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->day03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->day03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 ->day03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. 过滤器

2. axios

3. 组件生命周期

4. 组件化开发  
5. SPA

一. 过滤器:

1. 什么是过滤器: 对变量的原始值进行加工后再显示出来的一种特殊函数

2. 为什么: 有些值不能直接给人看，必须加工后才能给人看。

比如: sex:1 时间ms

3. 何时: 今后只要一个变量的原始值必须加工后才能给人看时，都要定义过滤器

4. 如何定义:

Vue.filter("过滤器名",function(oldVal){//要求1: 必须有一个形参，来接住变量的原始值

//要求2: 必须返回加工后的新值

return 新值

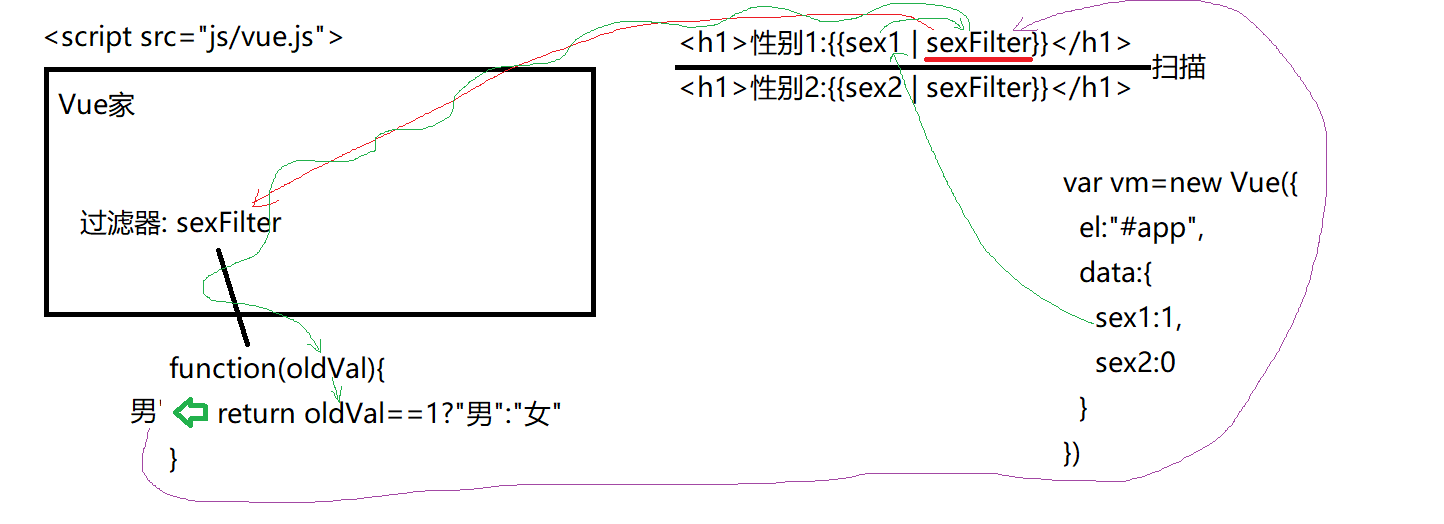
})

5. 使用:

<元素>{{ 变量 | 过滤器}}</元素>

连通

6. 原理: 变量的原始值先传给过滤器的形参oldVal，在过滤器内部加工后，返回新值，显示在绑定的位置上给人看。



7. 过滤器可以传参:

(1). 定义时:

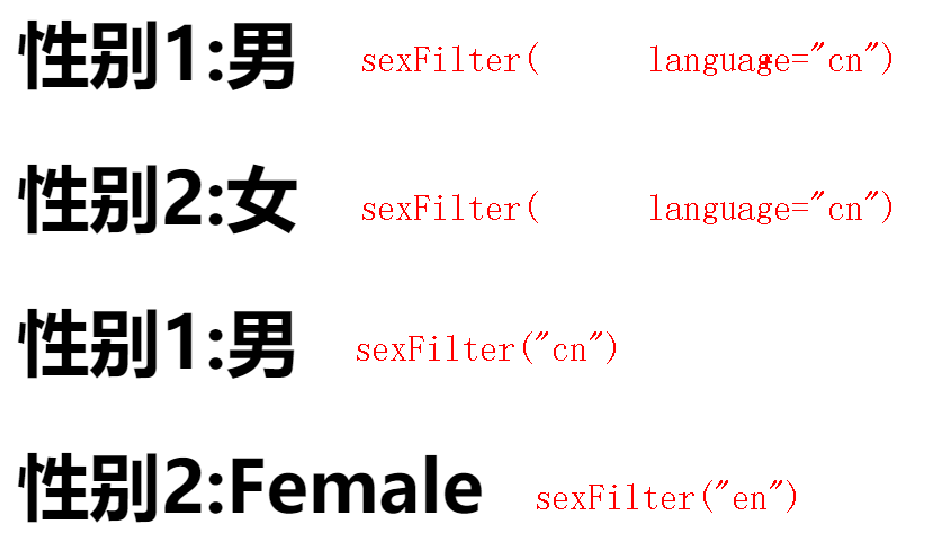
Vue.filter("过滤器名",function(oldVal, 自定义形参,...){

return 加工后的新值

})

