## day01

正课：

1. 什么是VUE:

2. 如何使用VUE:

3. MVVM设计模式

1. 什么是VUE:

VUE是渐进式的基于MVVM设计模式的纯前端JS框架

渐进式: 可以在项目中逐步使用VUE的功能，也可以和其它技术混搭。

vs 全家桶: 要使用，就必须使用全套所有技术。且不能喝其它技术混搭。

MVVM设计模式: ?

纯前端: nodejs学的不好和vue无关！

框架: 原生DOM vs jQuery函数库 vs VUE框架

原生DOM: 步骤多， 函数繁琐

jQuery函数库: 步骤没少，每个函数都变的很简单

框架: 彻底简化了步骤，无需人工干预。

什么是框架: 已经是一个半成品的项目，封装了大量重复性劳动。人只要提供个性化定制即可。

何时: VUE只适合于数据操作为主的项目

还是五件事: 增删改查+事件绑定.

2. 如何使用:

下载: 2种方式:

1. 下载并引入vue.js文件: 前2天

2. 用脚手架代码: 最后一天讲

版本: 2.5

开发版: 未压缩版本 , 有完备的错误提示

生产版: 压缩版本 , 删掉了错误提示

3. MVVM设计模式

传统前端划分:

HTML: 专门保存网页内容和结构的文档

CSS: 专门定义网页样式的文档

JS: 为网页添加交互行为

问题: HTML和CSS太蠢了！不会动态变化

一切交互都只能靠JS添加。

导致JS中大量重复的代码和重复的步骤。

MVVM模式:

1. 界面View: 增强版的HTML和CSS，可跟据数据自动变化

2. 模型数据Model:集中存储一个页面内的所有变化的数据。

3. 视图模型ViewModel: 将View界面和Model模型数据包裹起来，统一管理，自动同步修改。

ViewModel中(new Vue()对象)

包含两大子系统:

1. 响应系统: 将模型数据包裹起来，为每个模型变量自动添加get()和set()保镖。

今后只要修改任何模型变量，都自动经过set()，set()中会触发通知: xx变量变为xx新值

通知会发给虚拟DOM树

2. 虚拟DOM树

什么是: 创建new Vue时，通过扫描完整DOM树，仅提取可能变化的元素和属性，组成的一颗及精简的虚拟DOM树。

优点: 1. 查找元素快

2. 封装了重复性的增删改查DOM操作

虚拟DOM树接到通知，快速找到受影响的元素。调用已经封装好的DOM函数，仅更新受影响的元素的受影响的属性。

## day02

正课：

1. 绑定语法:

2. 指令:

3. 双向绑定:

1. 绑定语法: 学名: 插值语法 Interpolation == {{}}

什么是: 让HTML可以自动找到程序中的变量的特殊语法

为什么: 因为传统的HTML是静态的，缺少动态变化的能力。导致js当中要想操作HTML，需要大量重复的代码。

何时: 只要HTML中某个位置的数据，需要根据程序中的一个变量动态变化！都要用绑定语法！

如何: 2步:

1. 先找页面中所有可能发生变化的地方有几处

2. 再在模型数据中定义相同数量的变量:

new Vue({

el:"#app",

data:{

变量名:值,

... : ...,

}

})

强调: HTML中有几处变化，data对象中就要有几个变量与之对应。

3. 在HTML中，可能发生变化的位置用绑定语法定义变量: {{变量或表达式}}

强调: 其实{{}}的用法和模板字符串中${}的用法完全一样！

能写：变量, 算术计算，关系/逻辑运算，函数调用, 访问数组元素, 三目——凡是有返回值的js表达式都能

不能: if else while for——都是程序结构，没有返回值！

结果: 运行时，HTML中的所有{{}}会自动去data中找同名的变量使用。且内存中的data中的变量值发生变化，HTML中的{{}}的值自动变化！用户最终看到的是{{}}中的变量或者表达式计算后的值！而看不见双花括号——节省了大量重复的查找和修改操作。——多亏了MVVM中的ViewModel中的两大子系统: 响应系统和虚拟DOM树。

2. 指令:

什么是: 为HTML添加更多新功能的Vue预置的自定义属性

为什么: 因为原来HTML缺少程序必须的功能: 判断/分支结构，循环。。。功能。所以只能依靠js中反复查找，反复修改来控制HTML元素的内容和状态。

如何: 13种:

1. 绑定属性:

什么是: v-bind属性专门动态绑定元素的属性值

为什么: 要绑定属性值，不能用{{}}，只能用v-bind或:简写

何时: 只要属性值需要根据变量动态变化时，就要用v-bind或:简写

如何: <img v-bind:src="pm25<100?'img/1.png':

pm25<200?'img/2.png':

pm25<300?'img/3.png':

'img/4.png'">

去{{}}换v-bind:

其实可简写: <img ~~v-bind~~:src="...

