正课：

1. 绑定样式:

2. 自定义指令:

3. 计算属性:

4. 过滤器:

5. axios:

6. 组件化开发:

一. 绑定样式:

1. 绑定内联样式:

<元素 style="不变的内联样式" :style="变量">

data:{

变量:{

css属性: "值",

... : ...

}

}

结果: vue会将变量对象中的属性翻译为style字符串格式，并和之前不带:的style合并为一个style.

比如: <div style="width:100px; height:100px; position:fixed" :style="divStyle">

data:{

divStyle:{

left:"100px",

top:"50px"

}

}

结果: <div style="width:100px; height:100px; position:fixed; left:100px; top:50px">

简写: <元素 style="不变的内联样式" :style="{css属性1:变量1, css属性2:变量2, ...}">

data:{

变量1:"css属性值",

变量2:"css属性值",

}

比如: <div style="width:100px; height:100px; position:fixed" :style="{left:left, top:top}">

data:{

left:"100px",

top:"50px"

}

结果还是: <div style="width:100px; height:100px; position:fixed; left:100px; top:50px">

2. 绑定class:

|  |
| --- |
| 将来只要发现class也会变化，就也用:class=xxx绑定 |

(1). 将整个class作为一个字符串绑定

问题：语法晦涩，且不便于单独操作一个class

(2). 推荐的做法还是将class作为对象来绑定

a. 标准写法:

<元素 class="不变的class" :class="变量"

data:{

变量:{

css样式类1: true或false,

css样式类2: true或false

}

}

结果：vue会仅将值为true的class加入到元素上，并和之前不带:的class合并使用。

比如: <span :class="spanClass">

data:{

spanClass:{

success: true,

fail:false

}

}

结果: <span class="success">

b. 简写: <元素 class="不变的class" :class="{class1:条件}"

比如: <span :class="{{success:success, fail:fail}}">

data:{

success: true,

fail:false

}

结果: <span class="success">

|  |
| --- |
| :前的css样式类名，虽然可以自定义，但是必须要求在样式表中有对应名称的css样式类才行。 |

二. 自定义指令:

|  |
| --- |
| 当vue提供的指令不够用时，就可自定义指令 |

1. 为整个Vue类型添加一个自定义指令:

Vue.directive("指令名", {

inserted(domElem){ //会在带有当前指令的dom元素被渲染到页面上后自动触发

//domElem会自动获得当前拥有这个指令的DOM元素对象

//对当前所在的DOM元素执行DOM操作

}

})

强调：定义指令时，指令名一定不要加v-前缀

2. 使用指令:

<元素 v-指令名 >

强调：使用指令时必须加v-前缀

3. 原理:

(1). Vue.directive()创建的指令其实是保存在整个vue大家庭中的。不专属于某个页面或某个new Vue()

(2). 每次new Vue()时，才扫描HTML页面。扫描时，只要碰到一个v-xxx指令，就会在Vue大家庭中查找是否包含该指令。（所以，new Vue()中即使没有数据和方法，也不能删除new Vue()）

(3). 只要找到该指令，就自动调用指令的inserted()函数，对当前DOM元素执行已经提前定义好的DOM操作！

三. 计算属性:

1. 什么是计算属性: 不实际存储属性值，必须通过其他属性的值计算，才能得出自己属性的值。

2. 为什么: 不是页面上需要的一切，数据库里都准备好了。有些数据，就需要临时计算获得。

3. 何时：只要页面上需要一个值，数据库中没有，但是可通过其他属性计算出来，就用计算属性

4. 如何：

(1). 在new Vue()中定义计算属性：

new Vue({

el:"#app",

data:{ ... },

methods:{ ... },

watch:{ ... },

computed:{

//虽然是一个属性，但是过程需要用一个函数计算出来，所以计算属性的本质其实是一个函数。只不过函数名是一个名词——属性名

属性名(){

... ...

return 计算结果

}

}

})

(2). 计算属性的用法和data中普通属性的用法完全一样！

5. 特点:

(1). 如果依赖的普通属性发生变化，则计算属性会自动重新计算

(2). methods vs computed

a. methods中的方法，必须用()调用才能执行，且每调用一次，就重复执行一次。vue不能缓存methods的计算结果。

b. computed中的计算属性，不用()就可执行绑定，且vue会缓存computed计算的结果。只要依赖的其它属性不改变，就不需要重新计算。除非依赖的属性改变了，才会自动计算。

四. 过滤器(filter):

1. 什么是过滤器: 对原始的变量值进行再加工后再显示的特殊函数

2. 为什么: 有些变量的原始值，不能直接给人看

比如: sex time status

3. 何时: 只要变量的原始值不能直接给用户看时，就要用过滤器

4. 如何：

(1). 先给Vue大家庭中添加一个过滤器

Vue.filter("过滤器名",function(oldValue){

//oldValue会接住变量的原始值

return 新值

})