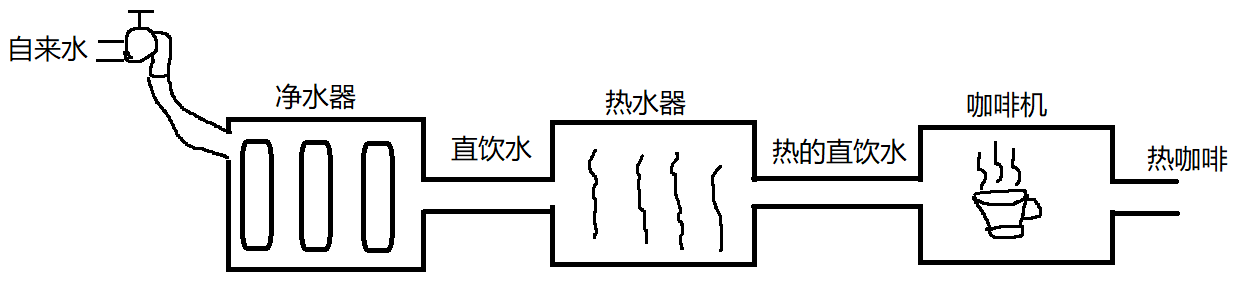
(2). 使用时:

{{变量 | 过滤器名(实参值, ...)}}



8. 过滤器还可连用:

强调: 后一个过滤器获得的值已经不是原始值了，而是前一个过滤器加工后的中间产物



9. 示例: 使用过滤器加工性别的1和0

|  |
| --- |
| <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <meta http-equiv="X-UA-Compatible" content="ie=edge">    <title>Document</title>    <script src="js/vue.js"></script>    <script>                                //语言:en  cn      Vue.filter("sexFilter",function(oldVal,language="cn"){        if(language=="cn"){          return oldVal==1?"男":"女"        }else{          return oldVal==1?"Male":"Female"        }      })      Vue.filter("sexIcon",function(oldVal){        if(oldVal==1){          return "♂";        }else if(oldVal==0){          return "♀";        }else if(oldVal=="男"||oldVal=="Male"){          return oldVal+" ♂";        }else{          return oldVal+" ♀";        }      })    </script>  </head>  <body>  <div id="app">    <h1>性别1:{{sex1 | sexIcon}}</h1>    <h1>性别2:{{sex2 | sexIcon}}</h1>    <h1>性别1:{{sex1 | sexFilter | sexIcon}}</h1>    <h1>性别2:{{sex2 | sexFilter | sexIcon}}</h1>    <h1>性别1:{{sex1 | sexFilter("cn") | sexIcon}}</h1>    <h1>性别2:{{sex2 | sexFilter("en") | sexIcon}}</h1>  </div>  <script>  var vm=new Vue({    el:"#app",    data:{      sex1:1,      sex2:0    }  })  </script>  </body> |

二. axios:

1. 什么是: 专门在各种平台发送ajax请求的基于Promise的函数库

(1). 各种平台: axios不仅可用于vue中，而且还可用于网页中、nodejs服务端都可用

(2). 发送ajax请求: axios只专注于做好发送ajax请求这一件事

(3). 基于Promise: axios请求回来自动触发的回调函数要放在Promise的.then()中，不再使用回调函数的方式。

2. 为什么:

(1). 今后只要发送ajax请求，都只用一句话就够了！再不会写xhr4步或6步

(2). jQuery虽然有$.ajax()很好用，但是$.ajax这一个函数是混在整个jQuery巨大的函数库内的，没有独立出来。

(3). Vue中几乎不用执行DOM操作，自然也就用不到jQuery

(4). 如果只以为$.ajax()一个函数就要把整个jQuery全部函数库引入，有点儿得不偿失，小题大做！

(5). 所以才需要第三方的专门发送ajax请求的独立的函数库

3. 何时: 今后在任何平台(框架、网页、服务器端nodejs...)发送ajax请求，都可用axios函数库

4. 如何:

(0). 引入axios.min.js: <script src="js/axios.min.js">

(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 里获得的就是服务器端返回的结果(已经被json.parse()编译成对象了)

})

(2). 发送post请求:

axios.post("服务器端接口地址",{ 参数1:值1, ...:... }).then(result=>{

//result.data 里获得的就是服务器端返回的结果(已经被json.parse()编译成对象了)

})

(3). 提前配置统一的服务器端接口域名地址:

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

结果:

axios.get("/index").then(result=>{ ... })

