Vue全家桶入门和企业级项目开发

# 一、Vue的快速感知

## 1.1 使用src引包的方式感知一下vue.js

官网：https://cn.vuejs.org/

logo ： 

我们可以去src引入vue（而不是用webpack进行CMD规范构建）来快速认知Vue框架。

**事实上，建议大家先用引包的方式学习一些基本的语法、事件绑定、双向数据绑定，然后上webpack。**

这个js代码的地址： https://cn.vuejs.org/js/vue.js

创建一个index.html页面，用src直接引入这个vue.js文件：

<!DOCTYPE html>

<html lang="en">

<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>

</head>

<body>

<div id="app">

<h1>{{a}}</h1>

</div>

<script src="https://cn.vuejs.org/js/vue.js"></script>

<script>

new Vue({

el : "#app" ,

data : {

a : 100

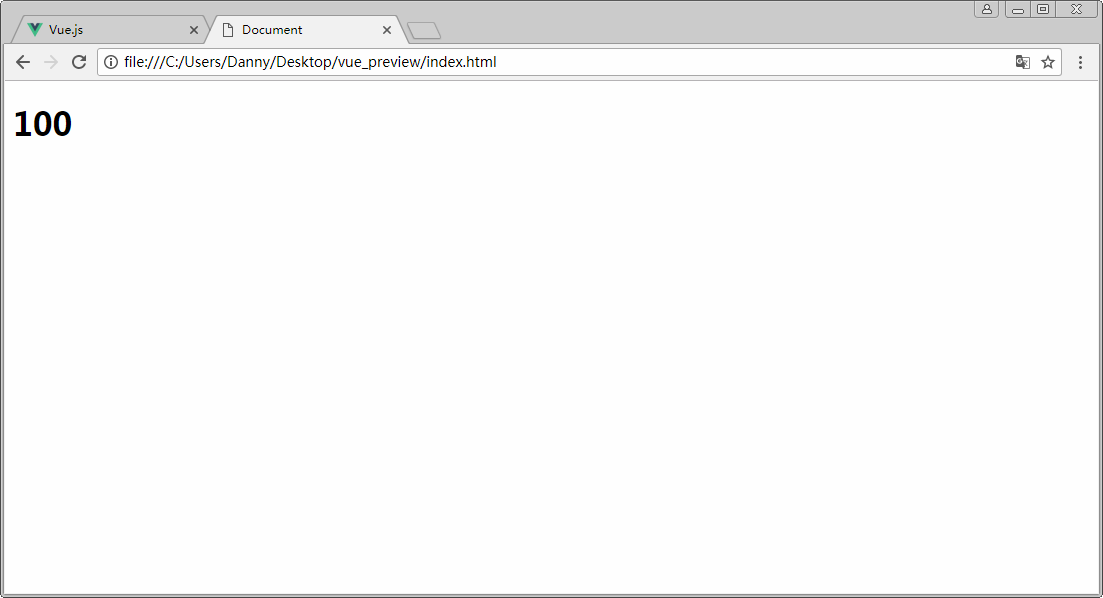
}

});

</script>

</body>

</html>



首先我们认知的一个React和Vue的区别：**React是在写类，Vue是在写实例**。

写实例要由于写类，因为不用经常.bind(this)了。

进行一下注释：

<script>

new Vue({

el : "#app" ,  **//挂载点，在这个div中是vue生效的地方，只能有唯一的挂载点。**

data : { **//数据，不是state**

a : 100

}

});

</script>

MVVM框架讲究的是：**一切的视图都是和数据相关的，数据变化，视图就要变化**。

vue中使用用双大括号进行插值：

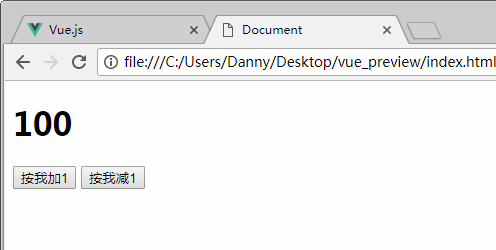
<div id="app">

<h1>**{{a}}**</h1>

</div>

## 1.2 vue的视图更新原理

我们创建了两个按钮：



<!DOCTYPE html>

<html lang="en">

<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>

</head>

<body>

<div id="app">

<h1>{{a}}</h1>

<button **v-on:click="add"**>按我加1</button>

<button **v-on:click="minus"**>按我减1</button>

</div>

<script src="vue.js"></script>

<script>

new Vue({

el : "#app" ,

data : {

a : 100

},

**methods : {**

**add(){**

**this.a ++;**

**},**

**minus(){**

**this.a--;**

**}**

**}**

});

</script>

</body>

</html>

一切的细节我们一会在“二”会讲解。

我们发现语句

this.a++;

不同于react中的

this.setState({"a":this.state.a + 1});

这是vue的最大特色。

因为vue做数据和视图的相关变化，机理是**数据劫持**。**Object.defineProperties()，定义vue组件对象中的data中的所有的变量的setter，让他们在被设置值的时候，刷新视图**。