(2). 在绑定时，用过滤器过滤变量的原始值

<元素>{{变量 | 过滤器名}}</元素>

(3). 结果: 变量的原始值会先交给过滤器修改，再将修改后的新值放在当前位置显示。

5. 过滤器加参数:

(1). 定义过滤器时:

Vue.filter("过滤器名",function(oldValue, 其它形参, ...){

//oldValue会接住变量的原始值

return 新值

})

(2). 在绑定时，用过滤器过滤变量的原始值

<元素>{{变量 | 过滤器名(实参值,...)}}</元素>

6. 过滤器可连用:

<元素>{{变量 | 过滤器1 | 过滤器2 | ... }}</元素>

注意: 后续的过滤器获得的值不再是原始值，而是前一个过滤器处理过的中间产物。

五. axios:

1. 什么是axios: 基于promise的专门发送ajax请求的第三方函数库。

2. 何时: 在任何位置想发送ajax请求，都可用axios代替其他函数库  
3. 为什么: 发送ajax:

(1). ajax 4步或6步——繁琐

(2). 自己封装ajax函数——功能不够强大

(3). jQuery的$.ajax()——必须引入jQuery才能用！——小题大做

(4). Vue框架曾经包含过vue-resource，也可以发送ajax请求。

(5). 第三方ajax函数库: axios——可独立使用，专门发送ajax请求。

总结: 今后只要在vue中发送ajax请求，都用axios

4. 如何:

(0). 引入axios.js

(1). 配置服务端基础路径:

axios.defaults.baseURL="http://localhost:5050"

(2). 发送get请求:

axios.get(

"接口的相对地址",

{

params:{

参数名: 参数值

}

}

).then(result=>{

//result.data才是服务器返回的结果

})

今后，所有处理响应结果的代码，都要写在.then中。不能写在.then外部。

(3). 发送post请求：

axios.post(

"接口的相对地址",

"uname=dingding&upwd=123456" //不能用对象语法

).then(result=>{

//result.data才是服务器返回的结果

})

问题: post方式不支持使用对象语法传参

解决: 可用qs.min.js模块将对象语法的参数翻译为查询字符串格式:

先引入qs.min.js

再使用Qs对象

axios.post(

"接口的相对地址",

Qs.stringify({

uname:"dangdang",

upwd:"22222"

})

//在运行时，被翻译为"uname=dingding&upwd=22222"

).then(result=>{

//result.data才是服务器返回的结果

})

六. 组件化开发:

1. 什么是组件: 拥有专属的HTML，CSS和JS，以及数据的页面独立区域。

2. 什么是组件化开发: 拿到页面后先划分组件结构。一个页面其实是由多人编写的多个组件拼接起来的。

3. 为什么:

(1). 便于多人分工协作，提高开发效率

(2). 松耦合，如果一个组件出错，不影响其他组件。

(3). 重用

4. 如何创建一个组件:

(1). 创建一个组件:

Vue.component("组件名",{

//跟new Vue()几乎完全一样！组件其实就是一个缩微版的new Vue()对象。只不过监控的范围小

~~el:"#app", //查找要监控的区域的父元素~~

template:`可重用的HTML片段——模板`,

|  |
| --- |
| 强调：组件的template中也必须只有一个唯一的父元素 |

~~data:{ ... }, //页面上所需的变量~~

data(){ //data:function(){

return {

//以前写在data中的所有变量，现在都要写在return中

}

},

methods:{

自定义方法和事件处理函数

},

...

})

(2). 在页面中使用组件: 组件其实是一个可重用的自定义的HTML标签而已。组件名就是标签名。

|  |
| --- |
| 强调: HTML标签名不区分大小写，所以组件名也不区分大小写，所以不要用驼峰命名！如果遇到多个单词组成的组件名，应该使用-来分割每个单词。且每个单词都要小写！ |

(3). 原理: new Vue()在扫描页面时，发现不认识的自定义HTML标签，就会去Vue家查找是否有同名的自定义组件。如果找到，就会用组件的template HTML片段代替页面上自定义标签的位置。并为这个小区域，创建一个组件对象(就是一个全功能的缩微版的new Vue())。

5. 组件化开发: 将来的一个页面都是由多个组件拼接而成的。将来拿到一个页面后，应该先划分组件。再把每个组件创建出来，最后拼接到一个页面上。

6. 今后组件分类:

(1). new Vue() 根组件，负责扫描整个页面，加载其它组件。——一个项目只有一个

(2). Vue.component() 全局组件，可在任何位置随意使用的组件。

(3). 子组件，仅限于在指定父组件内才能使用的组件，今后绝大多数组件都应该是子组件。

a. 将全局组件降级为一个普通的js对象，但是内容结构要和组件的要求保持一致。

子组件对象的命名通常用驼峰命名

b. 在父组件中添加:components:{ 子组件对象名 }

components:{}中的子组件对象名也用驼峰命名

但是，页面上依然使用-命名

作业:

1. 修改xz\_public项目中js/APIs文件夹中的两个函数，用axios代替$.ajax。