5. 示例: 发送axios请求，获取服务器端学子商城数据

|  |
| --- |
| <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>  /\*回顾:  $.ajax({    url:"服务器端接口地址",    data:"get或post",    dataType:"json",    success:function(result){//请求成功后自动调用    }  })\*/  axios.defaults.baseURL="http://xzserver.applinzi.com";  //获得学子商城首页6个商品  axios.get("/index")  .then(result=>{//请求成功后自动调用    console.log(result.data);  })  //获得5号商品的详细信息:  axios.get("/details",{    params:{ lid:5 }  }).then(result=>{    console.log(result.data);  })  //发送登录请求  axios.post("/users/signin",{uname:"dingding", upwd:"123456"})  .then(result=>{    console.log(result.data);  })  </script>  </body> |

三. 组件生命周期:

1. 什么是: 一个组件或一个"页面"加载过程中所经历的每个阶段

2. 为什么: 因为有些初始化操作，是应该在页面或组件加载完成后，就自动执行的！不需要人为手工操作

3. 何时: 今后只要希望在页面或组件加载完成后自动执行一些初始化任务时就需要用到组件或页面的生命周期

4. 一个"页面"或组件加载过程中需要经历哪些阶段: (仅以new Vue()举例) 4个阶段

(1). 创建(create)阶段: (必经)

a. 创建new Vue()对象

b. 加载data对象到new Vue()中

c. 暂时没有虚拟DOM树，可能无法执行DOM操作

(2). 挂载(mount)阶段: (必经)

a. 根据el所指扫描页面，创建虚拟DOM树, 并首次将data中的变量值渲染到页面显示

b. 既又了data，又有了虚拟DOM树，既可以操作data中的变量，又可以对页面执行DOM操作

(3). 更新(update)阶段: (只有在data中的变量被修改后才自动触发)

(4). 销毁(destroy)阶段: (只有手动调用$destroy()函数，销毁当前组件时才自动触发)

5. 生命周期钩子函数:

(1). 什么是: 在每个生命周期阶段自动触发的特定的回调函数(类似于Dom中的事件处理函数)

(2). 何时: 只要希望在特定的生命周期阶段执行一些操作时，都要把操作写在对应的生命周期钩子函数内，才能自动触发

(3). 包括: 8个 (每个阶段前后都有一个钩子函数)

beforeCreate()

创建(create)

created() //可发送axios请求，因为此时已经有data了，如果请求回来数据，可以放在data中，并自动更新到页面上

//小风险: 万一axios请求回来的很快，而后续mount阶段没有执行完，此时是不能执行DOM操作的。

beforeMount()

挂载(mount)

mounted() //最稳妥的发送axios请求的时机：因为此时既有data，又有虚拟DOM树。

beforeUpdate()

更新(update)

updated()

beforeDestroy()

销毁(destroy)

destoryed()



6. 示例: 使用生命周期钩子函数发送ajax请求

|  |
| --- |
| <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <meta http-equiv="X-UA-Compatible" content="ie=edge">    <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}} <a href="javascript:;" @click="del(i)">×</a>      </li>    </ul>    <a href="javascript:;" @click="destory">销毁当前new Vue()</a>  </div>  <script>  var vm=new Vue({    el:"#app",    data:{      products:[],//先定义一个空数组等着接axios请求回来的6个商品对象    },    methods:{      del(i){        this.products.splice(i,1);//复习数组函数      },      destory(){        this.$destroy()      }    },    beforeCreate(){      console.log(`创建data对象前自动触发...`)    },    created(){      console.log(`创建data对象后自动触发...`)    },    beforeMount(){      console.log(`创建虚拟DOM树前自动触发...`)    },    mounted(){      console.log(`创建虚拟DOM树后自动触发...`)      console.log(`    发送axios请求`);      axios.get("http://xzserver.applinzi.com/index")      .then(result=>{        console.log(`收到响应结果，并放入data中的products变量`)        this.products=result.data;//修改了data中的products，就会触发update阶段      })    },    beforeUpdate(){      console.log(`更新data中的变量前自动触发...`)    },    updated(){      console.log(`更新data中的变量后自动触发...`)    },    beforeDestroy(){      console.log(`在销毁当前组件或页面前自动触发...`)    },    destroyed(){      console.log(`销毁当前组件或页面后自动触发...`)    }  })  </script>  </body> |

四: 组件化开发:

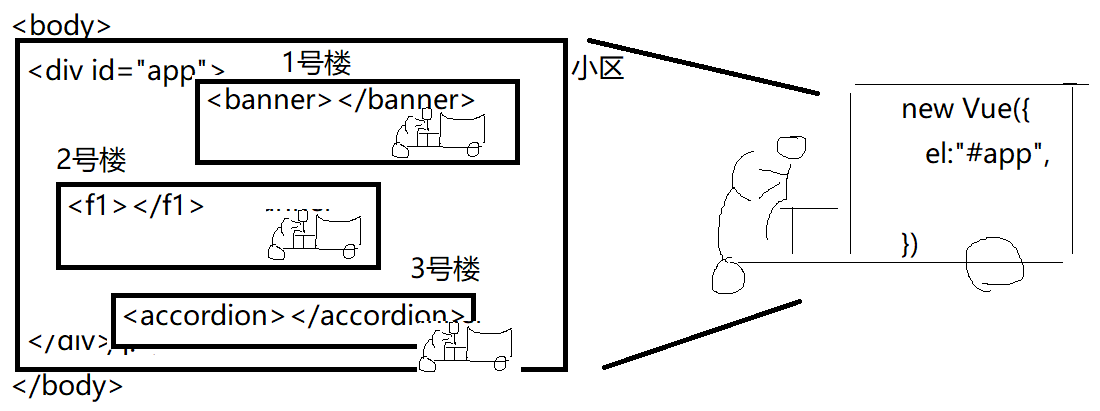
1. 什么是组件: 拥有专属的HTML+CSS+JS+数据的可重用的页面独立功能区域