视图数据劫持的机理：

<script>

var obj = {

a : 100

}

Object.defineProperties(obj, {

"a" : {

set(){

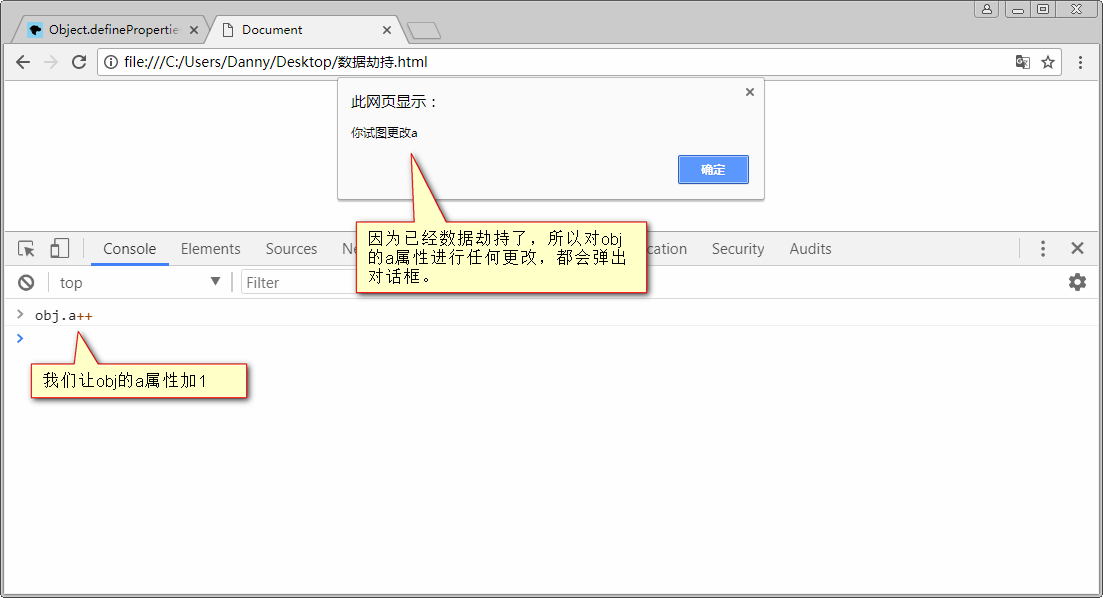
alert("你试图更改a");

}

}

});

</script>



更进一步的做实验：

将vue这个挂载的组件用一个变量来指向：

<script>

var myel = new Vue({

el : "#app" ,

data : {

a : 100

},

methods : {

add(){

this.a ++;

},

minus(){

this.a--;

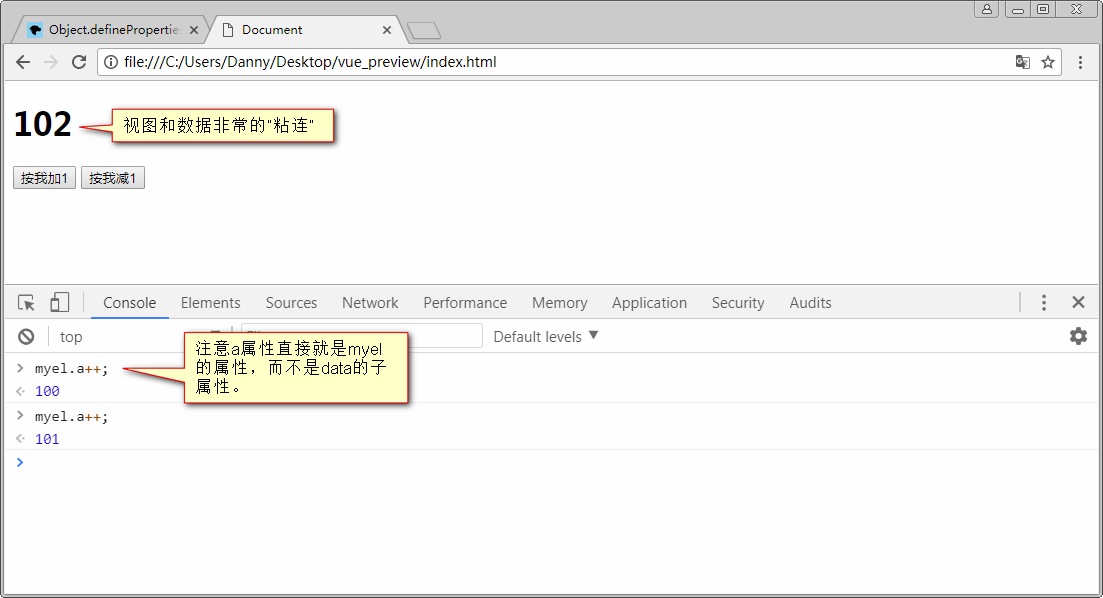
}

}

});

</script>

此时甚至可以在控制台进行数据的变化，视图也会乖乖更新。



我们做一个数组，点击按钮可以删除一项，你会发现vue的实现非常简单：

非常简单干脆的对数组进行一个pop()就可以了。

<!DOCTYPE html>

<html lang="en">

<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>

</head>

<body>

<div id="app">

<h1>{{a}}</h1>

<button v-on:click="add">按我加1</button>

<button v-on:click="minus">按我减1</button>

<br>

<button v-on:click="delitem">按我去掉一项</button>

<ul>

<li v-for="item in arr">{{item}}</li>

</ul>

</div>

<script src="vue.js"></script>

<script>

var myel = new Vue({

el : "#app" ,

data : {

a : 100 ,

**arr : ["白板","幺鸡","二条","三万"]**

},

methods : {

add(){

this.a ++;

},

minus(){

this.a--;

},

delitem(){

**this.arr.pop();**

}

}

});

</script>

</body>

</html>

但是react中很难：

this.setState({

arr : this.state.arr.filter((item,index)=>{

return index != this.state.arr.length - 1;