去{{}}换:

2. 控制元素的显示隐藏:

1. 控制一个元素的显示隐藏:

<ANY v-show="判断条件"

只要条件满足，就显示元素

一旦条件不再满足，就隐藏元素！

2. 控制多个元素，多选一显示:

<ANY v-if="判断条件" 单用时

v-if从现象上看，和v-show是完全一样的

但是原理不一样

鄙视: v-show vs v-if的差别:

v-show 用display:none 控制显示隐藏

v-if 用删除节点的方式，控制显示隐藏

如果一个元素频繁需要显示隐藏,v-show的效率高！

和v-else-if v-else 配合使用:

<元素1 v-if="条件1"  
 <元素2 v-else-if="条件2"

... ...

<元素n v-else

强调: v-if和v-else-if和v-else之间禁止插入一切其他元素。必须连续写！

原理: Vue会自动判断每个条件：

哪个条件符合，就只显示哪个条件的元素。其余元素都不显示。

如果前一个条件已经满足，则后续判断都不再执行。所以，不可能同时显示多个元素。

如果所有条件都不满足，会显示v-else的元素

3. 根据数组反复生成多个相同结构的HTML元素:

其实就是为HTML添加循环功能:

如何:

1. data中必须先定义一个可遍历的数组

2. 在HTML中使用v-for遍历数组，反复生成多个相同结构的元素，并动态绑定元素的内容。

语法:

<要反复生成的元素 v-for="(elem,i) of 数组" :key="i"

强调: 特例: v-for中的循环变量可被v-for自己或子元素用于绑定！

4. 事件绑定:

v-on:事件名="处理函数"

可简写为:

@事件名="处理函数"

强调:

1. 处理函数必须定义在:

new Vue({

el:"#app",

data:{ ... },

methods:{

处理函数(){

this.data中变量

}

}

})中的methods:{}结构中

2. 其实可以传参: @事件名="处理函数(实参值)"

3. 也可以用事件对象e：用法和DOM中完全一样

获得e: methods:{

事件处理函数(e){ ... }

}

后续:

获得目标元素，实现事件委托： e.target

取消冒泡: e.stopPropagation()

阻止默认行为: e.preventDefault();

键盘事件中获得键盘号: e.keyCode

获得鼠标坐标位置: e.screenX, e.screenY

e.clientX, e.clientY

e.offsetX, e.offsetY

5. 避免用户短暂看到{{}}

问题: 如果new Vue加载慢，则可能短暂看到{{}}

解决: v-cloak 可让元素在new Vue加载之前暂时隐藏。当new Vue加载完成后，会自动查找页面中的所有v-cloak斗篷删除该属性，让元素显示出来。

强调: v-cloak没有属性值！

问题: v-cloak空有属性名，没有配套的样式

解决: 只能自己手写！用属性选择器:

[v-cloak]{display:none}

强调: v-cloak不能改名！改名了，vue就找不到了！

其实，除了v-cloak外，还可用v-text代替{{}}绑定元素内容，避免短暂看到{{}}

如何:

<ANY v-text="`js模板字符串语法，用${变量}动态生成内容`"

比如:

<h1 v-text="`姓名: ${uname}`"></h1>

<h2 v-text="`性别: ${sex==1?'男':'女'}`"></h2>

原理: v-text在未绑定时，是一个浏览器不认识的属性，所以，显示不出来。直到new Vue加载完，认出v-text，才用V-text的内容代替元素的innerHTML内容。

v-cloak vs v-text:

v-cloak: 需要额外添加css样式，且需要选择放在哪个元素上，隐藏什么范围内的元素。整个网页都隐藏，用户体验还是不好的。

v-text: 仅对当前元素有效，不会影响其它元素的显示不显示。用户体验比v-cloak要好。但是使用反引号的语法，很晦涩。

6. 绑定HTML片段:

问题: 使用{{}}绑定HTML片段，VUE不会解析HTML片段为网页内容。而是原样显示HTML代码

解决: 今后，只要绑定一段HTML片段，必须用v-html。

如何: <ANY v-html="变量"></ANY>

7. 只在页面加载之初，绑定一次。之后及时数据变化，也不反复更改页面：v-once

强调: v-once没有任何属性值，写在元素中，就起作用

原理:

在构建虚拟DOM树时，会扫描到v-once的元素，所以首次绑定，能绑定v-once的元素的内容。

首次绑定后，v-once的元素会从虚拟DOM树中移除。从此，再出发变量改变时，不会再修改v-once的元素。

8. 万一内容的正文中，有{{}}，但是不是用于绑定的，会出错。

解决: v-pre 可阻止内容中的{{}}被编译，而是让{{}}原样显示。

总结:

1. 如果元素内容要变化:

用 {{变量或表达式}} 绑定

也可用v-text="`变量或表达式`"

2. 如果元素的属性值要变化: 用 :属性名="变量或表达式" 绑定

3. 一个元素控制显示隐藏: v-show="条件"

4. 多个元素选其一显示:

v-if="条件" v-else-if="条件" v-else

5. 反复生成多个相同结构的元素时:

v-for="(elem, i ) of 数组" :key="i"

6. 只要绑定事件处理函数都用: @事件名="处理函数"

7. 只要不希望用户短暂看到{{}}语法:

v-cloak 或 v-text

8. 只要绑定HTML片段: v-html

9. 只要希望只在页面加载之初绑定一次，之后不再变化，就用v-once.

