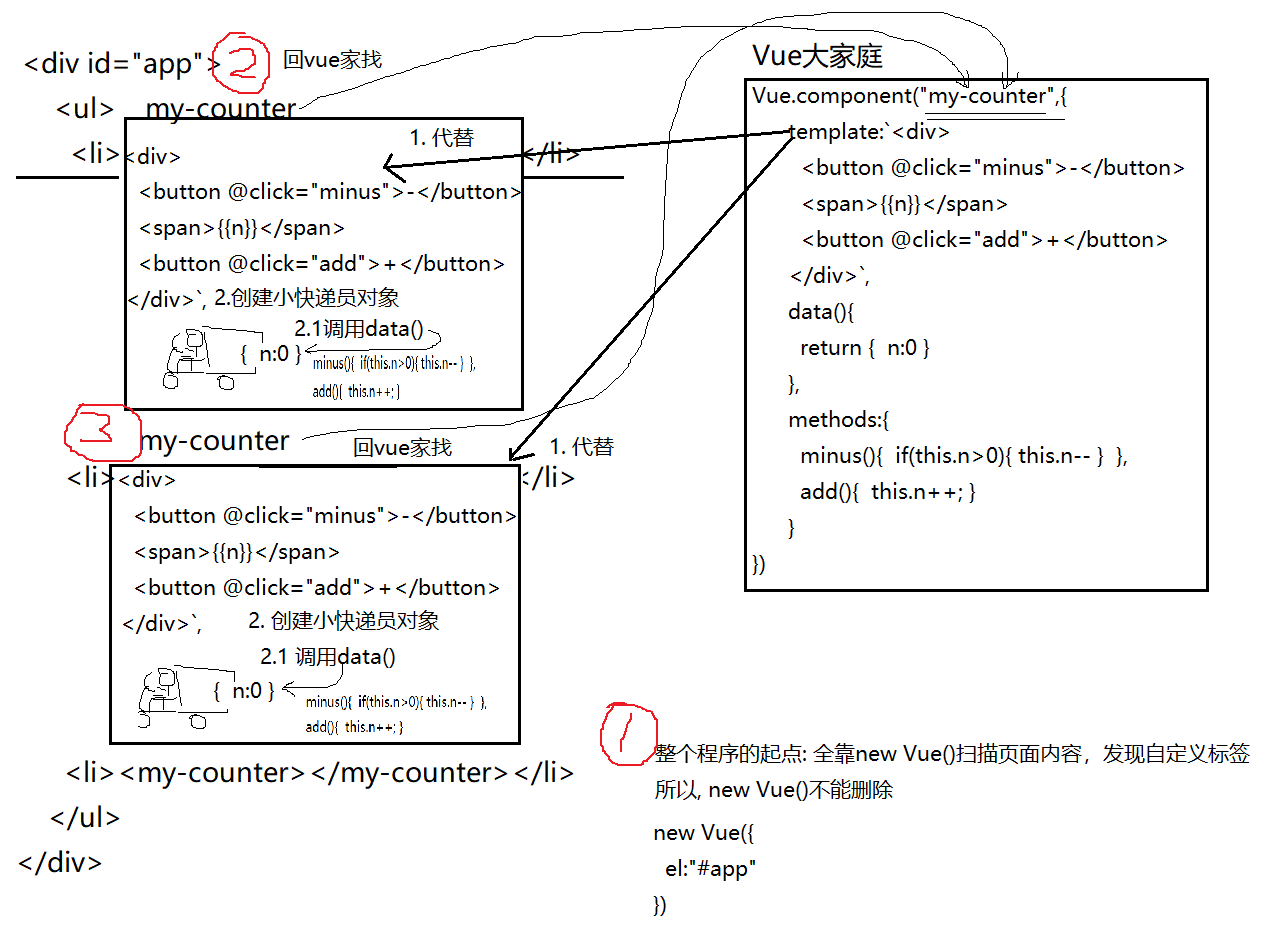
(1). new Vue()在扫描页面时，发现不认识的标签，就会去vue大家庭中查找是否有同名的自定义组件