})

});

react中经常会出现这样的画风：

this.setState({

"a" : {

**...this.state.a ,**

"b" : {

**...this.state.a.b ,**

"c" : {

}

}

}

});

之所以出现这样的画风，是因为react本质上state是不能直接改变的，this.state.a++是坚决错误的语句。必须setState()一个属性的完整样貌。但是vue中，this.a++是非常正确的语句

## 1.3 vue的指令系统

Angular在2009年诞生的，发明了指令(directive)。

<div **ng-repeat=""**></div>

Vue采用了指令系统：

<li **v-for="item in arr"**>{{item}}</li>

这个指令要比jsx中的这个要简单的多：

{

this.state.arr.map(item=>{

return <li></li>

})

}

vue中所有的指令都是v-开头的。

**指令系统的原理就是正则表达式，所有指令的值都是引号包裹的（引号是正则表达式工作的定界符），正则表达式会进行值的提取，然后执行。**

所有v-开头的这些指令，**后面的值一定是引号包裹的（单双引号都行），但不要认为是字符串，里面的内容是活的**，



接下来就要起飞了！

# 二、学习Vue的基本语法和使用

我们这部分仍然用src引包方式学习

## 2.1 基本的vue实例的基本结构

在实例化vue的时候（任何一个项目，只实例化vue一次）要传入一些参数。

el：挂载点

data:数据

computed : 计算后的数据

methods : 方法

<!DOCTYPE html>

<html lang="en">

<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>

</head>

<body>

<div id="app">

<h1>{{a}}</h1>

<button v-on:click="add">+</button>

<h1>数字的平方：{{pingfang}}</h1>

</div>

<script src="vue.js"></script>

<script>

//一个项目中，只会实例化vue一次

new Vue({

**el** : "#app" ,

**data** : {

a : 100

},

**methods** : {

add(){

this.a ++;

}

},

**computed** : {

pingfang(){

return this.a \* this.a

}

}

});

</script>

</body>

</html>

我们具体来看computed计算后的值，重点它是一个值，定义的时候是用函数来定义，return的值就是它的值。



但是，不要认为所有和data相关的值都要写在computed中，我们的methods中也能写：



## 2.2 插值-模板语法

首先要知道的是{{}}和v-text指令是完全等价的，事实上，在{{}}运行的时候机理就是换为v-text，用v-text指令的处理器来进行渲染。

<div id="app">

<h1>{{a}}</h1>

<h1 v-text="a"></h1>

</div>

双大括号插值插入的是纯文本，dom的纯文本不会渲染：

<p>

{{b}}

</p>

new Vue({

el : "#app" ,

data : {

a : 1 ,

b : "<input type='text' />"

}

});



为了识别为dom，此时用：

<p **v-html="b"**></p>

{{}}中能够有：

* 简单运算
* 函数的调用
* 三元运算符
* .filter、.reduce等函数
* Math对象、Date对象、Number构造函数

不能有：

* if语句
* for循环

比如正确的：

<p>

我爱你{{5000 \* 2}}年

</p>

<p>

我爱你{{nian()}}年

</p>

<p>

{{[2,3,4,5,6].reduce((a,b)=>a+b)}}

</p>

<p>

{{Math.random()}}

</p>

<p>

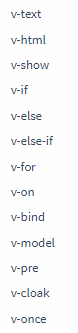
{{c ? "哈哈" : "嘻嘻"}}

</p>

# 三、指令

v-开头的属性叫做指令，指令的值必须是引号包裹的。

能够用哪些指令：https://cn.vuejs.org/v2/api/



## 3.1 v-show和v-if和v-else和v-else-if

v-show和v-if都是接受布尔值，

**v-show机理是显示和隐藏，v-if是上树与下树。**

**v-show的效率高，v-if能触发子组件的生命周期**

<div id="app">

<button v-on:click="changeBox1">显示/隐藏</button>

<div class="box" v-show="isShowbox1"></div>

<br>

<button v-on:click="changeBox2">上树/下树</button>

<div class="box" v-if="isShowbox2"></div>

</div>

v-else表达是的取这个值的反面，一般紧跟携带v-if的标签。v-else不需要有值的。

<div class="box" **v-if="isShowbox2"**></div>

<div **v-else**>盒子被隐藏了</div>

|  |  |
| --- | --- |
|  |  |

一般v-else有点不习惯，喜欢写!：

<div **v-if="!isShowbox2"**>盒子被隐藏了</div>

v-else-if暗含“跳楼现象”，已经暗含不满足上面的v-if了。

<div **v-if="a < 10"**>这个数字小于10</div>

<div **v-else-if="a < 20"**>这个数字在10~20之间</div>

<div **v-else**>这个数字大于等20</div>

|  |  |  |
| --- | --- | --- |
|  |  |  |

## 3.2 v-for

用来循环某一个元素，数据必须是数组或者类数组对象！

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Document</title>

</head>

<body>

<div id="app">

<ul>

<li **v-for="item in arr"**>{{item}}</li>

</ul>

</div>

<script src="vue.js"></script>

<script>

//一个项目中，只会实例化vue一次

new Vue({

el : "#app" ,

**data : {**

**arr : ["A","B","C","D","E"]**

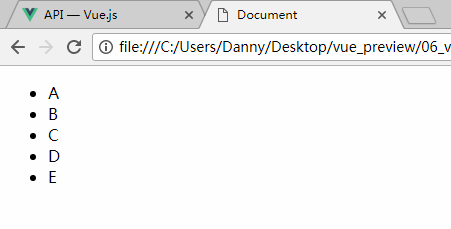
**}**

});