10. 如果内容中包含不想被vue编译的{{}}正文，可用v-pre阻止VUE编译元素的内容。

## day03

正课:

1. 双向绑定:

2. 计算属性:

3. 绑定样式:

4. 自定义指令:

5. 过滤器:

6. axios:

7. new Vue的生命周期

1. 双向绑定:

什么是单向绑定: 只能将程序中的修改更新到页面中。但是无法将页面中的更改，更新回程序中。

只能从Model -> View 不能从View -> Model

比如: {{}} v-text : 等等 都是单向绑定

比如: <input :value="变量"

结果: 变量中的值可正常显示在文本框中

但是文本框内容修改，变量的值不会改变

什么是双向绑定: 即能将程序中的修改更新到页面中，又能将界面中的修改，反向更新会程序中

何时: 只要希望通过表单元素的修改，来更改程序的内容时。——只要绑定表单元素，都用双向绑定。

如何：不要用: ， 要用v-model:属性名="变量"

原理: v-model其实就是自动为表单元素绑定了oninput或onchange事件。在这些内容更改事件中，自动修改当前Vue对象中的模型变量。

普通绑定为什么不能更新程序中的数据：因为没有自动添加事件，导致界面上更改，Vue框架不知道，自然就无法更近模型变量。

其它表单元素的双向绑定:

1. 单选按钮radio: 要绑定checked

不绑定value，因为radio中的value是写死的，固定不变的。等着被选择。

绑定checked: 因为是否选中radio，取决于checked属性的值。用户更改选中项，改的也是checked属性值。——radio只能绑定v-model:checked="变量"

绑定时:

<input type="radio" name="sex" value="1" v-model:checked="变量"

<input type="radio" name="sex" value="0" v-model:checked="变量" ~~"变量==值"~~

小脾气: v-model在radio中：

从程序绑定到页面时: 会自动用checked后绑定的变量值和当前radio写死的value值做比较，返回bool值决定当前radio是否选中。时不用自己在checked后写完整条件！只写一个变量名即可！

改变选中状态时，v-model会将选中的radio的写死的value值更新回程序中绑定的变量上

2. 单个checkbox绑定: 也绑定checked属性

<input type="checkbox" v-model:checked="isAgree"

3. select元素绑定: 绑定的是select元素整体的value属性

因为，无论选择了哪一个option，当前选中项的值都 变成select元素的value属性值。

如何: <select v-model:value="变量">  
 <option value="值1">

<option value="值2">

... ...

原理:

当从Model->View绑定时: 用变量的值和每个option的value做比较，按个相等，就选中哪一个  
 当修改select下的选中项后，将当前选中的option的value值更新回变量去

其实，以上绑定都有一个共同的简写: 只写v-model即可，可省略:value :checked等。v-model会个识别元素中起决定作用的可变的属性，自动绑定。

watch监控机制:

什么是: Vue中的watch监控机制可监控vue所有模型变量的变化。只要模型变量变化，就自动触发一个同名的函数。在函数中可调试变量的值或执行实时操作

何时: 希望只要变量的值变化，就立刻做一件事儿时

如何: new Vue({

el:"#app", //找要监控的整个页面空间

data:{

变量1:值1

}, //所有页面需要的模型变量

methods:{ ... }, //所有事件处理函数和一些自定义函数

created(){ ... }, //当new Vue()对象加载完自动执行

watch:{ //定义监控变量的函数

变量1(){//必须同名，只有同名的函数，才能监视同名的变量

执行一切js或vue的操作

}

}

})

2. 计算属性:

什么是: 不实际属性值，绑定时，动态通过其他现有属性，计算出自己的属性值。

何时: 页面上需要一个值，但是，这个值数据库中没有直接保存，而是需要动态计算出来。

如何: new Vue({

el:"#app",

data:{ },

methods:{ },

created(){ },

watch:{ },

computed:{

属性名(){

//通过其他变量计算获得最后结果

return 结果

}

}

})

使用计算属性：计算属性虽然定义成函数，但是，用法和普通模型变量完全一样！不要加()

特点:

1. 只要依赖的其它变量变化，计算属性都会自动重新计算

2. 一个属性如果多个位置使用，其实只计算一次。

原理: 计算属性的值，每计算一次，结果都会被Vue框架缓存起来，反复使用。除非依赖的变量发生改变时，才重新计算，然后再重新缓存，反复使用。

vs 函数: 也可以实现计算的效果

问题: 不会被缓存！每用一次就重复计算一次！

总结: 如果只关心计算结果时，首选计算属性

如果不关心结果，甚至没有返回值，只关心执行的过程时，首选函数。

3. 绑定样式:

2种:

1. 动态绑定单个css属性:

2种:

1. 用字符串方式，绑定整个style属性：

<ANY :style="变量"  
 data:{

变量:"top:50px; left:50px"

}

缺点: 在字符串中不便于修改某一个css属性

2. 都用对象方式，绑定style:

<ANY :style="变量"

data:{

变量:{

top:"50px", //必须px

left:"50px" //必须px

}

//绑定时，被自动翻译为字符串:

//"top:50px; left:50px"

}

强调: 固定的style和动态绑定的style可同时存在

比如：

<div style="position:fixed" :style="divStyle">

data:{

divStyle:{

top:"50px", //必须px

left:"50px" //必须px

}

}

运行时，结果: 动态绑定的style会和写死的style合并为一个style:

<div style="position:fixed; top:50px; left:50px"

2. 动态绑定class

2种:

1. 将整个class属性当做一个字符串绑定

问题: 不便于控制一个class的启用或禁用

2. 今后也要使用对象格式绑定:

<b class="固定不变的class" :class="变量名"

data:{

变量名:{

class1: true/false,

class2:true/false,

... ...

}

}

运行时:

1. 将变量中值为true的class转为字符串:

比如: { down:true, fade:true }

-> "down fade"

{down:false, fade:true}

-> "fade"

2. 用翻译好的class字符串和元素上规定的class=""，合并为一个class

如果想用程序切换使用/不使用某个样式类，只要修改类名对应的true/false就行

值为true，就是应用该样式类

值为false，就是不应用该样式类

4. 自定义指令:

何时: 当希望元素自动执行一件事，但是vue中的指令不够用，就可以自定义指令。

如何:

1. 定义一个新的自定义指令:

Vue.directive("指令名",{//指令名不应该带v-前缀

//会在当前元素加载后，自动执行

inserted(elem){

//elem: 接住的是当前指令所在的DOM元素

//可对elem执行任何原生DOM的函数

}

})

强调: 应该定义在不属于任何new Vue()的外部全局

将来所有new Vue()都可以共用这个自定义指令。

2. 使用指令: <ANY v-指令名

但是，使用指令时，必须加v-前缀

原理: 当<ANY>这个元素在HTML中加载完之后，自动去调用指令中携带的inserted()函数

5. 过滤器:

什么是: 将不能直接使用的变量值，过滤成可以给人看的新值的函数

何时: 只要变量的原始值不是想要的，都可用过滤器过滤成想要的值给人看。

如何:

1. 定义过滤器:

Vue.filter("过滤器名称",function(oldVal){

return 新值

})

2. 使用过滤器:

在绑定语法中都可用"|"将过滤器连接在变量之后

运行时: 原始变量值->过滤器->oldVal参数->return过滤后的新值->给人看

过滤器参数:

何时: 当根据不同值，选择过滤成不同的新结果时

如何: 2步:

1. 定义过滤器时添加形参:

function(oldVal,lang="cn"){

2. 使用过滤器时传入实参:

{{sex2|sexFilter("en")}}

强调: 定义时添加过滤器形参，从第二个形参开始添加。但是使用过滤器时，自定义实参值，却从第一个实参传入。但是还不会出错！因为过滤器函数在调用时，自动回将原始值作为第一个实参传入。我们添加的自定义实参，只能排在第二。

其实程序中的过滤器也可以用|串联多个:

{{变量|过滤器1|过滤器2|... }}

坑：定义后一个过滤器时，拿到的值不是原始值，而是上一个过滤过滤后的中间产物。

6. axios

什么是: 专门发送ajax请求的基于Promise的函数库

为什么:

1. 然哥四步/五步:

2. 可以自己封装

3. jQuery库中的$.ajax()

将来在Vue项目中，几乎不会用jQuery

如果仅仅为了发ajax请求而引入jQuery函数库，特别小题大做。

4. Vue官方有一个vue.resource组件

但已停止维护

5. axios: Vue官方推荐的最好的ajax函数库。

何时: 今后只要在任何位置想发ajax请求，都可用axios

如何:

1. 引入axios.js

2. 调用axios.get()函数发送请求

axios.get("url",{

params:{//相当于$.ajax中的data:{ ... }

参数1:值1,

... : ...

}

//运行时params中的参数会被翻译为字符串，拼接到url的结尾:

"url?参数1=值1&参数2=值2&..."