2. 为什么: 重用，提高开发效率

3. 何时: 今后只要发现页面中有需要重用的区域，都要封装为组件。

4. 如何定义组件:

//其实组件对象就是缩微版的new Vue()，功能完全一样，只不过只监控组件这一小块区域而已。



//组件名: 因为将来组件名会成为HTML中的标签名，又因为HTML不区分大小写，所以，当组件名包含多个单词时，不要用驼峰命名！应该全小写，并用-分割每个单词

Vue.component("组件名",{

//组件的成员：new Vue()有什么成员，组件就可以有什么成员，除了个别有变化之外。

~~el:" ... ", //因为组件是先定义，后反复使用，所以定义组件时，页面上还没有组件的影子，所以不能查找~~

//因为组件是可以复制并反复使用的，所以组件的HTML应该称为模板template

template:`<唯一父元素 class="组件类名">

HTML片段

</唯一父元素>`,

~~data:{ ... }, //因为每个组件的副本都应该有自己专属的data变量，组件的副本与副本之间不能的data变量不能互相影响，所以，data不能只是一个写死的对象了~~

data(){ //今后每使用一次组件的副本，就自动调用data()函数，每次都可以返回一个新的data对象给当前组件副本，就保证了每个组件副本的data都是专属的互不干扰的！

return { //new Object() //相当于new Vue()中的data{}

变量:值

}

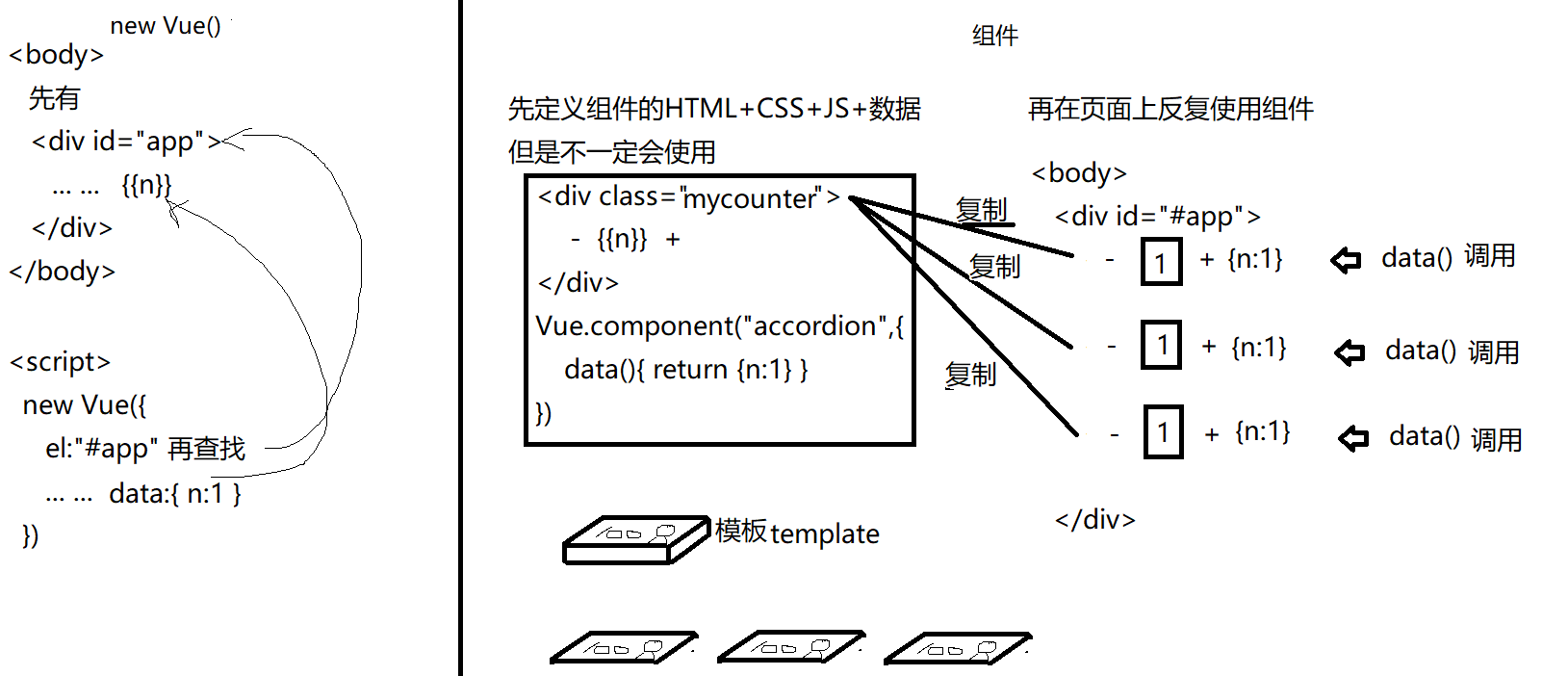
}

//其余成员，同new Vue()是完全一样的了！

methods:{ ... },

...