</script>

</body>

</html>



如果数组的项都是对象，也行：

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Document</title>

</head>

<body>

<div id="app">

<ul>

<li v-for="item in arr">{{item}}</li>

</ul>

<ul>

<li **v-for="item in arr2"**>**{{item.name}}**</li>

</ul>

</div>

<script src="vue.js"></script>

<script>

//一个项目中，只会实例化vue一次

new Vue({

el : "#app" ,

data : {

arr : ["A","B","C","D","E"] ,

**arr2 : [**

**{"id" : 1 , "name" : "小明" ,"sex" : "男"},**

**{"id" : 2 , "name" : "小红" ,"sex" : "女"},**

**{"id" : 3 , "name" : "小强" ,"sex" : "男"},**

**{"id" : 4 , "name" : "小青" ,"sex" : "女"}**

**]**

}

});

</script>

</body>

</html>

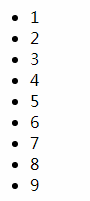
常数的循环：

<ul>

<li **v-for="item in 9"**>**{{item}}**</li>

</ul>

从1开始：

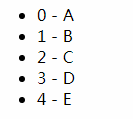


用逗号隔开一个参数，表示下标序号：

<ul>

<li v-for="(item **, index**) in arr">**{{index}} - {{item}}**</li>

</ul>



对象也可以被遍历，参数顺序：v、k、i。不是k、v、i。

比如对象：

obj : {

"hobby" : "羽毛球",

"height" : 178,

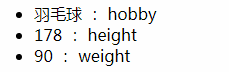
"weight" : 90

}

<ul>

**<li v-for="(v , k ,i) in obj">{{k}} ： {{v}}</li>**

</ul>



v-for 默认行为试着不改变整体，而是替换元素。迫使其重新排序的元素，你需要提供一个 key 的特殊属性：

<ul>

<li v-for="item in arr2" **v-bind:key="item.id"**>{{item.name}}</li>

</ul>

效率上的增加，强制重排元素。

## 3.3 九九乘法表

非常简单，甚至没有data，没有methods。

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Document</title>

</head>

<body>

<div id="app">

<table>

**<tr v-for="i in 9">**

**<td v-for="j in 9">**

**{{i}} \* {{j}} = {{i \* j}}**

**</td>**

**</tr>**

</table>

</div>

<script src="vue.js"></script>

<script>

//一个项目中，只会实例化vue一次

new Vue({

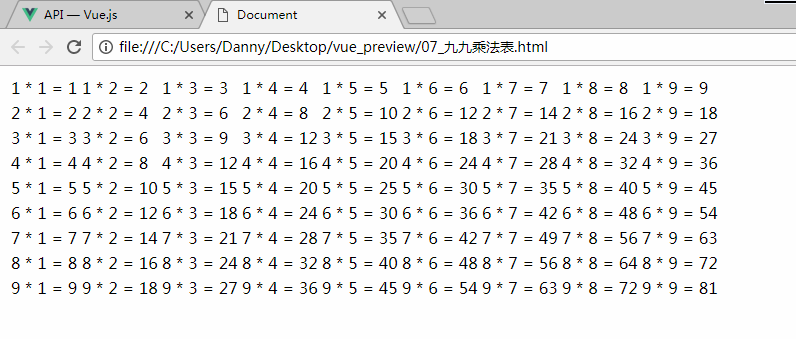
el : "#app"

});

</script>

</body>

</html>



react中如果要重复DOM元素，方法是数组的自动展开

[<td></td> , <td></td> , <td></td> , <td></td>]

然后用

{

tds.map(item=>{

})

}

vue确实牛逼

## 3.4 v-on

表示添加一个事件监听。

<button **v-on:click="add"**>按我</button>

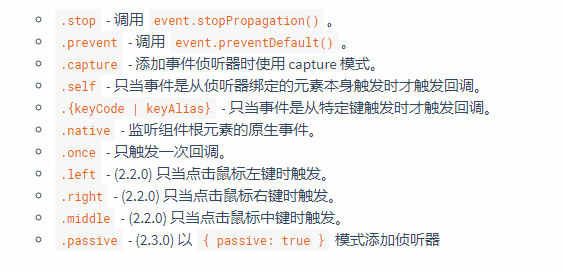
v-on叫做指令，:click叫做参数。

<button v-on**:mouseenter**="add">触摸我</button>

事件名没有on

可以加修饰符，比如.once，此时这个按钮只能被按一次

<button v-on:click**.once**="add">我只能按一次</button>



比较有意思的修饰符就是.13，表示按下回车键：

<input type="text" **v-on:keydown.13="tijiao()"** ref="txt"/>

<script>

//一个项目中，只会实例化vue一次

new Vue({

el : "#app" ,

data : {

a : 100

},

methods : {

add(){

this.a ++;

},

**tijiao(){**

**alert("成功提交" + this.$refs.txt.value)**

**}**

}

});

</script>

v-on可以简写为@：

<button **@**click="add">按我</button>

<button **@**mouseenter="add">触摸我</button>

<button **@**click.once="add">我只能按一次</button>

<input type="text" **@**keydown.13="tijiao($event)" ref="txt"/>

建议大家都@添加监听，并且扎眼，很快能看出这个东西有啥监听。

如果需要用event事件对象，传入$event即可

<input type="text" v-on:keydown.13="tijiao(**$event**)" ref="txt"/>

函数：

<script>

//一个项目中，只会实例化vue一次

new Vue({

el : "#app" ,

data : {

a : 100

},

methods : {

add(){

this.a ++;