}).then( result=> {

为this保持和外部VUe一致，必须用箭头函数

result返回的不只是服务端响应结果

真正的响应结果包含在result.data中

})

7. new Vue的生命周期:

什么是: 一个new Vue()的加载过程:

何时: 如果希望在加载过程某个阶段，自动执行一项任务时。

如何: new Vue()的声明周期包括4个阶段，8个函数

钩子函数——其实就是在不同阶段自动执行的回调函数

前: beforeCreate(){ ... }

1. 创建(create)阶段: 创建new Vue()对象和data()对象

已经有data对象了，但是没有虚拟DOM树

可以发送ajax请求

因为没有虚拟DOM树，所以暂时不能用DOM操作

后: created(){ ... }

前: beforeMount() { ... }

2. 挂载(mount)阶段: 创建虚拟DOM树，将data中的变量值开始向DOM树上绑定

即有data对象，又有虚拟DOM树

即可发送ajax请求，又可执行DOM操作

后: mounted(){ ... }

//后两个阶段不是必须:

前: beforeUpdate(){ ... }

3. 更新(update)阶段: 当data中的变量值被改变时才触发

后: updated(){ ...}

前: beforeDestroy(){ ... }

4. 销毁(destroy)阶段: 当调用专门的$destroy函数销毁一个组件时，才触发

后: destroyed(){ ... }

## day04

正课：

1. 组件:

2. 组件化开发:

3. SPA:

4. 脚手架:

1. 组件component:

什么是: 拥有专属的HTML,CSS,JS和数据的页面独立区域

为什么: 重用！

何时: 只要发现页面中有一个区域可能被反复使用，都要定义成组件。

如何:

1. 创建: Vue中的组件其实是一个包含html,数据和功能的对象

每个组件都是一个缩微的new Vue()

new Vue()中有什么组件中就有什么

Vue.component("组件名",{

~~el:"选择器",~~

template: //将来会反复复制多个和这里一模一样的html片段，所以将这一份标准的的片段称为模板

` <div> //组件模板必须用唯一的父元素包裹  
 <button>-</button> <span>1</span> <button>+</button>  
 </div>`,

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

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

return {

//模型变量

}

},

methods:{ ... }, //同new Vue()

created(){ ... }, //同new Vue()

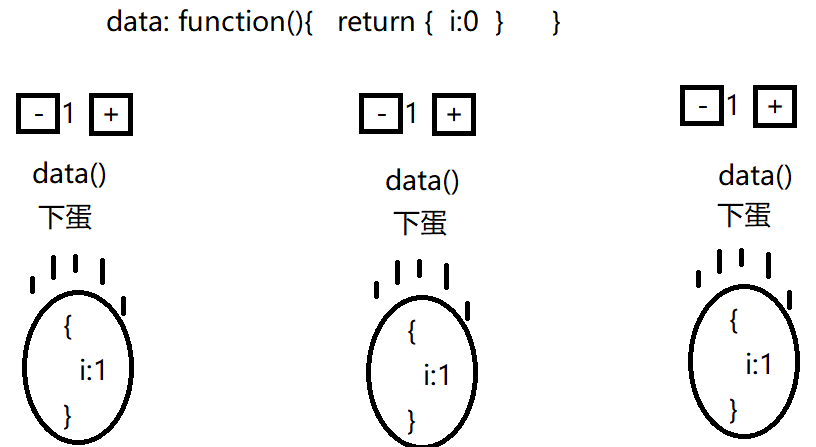
watch:{ ... }, //同new Vue()

computed:{ ... }, //同new Vue()

mounted(){ ... } //同new Vue()

})

为什么: data必须是function:



2. 使用组件: 组件在html中其实就是一个可重用的标签而已。组件名就是标签名。所以组件名起名不要用驼峰命名！因为HTML标签不区分大小写。可用-分隔多个单词

new Vue()要不要写？答：必须写！

因为是全靠new Vue()扫描页面时，发现不认识的标签，才会去加载自定义的组件。

加载时: 会用组件的template中的HTML模板片段，代替不认识的标签。

2. 组件化开发:

什么是: 将所有要做的HTML页面，切割为多个独立的组件，可以分工协作开发。

为什么: 1. 分工协作；2. 松耦合

何时: 今后做页面，都是先将页面划分成多个组件，再分别开发每个组件。

如何:

1. 将页面划分成多个功能区域-组件

2. 创建组件.js，其中定义组件对象：2种:

1. 全局组件: Vue.component()

什么是: 可以在页面任何位置使用的组件

何时: 只要页面中一个区域，可能在多个页面或者多个不同位置随意使用时

问题: 不受约束，随处可用

2. 子组件:

什么是: 必须在指定父组件内才能使用的组件

何时: 如果希望限制一个组件只能在其父组件内使用时，就必须定义子组件

如何:

1. 创建子组件:

var 组件对象名={ 和组件标准写法必须一致 }

强调:

1. {}中必须和Vue.component()中的{}内容格式一致！。

2. 变量名必须用驼峰命名

2. 在父组件中用components属性包裹所有子组件:

Vue.component("父组件",{

template:

data:

methods:

...

components:{ 子组件对象名,... }

})

3. 在主界面或父组件中，使用子组件标签，引入子组件的内容

结果: 浏览时，所有组件(父和子)，最终都合并为一个整体。不再包含不认识的标签。



问题: 父组件中的公共数据，子组件可以直接使用吗？

答: 不行！因为每个组件都是专属的独立的数据。互相之间没有关系。即使是在位置上是父子关系的组件之间，也没有数据的关系。

问题：如何让子组件使用父组件中的公共数据:

答: 父组件可以将自己的模型变量赋值给子组件的属性

2 步

1. 子组件定义自定义属性，准备接受父组件的数据

var 子组件={

template:

data(){ return { } },

props:[ "自定义属性名" , "自定义属性名"],

methods:{ ... },

...