})



5. 如何使用组件: 组件其实就是一个可重用的自定义HTML标签而已。组件名就是标签名

比如: Vue.component("my- counter ",{ ... })

在页面上使用时: <my-counter></my-counter>

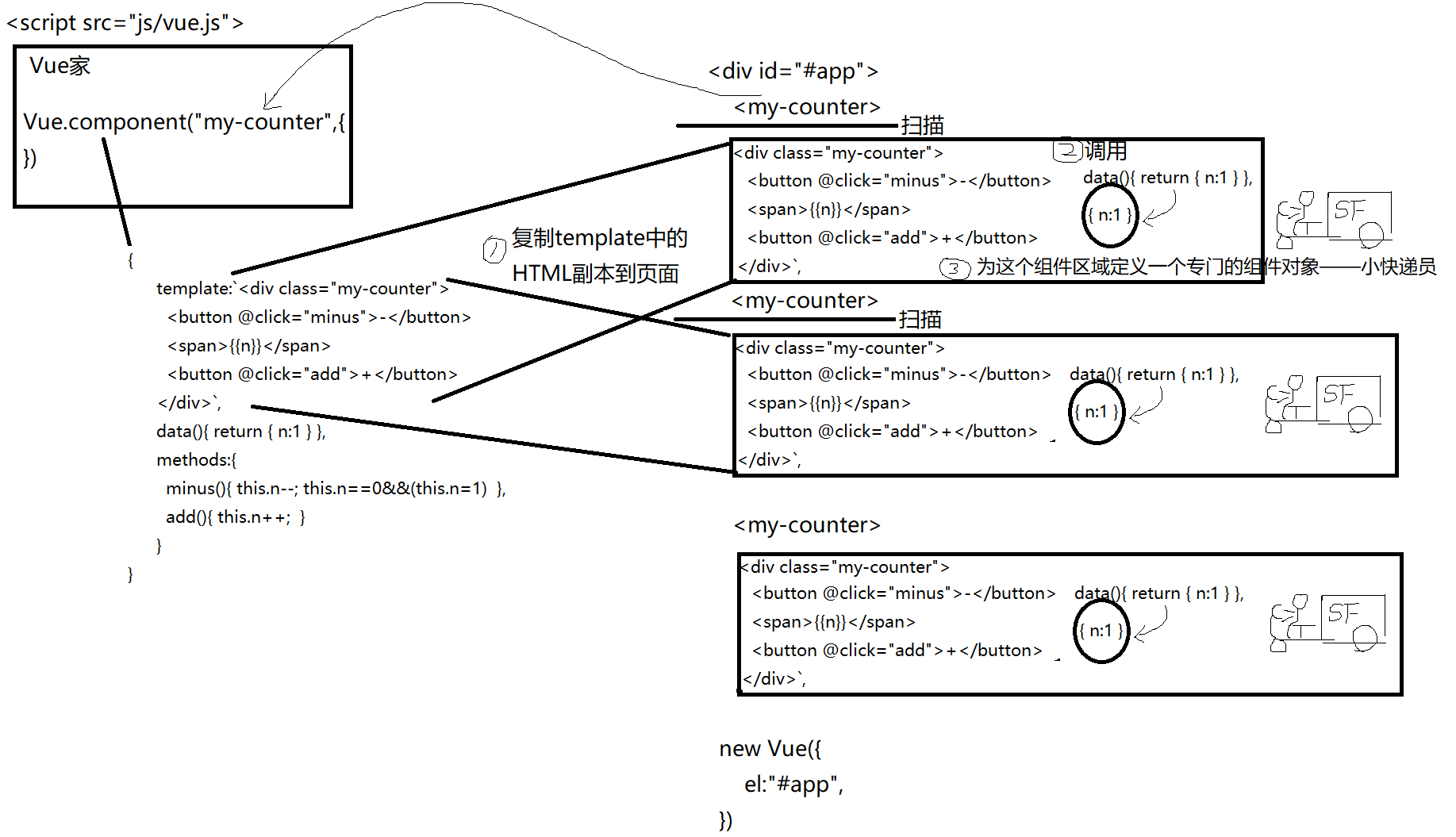
6. 原理:

(1). new Vue()在扫描页面时会发现不认识的标签，就会回Vue家找有没有对应的自定义组件

(2). 如果找到自定义标签对应的组件，先复制组件的template中的HTML模板到页面中代替自定义标签所在的位置

(3). 自动调用一次组件的data()函数，为这个组件副本创建一个专属的data对象

(4). 为这次组件的副本创建一个专属的Vue类型对象(快递员)监控这一小块组件副本区域。



7. 示例: 定义修改数量的组件

|  |
| --- |
| <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <meta http-equiv="X-UA-Compatible" content="ie=edge">    <title>Document</title>    <script src="js/vue.js"></script>    <script>      Vue.component("my-counter",{        template:`<div class="my-counter">          <button @click="minus">-</button>          <span>{{n}}</span>          <button @click="add">+</button>        </div>`,        data(){          return {//相当于以前的data            n:1          }        },        methods:{          minus(){            this.n--;            this.n==0&&(this.n=1)          },          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>  var vm=new Vue({    el:"#app",    data:{      }  })  </script>  </body> |

8. 组件化开发: 将来所有的页面都是由多个组件组成的。将来做页面第一件事不是上来就写HTML代码！而是先划分组件区域！然后，分头定义各个组件，最后再组合成一个页面。

9. 为什么:

(1). 便于大项目分工协作:

a. 问题: 前端的工作量和代码量越来越大，但是操作系统有限制，一个文件只能同时被一个人编辑，无法分工协作！

b. 解决: 将一个大的页面划分为多个小的组件，分给多个人去并行开发，效率比一个人开发要高的多。

(2). 松耦合: 人与人之间互不影响。一个人出错，不会牵连大家！

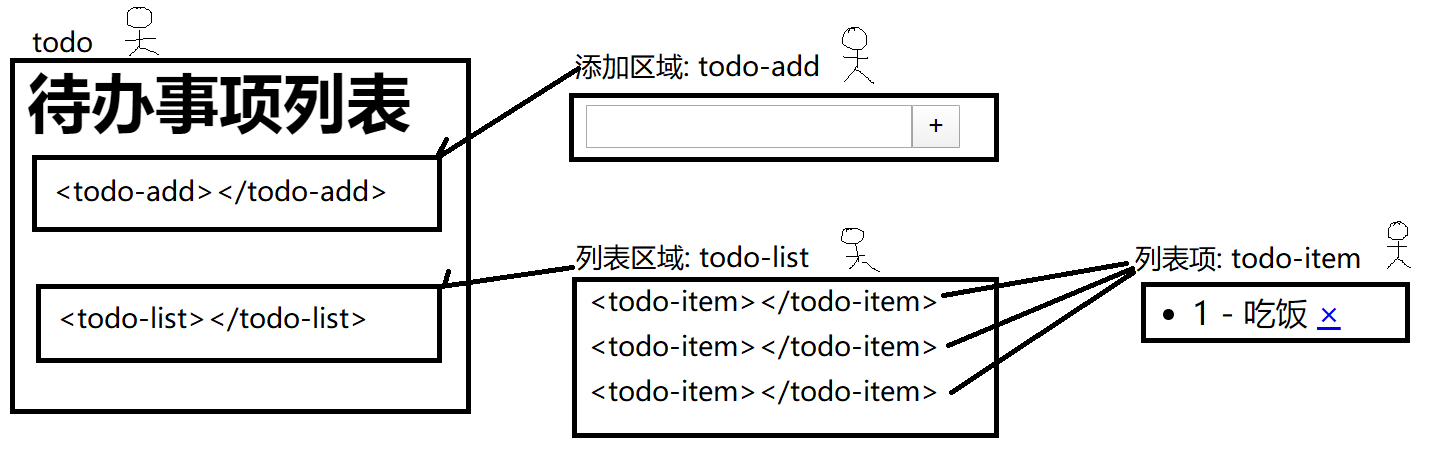
10. 何时: 今后所有的框架，都是组件化开发的！

11. 如何: 仅以"待办事项列表"需求为例

(1). 拿到界面先划分组件: 原则:

a. 按功能/按区域划分

b. 按是否重用划分



(2).将每个组件在独立的js文件中创建为一个组件对象:

(3). 在上级组件中通过子组件标签的方式引入子组件

(4). 在.html文件中先把所有组件js文件都引入页面(排名暂时不分先后)，然后在<div id="app">中加入父组件的标签

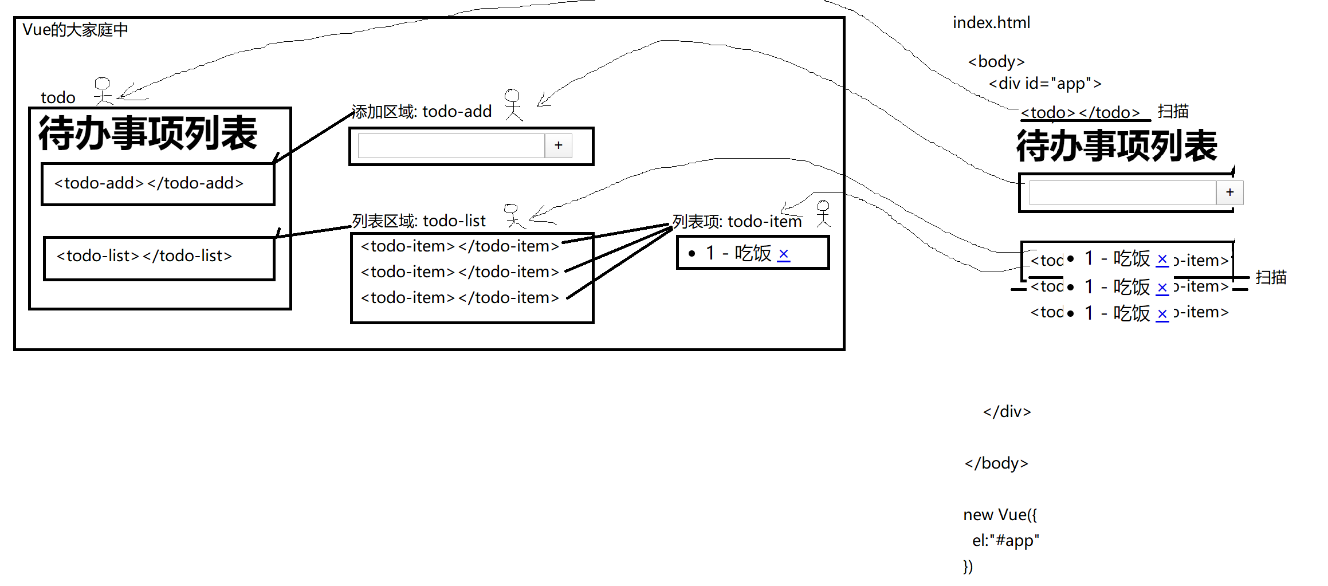
12. 原理:   
 (1). new Vue()开始扫描<div id="app">中的所有内容

(2). 只要找到一个不认识的标签，就去Vue家里找有没有对应的组件

(3). 只要发现对应的组件，就用组件的template模板，代替<div id="app">不认识的标签的位置。

(4). new Vue()会再次扫描刚替换上来的组件的HTML代码，如果又发现不认识的标签，则依然会继续回家找对应的子组件，替换父组件中子组件标签的位置。

(5). 直到<div id="app">中，所有的标签都是HTML标准固定的标签时，整个页面的加载才算完成.



13. 问题: 按理说子组件应该只在其规定的父组件内才有效。但是Vue.component()创建的组件，可以随意在任意位置使用！

14. 组件的分类: 3类：

(1). 根组件: new Vue()负责扫描并监控整个页面

(2). 全局组件: Vue.component()创建的组件。可在放在任意位置，没有任何限制！

(3). 子组件: 规定只能在指定的一个父元素下使用的组件

a. 何时: 如果一个组件规定只能在一个父元素下才能使用，都要创建为子组件

b. 如何: 2步:

1). 定义一个普通的组件对象: 其实就是一个普通的js对象中包含组件的属性成员

强调: 对象名其实就是组件名转换为驼峰命名的形式：去横线变驼峰

比如: 想定义一个<todo-item>的组件

var todoItem={ ... }

var 对象名={

template:`...`,

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

methods:{ ... },

... ....

}

2). 在这个子组件的父组件中添加一个components成员:

父组件:{

template:`...`,

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

methods:{ ... },

... ...,

components:{ 子组件对象名, ... }

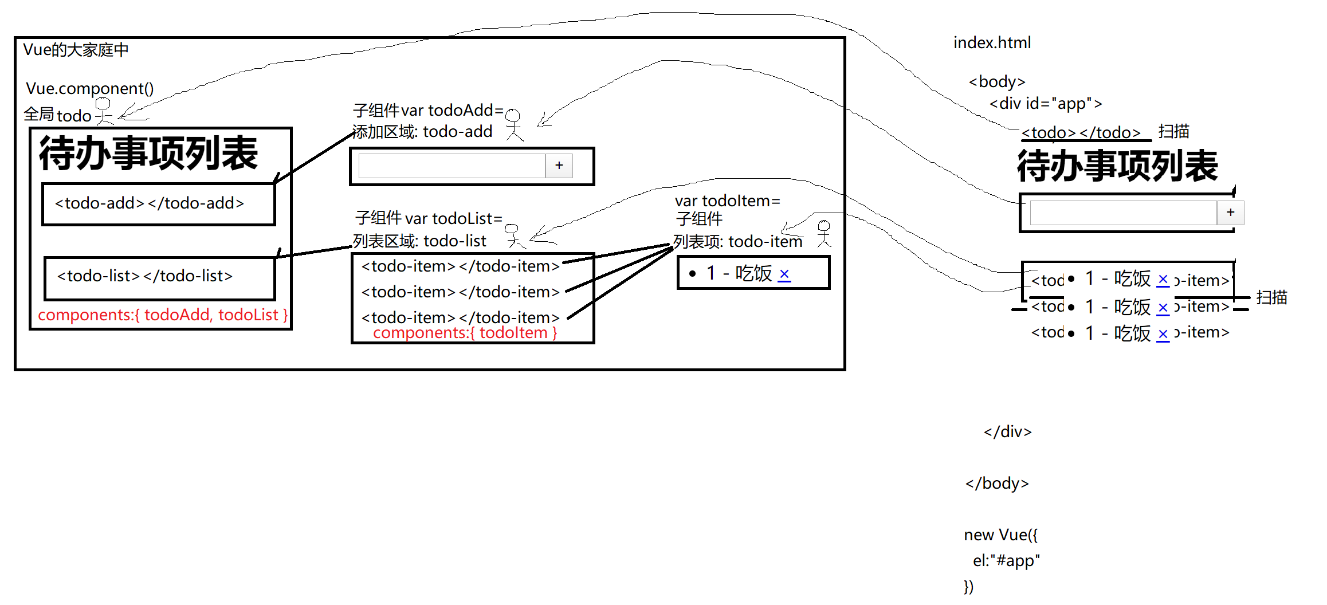
}

c. 结果:

1). components中的组件，只能在当前父组件template中使用。一旦在当前父组件之外使用，就会报错!

2). 虽然子组件对象名采用的是驼峰命名，但是components会将驼峰命名自动翻译为-分割。比如: components: { todoItem }， 在template中应该用<todo-item>

3). 强调: 一旦有了父子组件的关系，在引入时，子组件必须先于父组件引入！



15. 示例: 使用组件化开发实现todo案例的HTML内容

5\_todo-add.js

|  |
| --- |
| var todoAdd={    template:`<div class="todo-add">      <input type="text"><button>+</button>    </div>`  }; |

5\_todo-item.js

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

5\_todo-list.js

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

5\_todo.js

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

5\_todo.html

|  |
| --- |
| <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <meta http-equiv="X-UA-Compatible" content="ie=edge">    <title>Document</title>    <script src="js/vue.js"></script>    <script src="5\_todo-add.js"></script>    <script src="5\_todo-item.js"></script>    <script src="5\_todo-list.js"></script>    <script src="5\_todo.js"></script>  </head>  <body>  <div id="app">    <todo></todo>  </div>  <script>  var vm=new Vue({    el:"#app"  })  </script>  </body> |

16. 组件间传值:

(1). 问题: 组件化开发中，子组件无权使用父组件中data中的数据，因为组件的数据都是专属的！

(2). 解决: 组件间传值: 4种:

a. 父给子:

1). 何时: 在子组件中想使用父组件的数据时

2). 如何: 2步:

i. 父组件template中:用绑定语法给子组件标签添加一个自定义属性

父组件:{

template:`

<子组件 :自定义属性="父组件变量"></子组件>

`

}

ii. 子组件对象中: 使用props成员获得同名属性的值

子组件={

props:[ "自定义属性" ]

}

3). 结果: 在子组件内，props中的属性名的用法和data中变量的用法完全一样！只不过来源不同而已。Data中的变量是自己的。而props中的属性是外部给的。

4). 使用组件间传参实现todo案例的数据绑定

5\_todo-item.js

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

5\_todo-list.js

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

5\_todo.js

|  |
| --- |
| Vue.component("todo",{    template:`<div class="todo">      <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 }  }) |

b. 子给父: 看作业中小程序视频

c. 兄弟间: 看作业中小程序视频

d. ref和$refs: 看作业中小程序视频

总结:

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

Vue绑定原理: 访问器属性+虚拟DOM树

new Vue({

el:"#app",

data:{ ... },

methods:{ ... }, //调用时加()，不缓存结果，会反复计算

computed:{ ... },//调用时不加(), 缓存结果，不会反复计算

watch:{ ... }

})

绑定和指令:

1. 如果元素内容可能发生变化: {{}} v-text

2. 如果元素的属性值发生变化: :

3. 控制一个元素显示隐藏：v-show

4. 两个元素二选一显示隐藏: v-if v-else

5. 多个元素多选一显示隐藏: 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. 绑定事件: @ $event

8. 绑定HTML内容: v-html

9. 防止用户短暂看到双花括号: v-cloak v-text

10. 只在开始时绑定一次: 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

绑定样式:

1. 只操作一个元素的一个css属性值:

<元素 style="不变的css属性" :style="变量"

data:{

变量:{

Css属性: "属性值",

... : ...

}

}

2. 批量修改一个元素的多个css属性值: class

<元素 class="不变的class" :class="变量"

data:{

变量:{

Class名1: true或false,

... : ...

}

}

自定义指令:

Vue.directive("指令名",{

inserted(domElem){

}

})

作业:

1. (必须)看小程序视频学习待办事项列表案例:

小程序->在线->VUE->day03 6 Vue组件化开发 todo 实现封装组件 HTML和props传值...

2. (必须)看小程序视频学习组件间传参与ref:

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. (扩展)看小程序视频学习JS模块化开发

小程序->在线->VUE->day03 模块化(7个视频) 1-手工实现 2-CommonJS/AMD ... ...