},

**tijiao(e){**

**alert("成功提交" + e.target.value)**

**}**

}

});

</script>

## 3.5 v-bind（难点）

**将值变为“动态”，我们使用v-bind。**

v-bind:什么属性都行

比较常见的v-bind是：

v-bind:disabled

v-bind:value

v-bind:style **注意这里面要的是标准JSON**

v-bind:class **注意这里面要的是标准JSON**

<div id="app">

<p>

<button **v-bind:disabled="a"**>你好</button>

</p>

<p>

<input type="text" **v-bind:value="b"** />

</p>

<p>

<h1 **v-bind:style='{**

**"color" : "red",**

**"font-size" : "80px"**

**}'**>你好</h1>

</p>

<p>

<input type="text" **v-bind:placeholder="c"**>

</p>

<p>

<h1 **v-bind:class='{**

**"cu" : true,**

**"hong" : true,**

**"xie" : false**

**}'**>你好</h1>

</p>

</div>



v-bind:value可以简写为:value

<p>

<button **:disabled="a"**>你好</button>

</p>

<p>

<input type="text" **:value="b"** />

</p>

<p>

<h1 **:style='{**

**"color" : "red",**

**"font-size" : "80px"**

**}'**>你好</h1>

</p>

<p>

<input type="text" **:placeholder="c"**>

</p>

<p>

<h1 **:class='{**

**"cu" : true,**

**"hong" : true,**

**"xie" : false**

**}'**>你好</h1>

</p>

冒号为什么是难点呢？在于字符串的拼接。

比如现在d的值是"luhan.png"，此时正确写法有两个：

<p>

<img **:src="`images/${d}`"** alt="">

</p>

<p>

<img **:src="'images/' + d"** alt="">

</p>



# 四、双向数据绑定v-model

## 4.1 概念

受控就是双向绑定：

控件从data要值

控件改变值就改变data

我们在react中如果做表单控件的受控，此时要写value={} 和onChange={}。

但是在vue中，**使用v-model将快速实现数据的双向绑定！堪称神迹！**

原来react：

<input type="text" value={this.state.a} onChange={(e)=>{

this.setState({

a : this.state.a

})

}}>

现在vue：

<div id="app">

<h1>{{a}}</h1>

<input type="text" **v-model="a"**>

</div>

表单元素与a进行双向绑定

注意：

* **v-model暗含事件监听**，携带v-model指令的表单控件，不能有其他的v-on:！
* 只有表单控件才能有v-model，div、p不能有v-model

单选按钮的受控（受控和双向数据绑定是一个意思）：

<p>

<input type="radio" name="sex" value="男" **v-model="b"**/> 男

<input type="radio" name="sex" value="女" **v-model="b"**/> 女

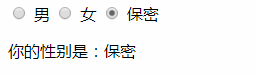
<input type="radio" name="sex" value="保密" **v-model="b"**/> 保密

</p>

<p>

你的性别是：{{b}}

</p>



复选框的受控，直接出数组：

<p>

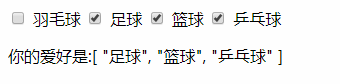
<input type="checkbox" value="羽毛球" name="hobby" **v-model="c"** /> 羽毛球

<input type="checkbox" value="足球" name="hobby" **v-model="c"** /> 足球

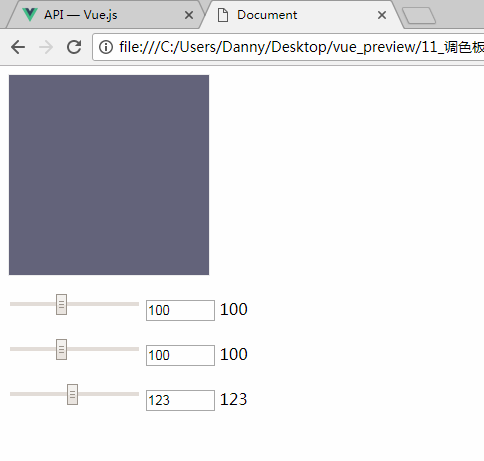
<input type="checkbox" value="篮球" name="hobby" **v-model="c"** /> 篮球

<input type="checkbox" value="乒乓球" name="hobby" **v-model="c"** /> 乒乓球

</p>



## 4.2 调色板



<div id="app">

<div class="box" :style="{'background-color' : `rgb(${color.r},${color.g},${color.b})`}"></div>

<p>

<input type="range" min="0" max="255" **v-model="color.r"**>

<input type="number" min="0" max="255" **v-model="color.r"**>

{{color.r}}

</p>

<p>

<input type="range" min="0" max="255" **v-model="color.g"**>

<input type="number" min="0" max="255" **v-model="color.g"**>

{{color.g}}

</p>

<p>

<input type="range" min="0" max="255" **v-model="color.b"**>

<input type="number" min="0" max="255" **v-model="color.b"**>

{{color.b}}

</p>

</div>

<script src="vue.js"></script>

<script>

new Vue({

el : "#app" ,

data : {

color : {

r : 100,

g : 100,

b : 123

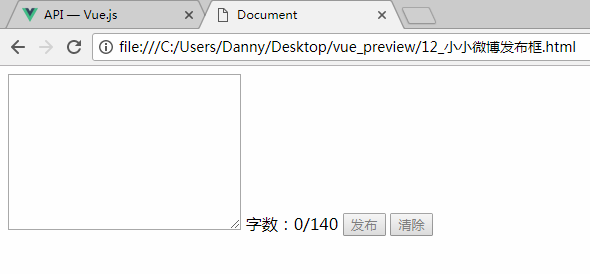
}

}

});

