# 一 vue简介

## 1 渐进式

vue是一个渐进式MVVM框架（VM：View-Model视图模型），只关注视图层（view），用来构建Web应用界面。

vue的功能渐进：

声明式渲染： vue的核心库提供了数据渲染功能（即vue作为模板引擎）

实现视图与数据的解耦。

组件系统： 对界面进行组件化

前端路由： 可以用来制作移动端单页面应用

状态管理： 对共享数据进行管理

所谓渐进式，即我们需要哪些功能就使用框架的哪些模块即可，vue这样处理使其减少侵入性。

## 2两个核心点

**1：响应的数据绑定。**

当数据发生改变，自动更新视图。其原理是利用了Object.definedProperty中的setter/getter代理数据，监控对数据的操作。（由于该属性IE8不兼容，所以vue只支持IE9以上版本）

**2：组合的视图组件。**

ui页面可以映射为组件数，这些组件具备可维护、可重用、可调试性。

## 3 虚拟DOM

在操作大量DOM时，js的运行速度会被严重拖累。时常在更新数据后需要重新渲染页面，这样会造成一个困扰：数据未发生改变的地方也要被重新渲染一遍，资源严重浪费。

利用在内存中生成与真实DOM与之对应的数据结构，这个在内存中的结构称之为虚拟DOM，当数据发生变化时，能智能的计算出重新渲染组件所需要的最小资源，并应用到DOM上。

## 4 HelloWorld

|  |
| --- |
| *<!-- 模板 -->* <**div id="demo"**>  <**input type="text" v-model="msg"**>  <**span**>{{msg}}</**span**> </**div**>  <**script**>  *//VUE实例* **new Vue**({  **el**: **'#demo'**, *//挂载元素* **data**: {  **msg**: **'hello world'** } })  *//打印dom* **console**.dir(**document**) </**script**> |

# 二 vue实例介绍

## 1 vue实例的创建

每一个应用都是通过Vue这个构造函数来创建根实例来启动的：

let app = new Vue({

//此处书写各种选项对象

})

传入的选项对象，包含：挂载元素、数据、模板、方法等

el： 挂载点，可以是元素，也可以是CSS选择器，支持原生JS写法，

data： 代理数据

methods：定义方法

## 2 vue实例属性的访问

**每个Vue实例都会代理其对应data对象中的所有属性，这些属性是响应的，而新添加的属性不具备响应功能，改变后不会更新视图。**

Vue实例通过$开头变量暴露自身的属性和方法，如：$el（指定了vue的对象名也可以使用app.$el访问）等，我们可以通过打印vue实例对象来查看Vue实例有哪些自身的属性方法。

let app = new Vue({  
 el: '#app',  
 data: {  
 a: 2  
 }  
});  
console.log(app);  
console.log(app.a); *//2*console.log(app.\_data.a); *//2*

vue实例本身也代理了data对象里的所有属性，所以可以这样访问：

let myData = {  
 a: 1  
};  
let app = new Vue({  
 el: '#app',  
 data: myData  
});  
console.log(app.a); *//1*console.log(app.\_data.a); *//1*app.a = 2; *//修改属性，源数据会更改*console.log(myData.a); *//2*myData.a = 3; *//修改源数据，属性也会更改*console.log(app.a); *//3*

## 3 vue实例的生命周期

每个vue实例创建时，都会经历一系列的初始化过程，同时也会调用对应的生命周期钩子函数：

let app = new Vue({  
 el: '#app',  
 data: {  
 a: 2  
 },  
 created: function () {  
 console.log(this.a); *//2* },  
 mounted: function () {  
 console.log(this.$el); *//<div id="app"></div>* }  
});

created: 实例创建完成后调用，此阶段仅完成了数据观测，未挂载，$el不可用，适用 于数据的初始化；

mounted：el挂载到了实例上后盗用，业务逻辑一般写在此处；

beforeDestroy：实例销毁之前调用，主要解绑一些使用addEventLIstener监听的事件。

## 4 {{}}插值与双向绑定

<div id="app">  
{{a}}  
</div>  
  
<script src="vue2.5.16.js"></script>  
<script>  
let app = new Vue({  
 el: '#app',  
 data: {  
 a: 2  
 }  
});  
</script>

注意：如果要真的输出 {{}} 此时可以使用v-pre指令：<div v-pre>{{hi}}</div>

{{}}除了绑定属性外，可以使用js表达式进行简单的运算，以及使用三元运算符，但不支持语句与流程控制。

## 5 过滤器

vue支持在{{}}尾部添加管道符：(|) 对数据进行过滤，通常用来格式化文本，比如字母全部大写、货币千位使用逗号分隔等，过滤的规则书写在vue实例的filter选项中：

<div id="app">  
 {{a | formUP}}  
</div>  
  
<script src="vue2.5.16.js"></script>  
<script>  
let app = new Vue({  
 el: '#app',  
 data: {  
 a: 'ABC'  
 },  
 filters: {  
 formUP: function (value) { *//value就是需要过滤的数据* return value.toLowerCase();  
 }  
 },  
 mounted: function () {  
 let \_this = this; *// 声明一个变量指向vue实例this，保证作用域一致* this.timer = setInterval(function () {  
 \_this.a = 'ABC ' + new Date();  
 });  
 },  
 beforeDestroy: function() {  
 if (this.timer) {  
 clearInterval(this.timer);  
 }  
 }  
});  
</script>