}

结果: 当前组件将来变成标签时，在开始标签中就多了一个同名的自定义属性。只不过暂时没有值

<子组件 自定义属性=""

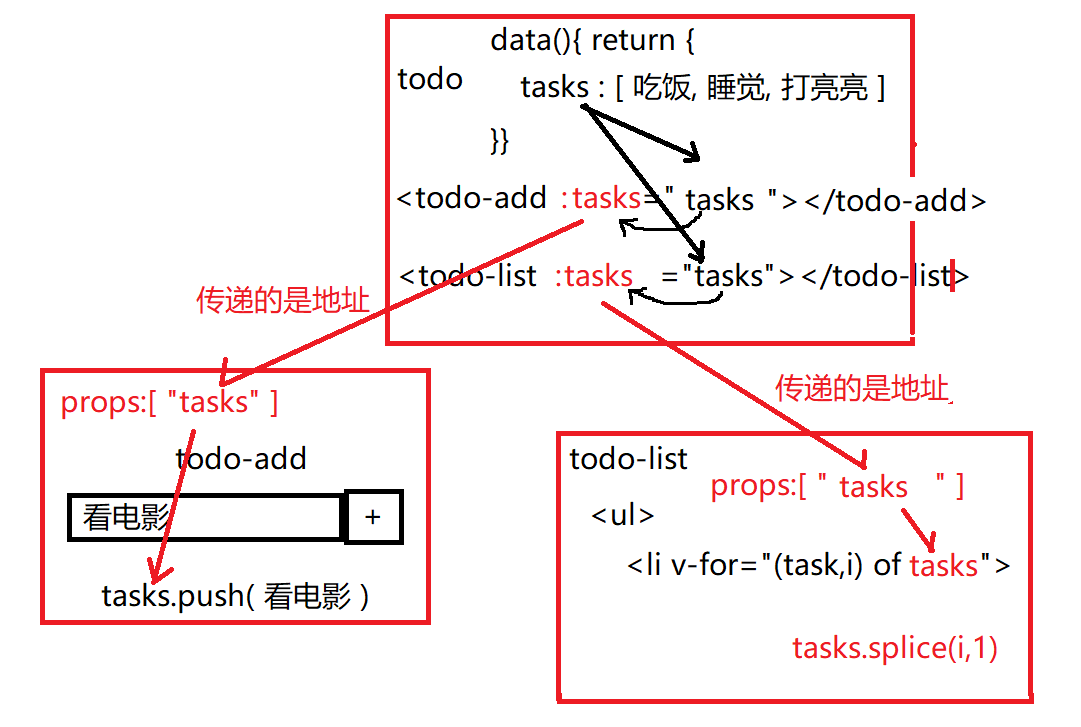
2. 父组件中使用子组件标签的位置，开始标签中通过:绑定的方式，将自己data中的值赋值给子组件自定义的属性

<子组件 :自定义属性="父组件的变量"

结果子组件对象内的自定义属性，就获得了父组件变量的值。

强调: props中的自定义属性等效于data中的模型变量。也可以在子组件内绑定使用。

强调: 子组件中可以修改父组件传来的数据。如果传来的是引用类型的对象: 数组或对象，则在子元素中修改变量等效于直接修改父元素中的原对象或数组。因为父子组件间传递引用类型的对象，实际传递的都是地址值。



3. SPA: Single Page Application

单 页面 应用

什么是: 整个应用程序只有一个完整的页面

为什么: vs 多页面

|  |  |  |
| --- | --- | --- |
|  | 多页面 | 单页面 |
| 页面个数 | 多个页面 | 只有一个页面 |
| 页面跳转 | 每跳转一个新页面，都重新请求服务器，重新下载页面文件，重建整棵DOM树 | 并不是真正的页面跳转，而是选择对应的组件模板代替当前页面中指定的占位符元素。不用重新请求服务器，  不用重复下载页面文件，不用重建整课DOM树，仅替换局部 |
| 共享资源  bootstrap.js  jquery.js | 多页面应用，每跳转一次新页面，都要重新请求和下载共享的资源文件。 | 每切换一个组件，实现跳转页面，不需要重建整个网页，所以不需要重复请求共享资源。 |
| 首次加载速度 | 多页面应用每次只需要请求自己需要的内容下载回来即可。 | 单页面应用需要在首次就将所有组件的模板下载下来。首屏加载会慢。——公司中已解决: 配合服务端实现异步延迟加载。 |
| 翻页动画 | 多页面应用不可能实现页面切换动画效果 | 单页面应用，因为所有的组件已经在客户端。所以完全可以做出页面切换动画效果。 |

何时: 今后，基本上都是单页面应用。

如何:

1. 先有一个完整的home.html文件

home.html中主体部分被用特殊标签占位，暂时为空

2. 再为每个"页面"，定义组件对象

比如: index.js details.js products.js login.js

3. 创建路由地址列表  
 什么是: 保存每个路由地址和组件对象对应关系的列表

如何:

var routes=[

{ path:"/index", component: index },

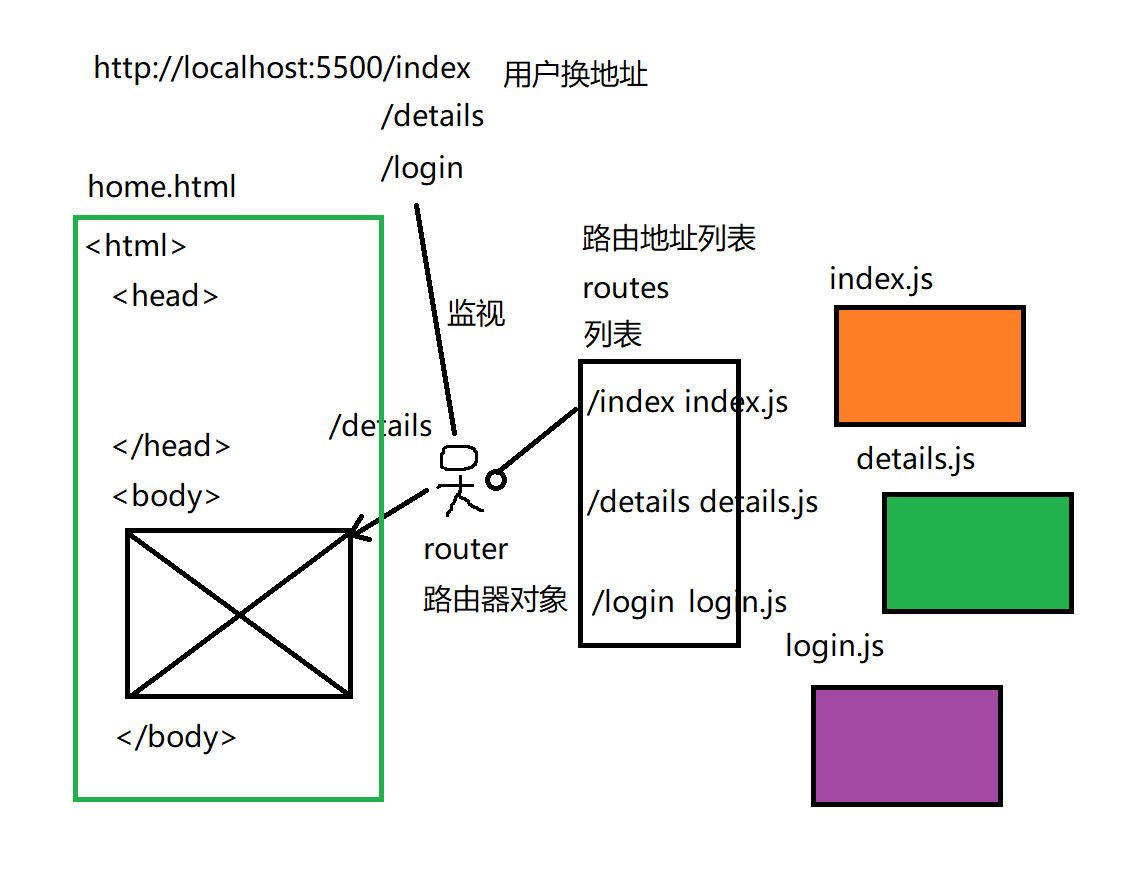
{ path:"/details", component: details },

{ path:"/login", component: login },

]

4. 创建一个路由器对象：

什么是: 专门监视地址栏中的变化，根据变化，在路由字典中查找对应的组件，用组件替换页面中占位符的位置。实现页面切换。



4. 脚手架:

什么是: 已经具备核心功能的半成品项目

为什么: 标准化项目的开发，便于分工协作。

何时: 今后只要使用Vue框架做开发，都要先创建Vue框架的脚手架项目代码。

如何: 2步:

1. 先安装一个能创建脚手架项目的命令行工具:

npm i -g @vue/cli

2. 用vue命令，创建一个项目的脚手架代码文件夹

vue create 项目文件夹名

第一步: Your connection to the default npm registry seems to be slow. Use https://registry.npm.taobao.org for faster installation? (Y/n) Y

第二步: ? Please pick a preset: (Use arrow keys)

> default (babel, eslint)

> Manually select features //选下边一个

第三步: ? Check the features needed for your project: (Press <space> to select, <a> to toggle all, <i> to invert selection)

>(\*) Babel //必选: 翻译

脚手架代码中采用了模块化开发：

抛出模块: ~~module.exports={ ... } nodejs中~~

ES6: export default { ... }

引入模块: ~~var 变量=require("路径") nodejs~~

ES6: import 变量 from "路径"