(2). 只要找到同名的自定义组件，就会先用组件的template中的HTML片段代替页面中不认识的标签的位置

(3). 为这个区域临时创建一个缩微的vue对象的副本：

a. 自动调用自定义组件中的data()，为这个组件副本创建一个专属的data对象保存专属的数据  
 b. 引入组件对象中规定的methods等其他函数

(4). 结果: 界面中的一个自定义标签，就变成了一个缩微的小的功能区域！



7. 示例: 定义一个点+/-号，数量增减的组件，并在页面中反复使用

7\_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>      //因为button span button，这个组合在项目中反复出现，所以应该封装为一个组件      //组件名中包含多个英文单词，应该用-分隔，而不用驼峰命名！      Vue.component("my-counter",{        //组件的template也要求必须只能有一个唯一的父元素包裹        //因为组件中只有span的内容可能发生变化，所以，用{{n}}来标记        //因为两个按钮可点击执行加减计算        template:`<div>          <button**@click="minus"**>-</button>          <span>**{{n}}**</span>          <button **@click="add"**>+</button>        </div>`,        data(){  **return {**//新对象            //因为组件的template中只需要一个变量n            n:0          }        },        //因为组件的template中需要两个事件处理函数        methods:{          minus(){            if(this.n>0){ this.n-- }          },          add(){            this.n++;          }        }      })    </script>  </head>  <body>    <div id="app">      <ul>        <!--因为组件名叫my-counter，所以-->        <li><my-counter></my-counter></li>        <li><my-counter></my-counter></li>        <li><my-counter></my-counter></li>      </ul>    </div>    <script>      //任何情况下new Vue()都不能删除      //我们所学的所有功能之所以可以运行，都是建立在new Vue()会扫描页面元素的基础上！      new Vue({        el:"#app"      })    </script>  </body>  </html>  运行结果: |

**六. 组件化开发**

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

1. 什么是组件化开发: 将来所有的网页其实都是由组件拼成的！将来我们拿到一个网页之后不是立刻开始编码，而是先将网页的内容划分为多个组件，然后再多人分工协作开发出一个页面！

2. 为什么:

(1). 现代的前端开发，网页内容和开发难度都极大增加！

(2). 组件化开发可以把一个网页拆分为多个组件，由**多人协作开发**！——极大的提高了开发效率。

(3). 一个组件出错，不影响整个页面的加载和使用！——**低耦合**！

3. 何时: 今后大型的网页都应该使用组件化开发

4. 如何:

(1). 拿到网页之后，先将网页内容划分为多个独立的功能区域

划分的原则: 2个

**a. 从上到下、从左到右、从外到内的区域顺序**

**b. 是否会被重用！**

(2). **每个组件**都是**一个**独立的**js文件**，其中包含一个组件对象的定义

(3). **父级组件**中使用组件**标签引用子级组件**

(4). 将所有组件.js文件引入到HTML文件中，在<div id="app">内</div>使用父级组件的标签，引入整个组件内容

5. 问题: 子组件必须在其规定的父组件内使用才有意义。但是Vue.component()创建的组件，在哪里都能随意使用，没有限制！

6. 解决: 其实，vue中的组件分为三大类:

(1). **根组件**(**root** component): **new Vue()** 这个网页，甚至整个项目只有一个！

(2). **全局组件**: 可在任意位置使用的组件，没有限制

**Vue.component(...)**

(3). **子组件:** 仅限制在其父元素内才能使用的组件。如何:

a. **创建子组件对象**:

**var 组件对象名(驼峰命名)={** //普通对象

//但是内容却必须符合组件的要求

**template:`xxx`,**

**data(){ return { ... }},**

**... ...**

**}**

b. 在其父组件内使用**components属性**，标记哪个子组件属于哪个父组件:

父组件{

template:`xxx`,

data(){ return { ... } },

methods:{ ... },

... ...