</script>

## 4.3 微博发布框



<div id="app">

<textarea cols="30" rows="10" **v-model="content"**></textarea>

<span :class="{

'danger' : content.length > 140

}">字数：{{content.length}}/140</span>

<button :disabled="content.length > 140 || content.length == 0">发布</button>

<button @click="clear" :disabled="content.length == 0">清除</button>

</div>

<script src="vue.js"></script>

<script>

new Vue({

el : "#app" ,

data : {

content : ""

},

methods : {

clear(){

this.content = ""

}

}

});

</script>

## 4.4 小小购物车



<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Document</title>

<style>

table,tr,td{

border:1px solid #eee;

}

td{

width:200px;

line-height: 30px;

}

</style>

</head>

<body>

<div id="app">

<table>

<tr v-for="item in cart">

<td>{{item.id}}</td>

<td>{{item.name}}</td>

<td>{{item.price}}</td>

<td>

<button **@click="minus(item.id)"**>-</button>

<input type="number" min="0" max="5" **v-model="item.number"**>

<button **@click="add(item.id)"**>+</button>

</td>

<td>{{item.number \* item.price}}</td>

</tr>

</table>

<h1>

总价：

{{

cart.reduce((a,b)=>a + b.price \* b.number , 0)

}}

</h1>

</div>

<script src="vue.js"></script>

<script>

new Vue({

el : "#app" ,

data : {

cart : [

{"id" : 1001 , "name" : "冰箱" , "price" : 1000 , "number" : 2 },

{"id" : 1002 , "name" : "洗衣机" , "price" : 2000 , "number" : 1 },

{"id" : 1003 , "name" : "手机" , "price" : 3000 , "number" : 1 }

]

},

methods : {

add(id){

**this.cart.filter(item=>item.id == id)[0].number++;**

},

minus(id){

**this.cart.filter(item=>item.id == id)[0].number--;**

}

}

});

</script>

</body>

</html>

## 4.5 三级省市县的联动

生动形象的说明了computed的意义。

<!DOCTYPE html>

<html lang="en">

<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>

</head>

<body>

<div id="app">

<p>

<select **v-model="sheng"**>

<option v-for="item in **allsheng**" :value="item">{{item}}</option>

</select>

<select **v-model="shi"**>

<option v-for="item in **allshi**" :value="item">{{item}}</option>

</select>

<select **v-model="xian"**>

<option v-for="item in **allxian**">{{item}}</option>

</select>

</p>

<p>

你的地址：{{sheng}} {{shi}} {{xian}}

</p>

</div>

<script src="vue.js"></script>

<script src="city.js"></script>

<script>

new Vue({

el : "#app" ,

data : {

sheng : "河北省",

shi : "石家庄市",

xian : "桥西区"

},

computed : {

allsheng(){

return citydata.map(item=>item.name);

},

allshi(){

return citydata.filter(item=>item.name == this.sheng)[0].city.map(item=>item.name);

},

allxian(){

var thecity = citydata.filter(item=>item.name == this.sheng)[0].city.filter(item=>item.name == this.shi)[0];

if(thecity){

return thecity.area;

}

return [];

}

},

watch : {

sheng(){

this.shi = "";

this.xian = "";

},

shi(){

this.xian = "";

}

}

});

</script>

</body>

</html>

## 4.6 可见的列

看看复选框的受控：

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Document</title>

<style>

table,tr,td , th{

border:1px solid #eee;

}

td{

width:200px;

line-height: 30px;

}

</style>

</head>

<body>

<div id="app">

<div>

<input type="checkbox" name="cols" value="id" **v-model="showCols"**> 学号

<input type="checkbox" name="cols" value="name" **v-model="showCols"**> 姓名

<input type="checkbox" name="cols" value="sex" **v-model="showCols"**> 性别

<input type="checkbox" name="cols" value="age" **v-model="showCols"**> 年龄

</div>

<table>

<tr>

<th **v-show="showCols.includes('id')"**>学号</th>

<th **v-show="showCols.includes('name')"**>姓名</th>

<th **v-show="showCols.includes('sex')"**>性别</th>

<th **v-show="showCols.includes('age')"**>年龄</th>

</tr>

<tr v-for="item in arr">

<td **v-show="showCols.includes('id')"**>{{item.id}}</td>

<td **v-show="showCols.includes('name')"**>{{item.name}}</td>

<td **v-show="showCols.includes('sex')"**>{{item.sex}}</td>

<td **v-show="showCols.includes('age')"**>{{item.age}}</td>

</tr>

</table>

</div>

<script src="vue.js"></script>

<script>

new Vue({

el : "#app" ,

data : {

**showCols : ["id" ,"name","sex","age"],**

arr : [

{"id" : 1 ,"name" : "小明1" ,"sex" : "男" , "age" : 14},

{"id" : 2 ,"name" : "小明2" ,"sex" : "男" , "age" : 14},

{"id" : 3 ,"name" : "小明3" ,"sex" : "男" , "age" : 14},

{"id" : 4 ,"name" : "小明4" ,"sex" : "男" , "age" : 14}

]

}

});