因为不是所有浏览器都认识ES6，所以，需要Babel最后将所有ES6的代码翻译为多数浏览器都认识的ES5代码。

( ) TypeScript //微软出的严格的script——不选

( ) Progressive Web App (PWA) Support

(\*) Router //必选

(\*) Vuex //Vue中的状态机制。当多个组件需要共用一些数据：比如当前登录用的用户名。这个公共的数据就要放在Vuex中——下个阶段老师将

( ) CSS Pre-processors

() Linter / Formatter //千万不要选！是代码质量检查工具。即使代码没错，格式不规范，也报错！

( ) Unit Testing

( ) E2E Testing

第四步: Use history mode for router? (Requires proper server setup for index fallback in production) (Y/n) n

因为必须要服务端支持，所以才不选。

默认Vue的地址栏中都是用#/路径来跳转

但是#可能和你的锚点地址冲突。

如果即想用vue，又想要锚点地址，只能启用history模式。启用后，地址栏中就没有#了。而是直接用"/路径"。

第五步: Where do you prefer placing config for Babel, PostCSS, ESLint, etc.? (Use arrow keys)

> In dedicated config files

>In package.json //将所有的配置都放在一个package.json中。

第六步: Save this as a preset for future projects? (y/N) N 不把这次的项目配置保存为今后项目的模板。

脚手架文件夹创建成功后:

Successfully created project xz\_vue.

👉 Get started with the following commands:

$ cd xz\_vue //先进入项目文件夹（vue create后的那个文件夹——xz\_vue）

$ npm run serve //=运行Live server+Babel的效果。是先将脚手架中浏览器不懂的代码，翻译为浏览器能懂的代码。然后，再将翻译后的页面运行在一个临时的开发服务器中。 强调: npm run serve 必须在项目文件夹内运行

保险起见是在vs code中打开xz\_vue，然后在package.json文件上右键单击在命令行打开，然后运行:

npm run serve 没有r结尾

2件事:

1. 启动开发服务器: http://localhost:8080/

2. 编译项目代码为浏览器认识的: Compile

强调:

1. 不用总是npm run serve，因为npm run serve可以自动感知代码的修改，并自动重新编译，重新运行。

2. 服务端nodejs的cors: 要改为http://localhost:8080/。如果不想总是改。可改为cors:\*。允许所有客户端都能跨域。

分析脚手架文件夹的结构：

public/ 将来所有不需要编译，就可直接使用的js和css，以及图片都放在public下。放在public下的文件，都不会被再次编译。比如: imgs文件夹，唯一的完整的首页index.html，bootstrap.css，jquery-3.2.1.js, bootstrap.js

放在public下的不需要编译的css和js，都要在唯一完整的页面index.html中引入。

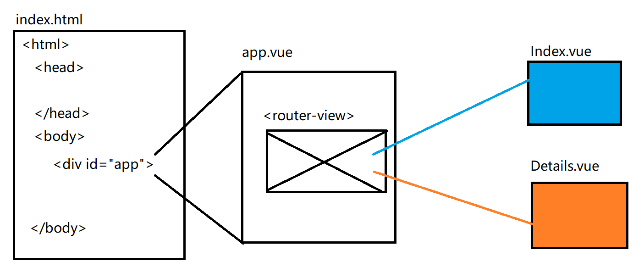
src/ 放的是所有自己写的代码

/assets 放自己编写的公共的css和js

/components 放全局组件或多个页面共用的子组件。比如: 页头组件，页脚组件，广告轮播组件...

/views 放所有的页面组件，有几个页面，就在views下创建几个组件文件。比如: 首页，详情页，商品列表页，登录页

app.vue 整个网站所有页面公共的容器组件



main.js 相当于以前的new Vue()。整个网站只有一个new Vue()。且main.js还承担了配置Vue项目的责任

比如: axios:

1. 用npm i -save axios 为脚手架项目添加axios模块。

2. 希望在Vue框架中所有组件里都能用axios。所以，应该讲axios添加到Vue整个类型的原型对象中。

2步: main.js

1. 用import语法引入axios模块

import axios from "axios"

2. 将axios对象强行添加到Vue的原型对象中: Vue.prototype.axios=axios;

结果: 所有组件中都可用this.axios.get()发送ajax请求。

router.js 是整个网站的路由器和路由字典，保存着所有路径和组件的对应关系。

.vue文件: 是脚手架专用的组件/页面文件。

今后，只要新建一个组件，或新建一个页面，都新建.vue文件

内容包括: 3大部分:

1. 模板片段: <template></template>中

要求: 唯一父元素包裹

2. 组件的css: <style scoped></style>

3. 组件的js代码:

<script>

//组件本质就是一个对象，对象需要抛出到外部被别人使用

export default {

//没有template和el了

data(){

return {

}

},

created(){

},

methods:{

},

computed:{

},

watch:{

}

}

</script>

路由跳转:

Vue中不能用a跳转:

应该用<router-link to="/路径"></router-link>

在运行时被翻译为: <a href="#/路径">