**components: { 子组件对象名(驼峰) }**

} components可自动将驼峰命名翻译为-分隔

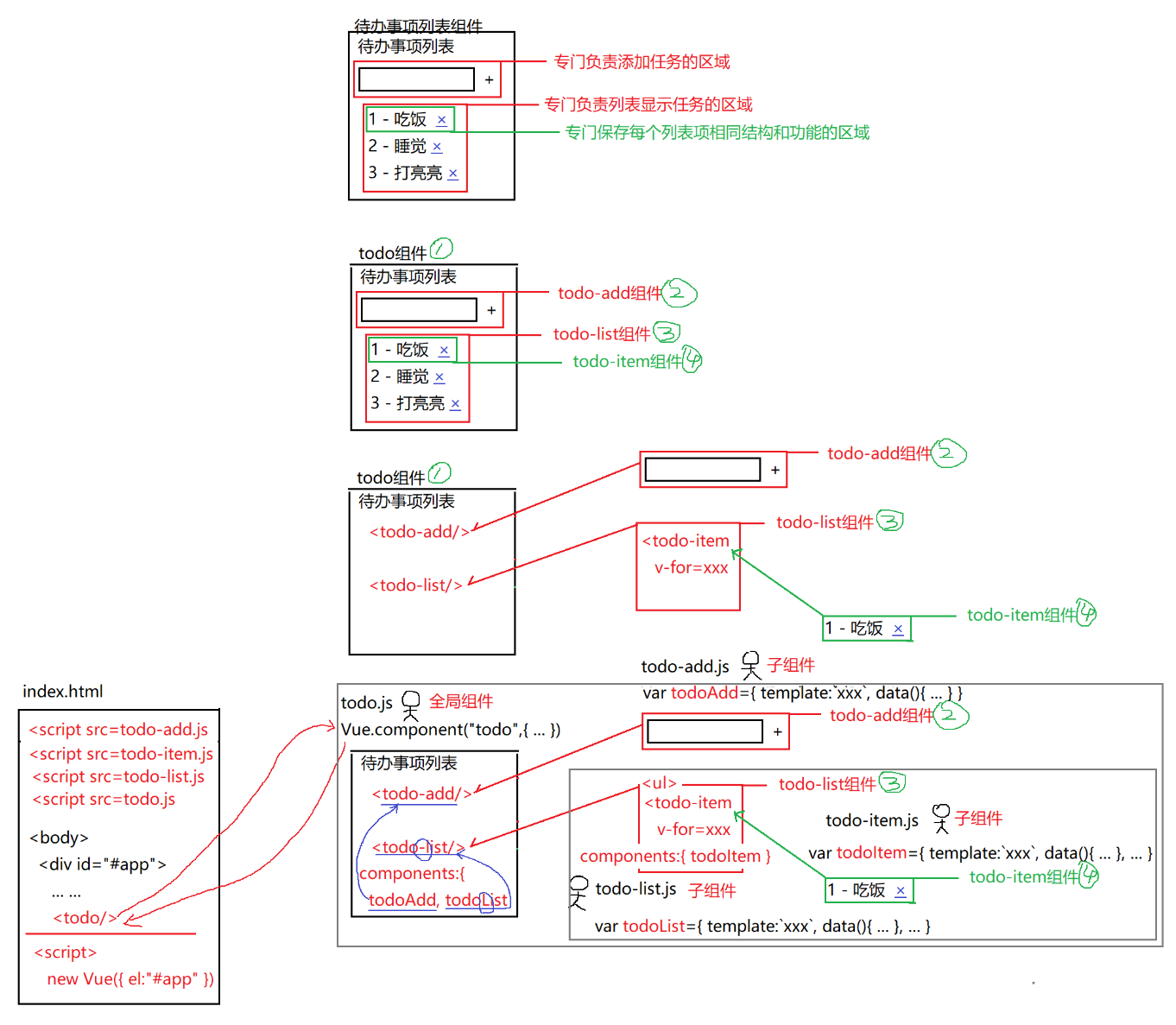
比如: todoList => <todo-list>

c. 结果: 在父组件的template中，可以使用子组件标签名(-分隔)，引入子组件的内容

d. 强调: **将组件引入页面时，子组件必须在其父组件之前引入！**

因为: 父组件的components属性中已经要使用子组件对象了！

7. 示例: 使用组件化开发的方式实现待办事项列表



todo / js/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 }    //自动翻译  todo-add   todo-list  }) |