</script>

</body>

</html>

注意复选框是直接v-model与一个数组进行双向绑定的。

# 五、组件化开发和webpack配置

## 5.1 vue中的组件

我们现在是src引包方式学习vue，做一个组件：

<body>

<div id="app">

<my-child></my-child>

<my-child></my-child>

<my-child></my-child>

</div>

<script src="vue.js"></script>

<script>

//来一个普通对象：

var child = {

template : "<div>我是子组件</div>"

}

var myel = new Vue({

el : "#app" ,

components : {

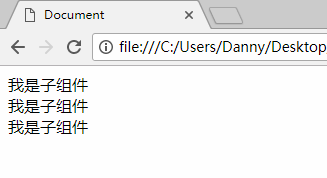
"my-child" : child

}

});

</script>

</body>



* 首先，我们发现，**一个vue中的组件就是一个普通JS对象**，就是JSON。这和React是完全不一样的。**因为react中，一个组件是一个类，并且是一个extends React.Component的类**。但是，这不意味着vue中的组件就不是“组件类”的实例，是。**vue中，组件的实例化是对程序员不透明的**。
* 组件必须进行“注册”操作，在React中，一个组件require了另一个组件，就能无脑使用这个组件，但是vue中不行！必须进行注册。就是在components中注册。

也可以用vue.component(k,v)来进行全局注册。

* React中组件的类名和上树的标签名是一致的，比如类名是App，上树必然是<App></App>。而Vue中用k、v对儿的形式，让你自由注册上树的名字。事实上，vue推荐使用含有短横的名字来表示自定义组件。但是每个公司不一样。



组件也是可以进行传值的：

<body>

<div id="app">

<my-child **:a="10"**></my-child>

<my-child **:a="20"**></my-child>

<my-child **:a="30"**></my-child>

</div>

<script src="vue.js"></script>

<script>

//来一个普通对象：

var child = {

**props : ["a"],**

template : "<div>我是子组件{{a}}</div>"

}

var myel = new Vue({

el : "#app" ,

components : {

"my-child" : child

}

});

</script>

</body>

## 5.2 配置webpack，用.vue文件写组件

vue提供了vue-cli，可以瞬间创建一个项目。但是，初学者千万别用cli起步，因为你不了解一些东西比如jslint等等。

创建一个空白文件夹，创建身份证。

全局安装webpack

cnpm install -g webpack

如果是webpack 4，webpack4要求安装webpack-cli

cnpm install -g webpack-cli

全局安装webpack-dev-server，这是一个前端的模拟小服务器，写不了接口，但是能够静态化一个文件夹。

cnpm install -g webpack-dev-server

创建webpack.config.js文件，注意里面我们多了一个属性：

const path = require('path');

module.exports = {

entry: "./www/app/main",

output: {

path: path.resolve(\_\_dirname, "./www/dist"),

filename: "bundle.js",

**publicPath: "/xuni/" //这是对webpack-dev-server的配置，配置虚拟路径**

}

}

在项目身份证中加上一个启动的命令：

{

"name" : "vue\_study",

"version" : "",

"scripts": {

**"dev" : "webpack-dev-server --content-base ./www --port 8080"**

},

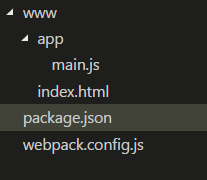
"devDependencies": {

}

}

--content-base表示将什么文件夹静态化，静态化URL

--port表示端口号



注意index.html中引入的是虚拟路径！！

<!DOCTYPE html>

<html lang="en">

<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>

</head>

<body>

<div id="app"></div>

**<script src="/xuni/bundle.js"></script>**

</body>

</html>

也就是说，在项目开发的时候，没有dist文件夹生成的，没有bundle.js文件生成！！！

有的只是8080端口的一个虚拟路径。

启动项目就是

npm run dev

接下来配置vue：

安装一个前端依赖：

cnpm install --save vue

安装3个开发依赖：

cnpm install --save-dev vue-loader

cnpm install --save-dev css-loader

cnpm install --save-dev vue-template-compiler

补充一个别名的配置，https://cn.vuejs.org/v2/guide/installation.html

完整的webpack.config.js文件：

const path = require('path');

module.exports = {

entry: "./www/app/main",

output: {

path: path.resolve(\_\_dirname, "./www/dist"),

filename: "bundle.js",

**publicPath: "/xuni/"**  //这是对webpack-dev-server的配置

},

module: {

rules: [

{

test: /\.vue$/,

include: [

path.resolve(\_\_dirname, "www/app")

],

exclude: [

path.resolve(\_\_dirname, "node\_modules")

],

loader: "vue-loader",

}

]

},

**resolve: {**

**alias: {**

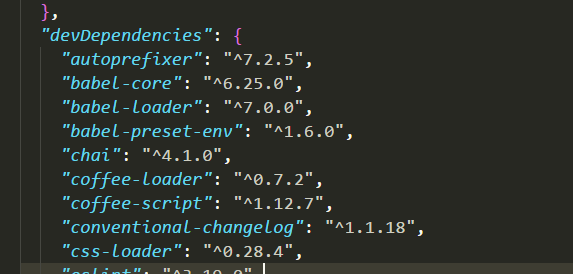
**'vue$': 'vue/dist/vue.esm.js' // 用 webpack 1 时需用 'vue/dist/vue.common.js'**