todo / js/todo-add.js

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

todo / js/todo-list.js

|  |
| --- |
| var todoList={    template:`<ul>      <todo-item></todo-item>      <todo-item></todo-item>      <todo-item></todo-item>    </ul>`,  **components:{ todoItem }**    //自动翻译为 todo-item  } |

todo / js/todo-item.js

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

todo / 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>    <script src="js/todo-add.js">      //var todoAdd={ ... }    </script>    <script src="js/todo-item.js">      //var todoItem={ ... }    </script>    <script src="js/todo-list.js">      //var todoList={        //components:{ todoItem }      //}    </script>    <script src="js/todo.js">      //components:{ todoAdd, todoList }    </script>    </head>  <body>    <div id="app">      <todo></todo>      <!-- <todo-item></todo-item> 报错: todoItem被限制只能todoList下使用！只要超出todoList使用就报错！ -->    </div>    <script>      new Vue({        el:"#app"      })    </script>  </body>  </html>  运行结果: |

8. **组件间传参**: **父给子**:

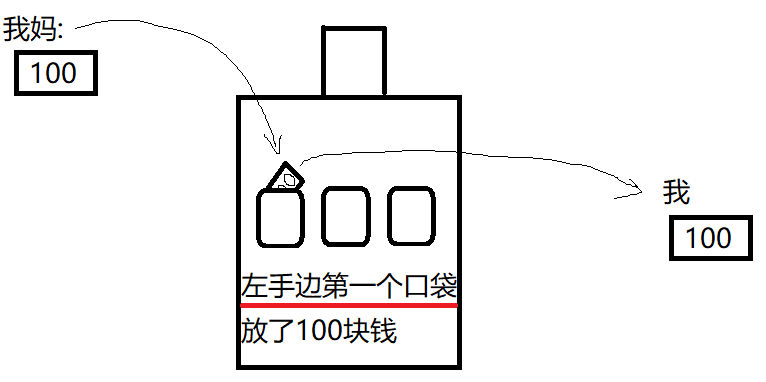
(1). 问题: 如果**多个组件**需要**共享**的**数据**，应该放在哪里？

(2). 错误的做法: 放在父组件中，指望通过继承关系，子组件顺理成章的可以直接使用父组件中的变量。

原因: Vue中组件是拥有专属的HTML+JS+数据的独立功能区域。组件与组件之间毫无关系！

(3). 正确的做法: 将**数据放在父组件中**，但是要**通过专门的手段**让父组件把数据传递给子组件——**组件间传参**！

(4). 如何: 2步:



a. 父组件放: 在父组件的template中子组件标签上

data(){  
 return {  
 **父组件变量: 值**

}

}

**<子组件 :自定义属性名="父组件变量"></子组件>**

意为: 先将父组件的一个变量值，保存到子组件身上一个自定义的属性(口袋)里

b. 子组件取: 在**子组件对象中添加props属性**

子组件对象:{

... ...

**props:[ "自定义属性名", ... ],**

意为: 从子组件身上，父组件装数据的同名属性(口袋)里取出父组件给的变量值

... ...

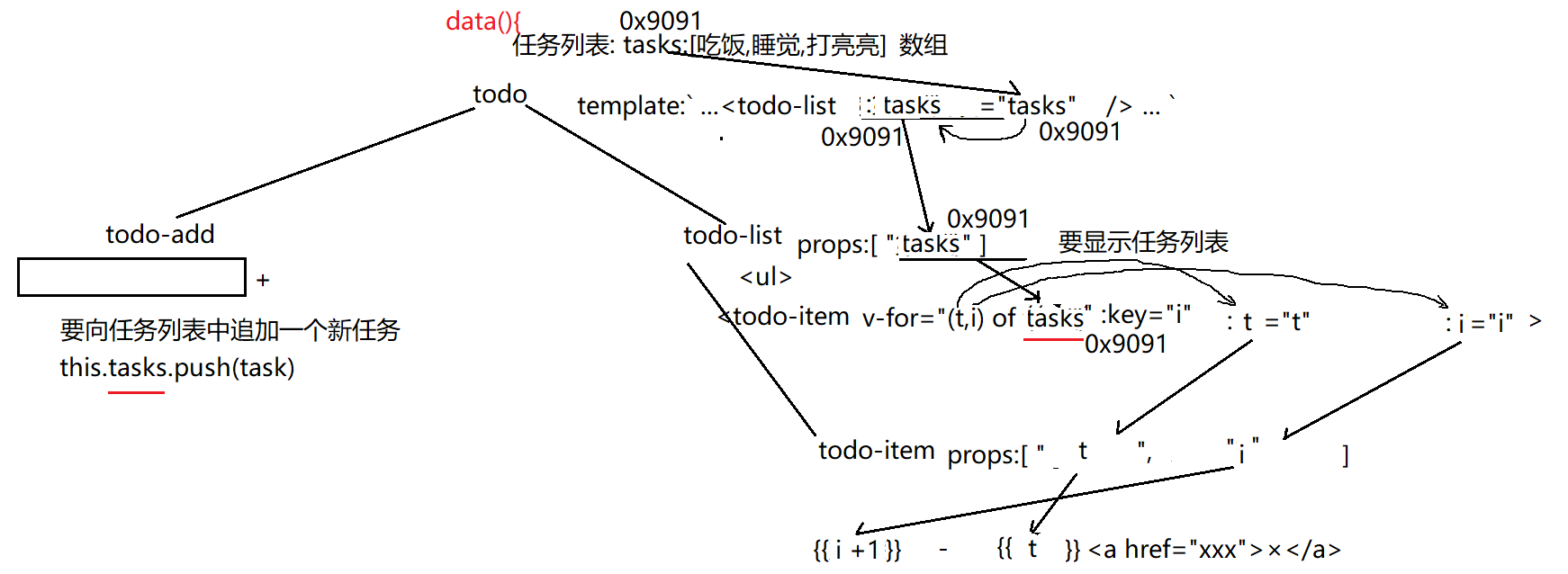
}

c. 结果: 在子组件对象中, props中的变量名用法和data中的变量完全一样！

d. props中的变量和data中的变量:

1). props中的变量值来自于外部给的

2). data中的变量值是自己给的。



9. 示例: 实现待办事项列表中的数据绑定

todo copy / js/todo.js

|  |
| --- |
| Vue.component("todo",{    template:`<div>      <h1>待办事项列表</h1>      <todo-add></todo-add>      <todo-list :tasks="tasks"></todo-list>    </div>`,    data(){      return {        tasks:["吃饭", "睡觉", "打亮亮"]      }    },    components:{ todoAdd, todoList }    //自动翻译  todo-add   todo-list  }) |

todo copy / js/todo-add.js

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

todo copy / js/ todo-list.js

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

todo copy / js/todo-item.js

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

todo copy / 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>    <script src="js/todo-add.js">      //var todoAdd={ ... }    </script>    <script src="js/todo-item.js">      //var todoItem={ ... }    </script>    <script src="js/todo-list.js">      //var todoList={        //components:{ todoItem }      //}    </script>    <script src="js/todo.js">      //components:{ todoAdd, todoList }    </script>    </head>  <body>    <div id="app">      <todo></todo>      <!-- <todo-item></todo-item> 报错: todoItem被限制只能todoList下使用！只要超出todoList使用就报错！ -->    </div>    <script>      new Vue({        el:"#app"      })    </script>  </body>  </html>  运行结果: |

总结:

1. 绑定语法+13种指令

(1). 如果元素的内容需要随变量自动变化: {{}}

(2). 如果元素的属性值需要随变量自动变化: :

(3). 控制一个元素显示隐藏: v-show

(4). 控制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). 只要反复生成多个相同结构的元素组成列表时: v-for :key="唯一标识"

(7). 只要绑定事件: @ $event

(8). 防止用户短暂看到{{}}: v-cloak和v-text

(9). 只要绑定原始HTML代码片段内容: v-html

(10). 如果元素的内容只在首次加载时绑定一次，之后都不会改变: v-once

(11). 保护内容中的{{}}不被编译: v-pre

(12). 今后只要想获得表单元素的值或状态: v-model

2. 绑定样式:

(1). 需要精确修改某一个css属性，就绑定style:

a. <元素 style="固定样式" :style="{css属性:变量名, ...}"

data:{  
 变量名:css属性值

... : ...