**}**

**}**

}

vue-loader中含有babel的那些东西：



项目起步：

www/app/main.js

import Vue from "vue";

new Vue({

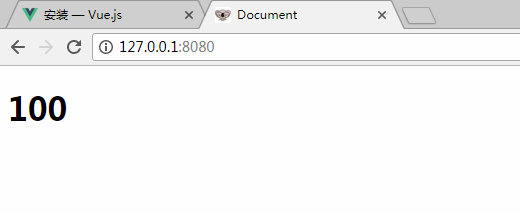
el : "#app" ,

render : h => h(app)

})

npm run dev

8080端口就能看见：



## 5.3 认识.vue文件

.vue文件中有三部分：

template : html结构

script : 程序

style : 样式表

app.vue文件：

<template>

<div>

<h1>我是app.vue组件{{a}}</h1>

</div>

</template>

<script>

import Vue from "vue";

export default {

data(){

return {

a : 100

}

}

}

</script>

<style>

</style>

这个文件可以被父组件引入：

import Vue from "vue";

**import app from "./components/app.vue";**

new Vue({

el : "#app" ,

**render : h => h(app)**

});

**任何一个组件的template必须只有一个根节点。**

每一个vue文件可以引入其他的vue文件，当做自己的组件：

<template>

<div>

<h1>我是app.vue组件{{a}}</h1>

**<my-foo></my-foo>**

**<my-foo></my-foo>**

**<my-foo></my-foo>**

</div>

</template>

<script>

import Vue from "vue";

**import foo from "./foo.vue";**

export default {

data(){

return {

a : 100

}

},

**components : {**

**"my-foo" : foo**

**}**

}

</script>

<style>

</style>



注意：

1）每一个.vue文件中的script部分必须引入vue，否则loader不处理了。

2）data部分是函数，而不是传统的属性对象

export default {

data(){

return {

a : 100

}

},

components : {

"my-foo" : foo

}

}

3）记得组件要注册，才能使用

4）样式表：

<style scoped>

h1{

color:red;

}

</style>

scope可以让组件和组件之间的样式不冲突

# 六、组件父子之间的数据传递和生命周期

## 6.1 父子之间的数据传递

父组件中通过子组件的标签的属性来将值传给子组件：

<template>

<div>

<h1>我是app.vue组件{{a}}</h1>

<button @click="add">按我加1</button>

<my-foo **:a="a"**></my-foo>

</div>

</template>

**子组件需要用props属性来接收父亲的值，props中有的值，data中就绝对不能有同名的变量**。

<script>

import Vue from "vue";

export default {

**props : ["a"] ,**

methods : {

add(){

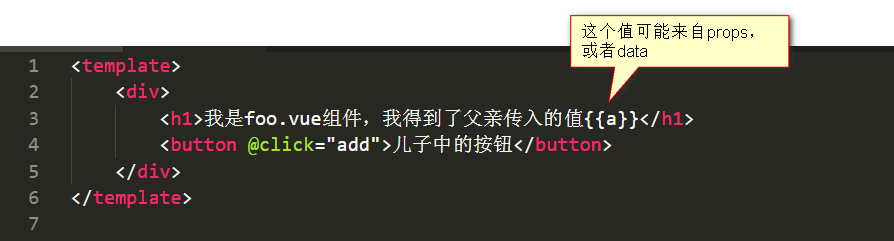
this.a++;

}

}

}

</script>



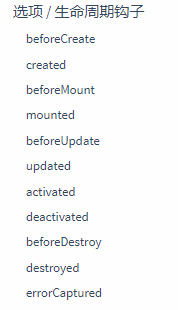
不需要this.props.的前缀。

**传输是单向的，只能父组件传给子组件，子组件不能修改父亲的值。**

**但是，引用类型值，子组件可以合法更改父组件传入的值。**

所以，我们在react开发的时候的方法论：父组件传给子组件一个改变自己state的函数，理论上在vue上是可行的，但是vue开发者往往不会这么做。他们会利用上面红字的特性，传一个JSON或者数组给子组件，子组件就可以合法改变父组件传入的值了。

## 6.2 生命周期



生命周期的图例：



# 七、日历组件

vue中非常好写，主要因为：

1）有v-model：双向数据绑定；

2）有computed、v-for等一些好属性、指令；

3）有:is指令

我们在笔记上只说重点。

大事1：我们喜欢一维数组，但是dom是两层的，两层的v-for嵌套，**我们喜欢的是用两层for从一个一维数组中选择指定的位置**。

[25,26,27,28,1,2,3,4,5,6,7,8,9……,1,2,3,4,5,6,7]

画风：

<table>

<tr **v-for="i in 6"** :key="i">

<td **v-for="j in 7"** :key="j">

**{{arr[i \* 7 + i]}}**

</td>

</tr>

</table>

大事2：computed和生命周期到底用谁？

先说computed怎么立即？一句话：相当于react中render函数中的return之前的部分。

**computed一定会在视图更新前执行**。

vue的文档没有将computed设置为生命周期，但是computed确实在两个声明周期的同时（可以认为是同时）执行：beforeMount、beforeUpdate。

大事3 ：

|  |  |  |
| --- | --- | --- |
| 2000 | 2010 | 2020 |
| 2001 | 2011 | 2021 |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

还是v-for常数简单！

大事儿4：

选择显示哪个组件、不显示哪个组件，vue开发者一般使用v-is指令。

<template>

<div>

<div **:is="'my-' + view + 'picker'" :date="date" :setView="setView"**></div>

<div **:is="'my-' + view + 'view'"** :year="date.year" :month="date.month" :day="date.day" :date="date" :setView="setView"></div>

</div>

</template>