}

b. <元素 style="固定样式" :style="变量名"

data:{  
 变量名:{

css属性名: 属性值,

... : ...

}

}

(2). 只要批量修改一个元素的多个css属性就绑定class

a. <元素 class="固定class" :class="{class名:bool值, ...}"

data:{  
 变量名:true或false,

... : ...

}  
b. <元素 class="固定class" :class="变量名"

data:{  
 变量名:{  
 class名:true或false,

... : ...

}

}

3. 只要希望在页面加载时自动对元素执行一些初始化操作时就用自定义指令:

(1). 添加自定义指令:

Vue.directive("自定义指令名",{

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. 今后只要根据其他变量的值动态计算出一个属性值就用计算属性:

<元素>{{计算属性}}</元素>

new Vue({

el:"#app",

data:{...},

methods:{...},

computed:{  
 计算属性名(){

计算过程

return 计算结果

}  
 }

})

5. 希望将变量的原始值先加工后再显示给用户看时就用过滤器:

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

return 加工后的新值

})

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

6. 只要在vue中发送ajax请求，就用axios

axios.defaults.baseURL="服务器端接口的公共基础地址部分"

axios.get(  
 "服务器端接口地址的相对路径",

{

params:{ 参数名: 参数值, ... }

}  
).then(result=>{

... result.data...

})

或

axios.post(  
 "服务器端接口地址的相对路径",

"参数名1=参数值1&参数名2=参数值2&..."  
).then(result=>{

... result.data...

})

强调: 在vue内使用axios，then中必须用箭头函数，保持then内this与外部this一致，都指向当前new Vue()对象

7. vue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(){ ... }

8. 只要希望重用一块独立的功能区域就用组件:

(1). 定义组件

Vue.component(`组件标签名`,{

template:`HTML内容片段`,

data(){ return { 变量 } },

//其余和new Vue()完全相同

})

(2). 在HTML中使用自定义组件

<组件标签名/>或双标记也行

(3). 原理: new Vue()扫描到自定义组件标签，

a.组件的template中的HTML内容代替页面中<组件标签>位置。

b. 并为这个小区域专门创建一个缩微版的vue类型对象。

1). 调用组件的data()函数为当前组件副本创建一个专属数据对象副本。

2). 引入组件对象中的methods等其他内容到当前组件对象副本中

9. 组件化开发:

(1). 三种组件:

a. 根组件: new Vue()

b. 全局组件: Vue.component(...)

c. 子组件: 3步

1). var 子组件对象名={

内容必须符合组件的要求

}

子组件对象名必须是驼峰命名

2). 父组件对象中:{

... ...

components{ 子组件对象名, ... ,... }

}

子组件对象名必须是驼峰命名

3). 父组件template中用<子组件标签名/>引入子组件内容

components会将子组件对象名的驼峰命名自动翻译为-分隔

所以, 使用子组件标签时，要用-分隔多个单词

(2). 组件间传参: 父给子

a. 父组件给:

<子组件 :自定义属性名="父组件变量">

b. 子组件取:

props:["自定义属性名"]

结果: 在子组件内，props中的"自定义属性名"与子组件自己data中的变量用法完全相同！

今日对应小程序视频列表:

小程序->在线->VUE->day02 10 计算属性

小程序->在线->VUE->day03 3. 过滤器

4. axios

5. 生命周期

6.1 创建单个组件

6.2 Vue组件化开发todo

作业:

1. 复习今日问题清单: 小程序->首页->VUE->day03、day04

2. 看小程序视频完成待办事项列表添加和删除功能：

小程序->在线->VUE->day03 6.2 Vue组件化开发todo 。。。

3. 看小程序视频，预习最后一天重要内容

小程序->在线->VUE->day04 1. SPA单页面应用...

2. 使用vue/cli工具创建学子商城项目脚手架

3. vue脚手架文件夹结构

小程序->在线->VUE->day05 -1. 懒加载