过滤器可以串联，而且可以修改参数，如：

{{msg | form1 | form2}}

{{msg | form(arg1,arg2)}}

## 6 计算属性

### 6.1 计算属性基本用法

过滤器只能用来处理简单的文本数据，复杂的数据变换，推荐使用计算属性。所有的计算属性都是以函数的形式书写在vue实例的computed选项对象中。

<div id="app">  
{{newNum}}  
</div>  
<script src="vue2.5.16.js"></script>  
<script>  
 let app = new Vue({  
 el: '#app',  
 data: {  
 num: 3  
 },  
 computed: {  
 newNum: function () {  
 return this.num\*3;  
 }  
 }  
  
 });

### 6.2 计算属性的setter和getter

每个计算属性都拥有setter和getter，6.1中的案例默认使用了getter，当手动修改计算属性的值，会触发setter函数，如下 app.fullName=’ww’ 触发了setter。但是大多数业务情况下很少用到setter，通常使用6.1中的默认写法，不必声明setter和getter。

计算属性出了应用于文本插值外，还经常用于动态设置样式名称class，style，动态传递props等。

注意：计算属性还可以依赖其他计算属性，且可以依赖其他vue的实例。

<div id="app">  
 {{fullName}}  
</div>  
<script src="vue2.5.16.js"></script>  
<script>  
 let app = new Vue({  
 el: '#app',  
 data: {  
 firstName: 'ls',  
 lastName: 'zs'  
 },  
 computed: {  
 fullName: {  
 get: function () { *//getter方法 读取时触发* return this.firstName + ' ' + this.lastName;  
 },  
 set: function (newVal) { *//setter方法 写入时触发* this.firstName = 'now set ' + newVal;  
 this.lastName = 'now set ' + newVal;  
 }  
 }  
 }  
 });  
 app.fullName = 'ww';

### 6.3 计算属性与methods区别

计算属性和methods内自定义的方法都可以实现属性的改变。但是计算属性是依赖于缓存的，计算属性依赖的数据发生变化是，才会重新取值，所以一般遍历大数组、大数据推荐使用后计算属性。

# 三 指令

## 1 常见指令汇总

指令是一种特殊的自定义行间属性，在Vue中，指令默认以v-开头。

v-on 绑定事件处理函数，简写为 @

v-bind 动态绑定属性，简写为 ：

v-text 更新数据，会覆盖已有结构

v-html 解析数据中的html结构

v-show 根据值真假，切换元素的dsplay属性

v-if 根据值真假，确定元素是否被销毁、重建

v-for 多次渲染

v-once 只渲染一次，数据更新不会渲染

v-else-if 多条件判断

v-else 条件都不符合，进行渲染

v-model 绑定模型，双向数据绑定

v-pre 跳过元素和子元素编译过程

v-cloak 隐藏未编译的Mustache语法，css中设置[v-cloak]{display:none}

## 2 v-bind v-if

### 2.1 基本使用

v-bind的基本用途是动态更新html元素上的属性，如id，class等：

<div id="app">  
 <span v-if="show">show</span>  
 <a v-bind:src="url" v-on:click="handleClose">链接</a>  
</div>  
<script src="vue2.5.16.js"></script>  
<script>  
 let app = new Vue({  
 el: '#app',  
 data: {  
 show: true,  
 url: '#'  
 },  
 methods: {  
 handleClose: function () {  
 this.show = false;  
 }  
 }  
 });  
</script>

注意：在普通元素上，v-on可以监听原生的DOM事件，监听事件的表达式可以是一个方法名，也可以直接书写方法体：v-on:click=”show = false”

同样的，vue实例也代理了methods方法，上述方法还可以写为：

methods: {  
 handleClose: function () {  
 this.close();  
 },  
 close: function () {  
 this.show = false;  
 }  
}

### 2.3 v-bind绑定class

<div id="app">  
 <span :class="{'active':isActive}"></span>  
</div>  
<script src="vue2.5.16.js"></script>  
<script>  
 let app = new Vue({  
 el: '#app',  
 data: {  
 isActive: true  
 }  
 });  
</script>

对象中也可以传入多个属性，动态切换class，且 :class可以与普通的class共存：

<div class=’static’ :class=”{‘active’:isActive,’error’:isError}”

如果:class的表达式过长或者逻辑非常复杂，可以绑定计算属性，一般当条件多于两个时，都可以使用data或者computed。

多个class可以使用数组进行绑定：:class=”[class1,class2]”，且支持三元运算符。

注意：如果是给组件绑定class，则只绑定到组件的根元素上。

## 3 v-html

如果需要输出html而不是纯文本，可以使用v-html：

<div id="app">  
 <span v-html="link"></span>  
</div>  
  
<script src="vue2.5.16.js"></script>  
<script>  
let app = new Vue({  
 el: '#app',  
 data: {  
 link: '<a href="#">链接</a>'  
 }  
});  
</script>

link的内容被渲染为一个具备点击功能的a标签，需要注意的是，如果将用户产生的内容使用v-html输出后，有可能导致XSS攻击，所以要在服务端对用户提交的内容进行处理，一般可将尖括号 <> 转义。

## 4 v-cloak与v-once

<style>  
 [v-cloak] {  
 display: none;  
 }  
 </style>  
</head>  
<body>  
<div id="app" v-cloak>  
 {{msg}}  
</div>  
<script src="vue2.5.16.js"></script>  
<script>  
 let app = new Vue({  
 el: '#app',  
 data: {  
 msg: '123'  
 }  
 });  
</script>

v-loak 是一个解决初始化慢导致网页闪动的办法，如果网速慢，vue文件没有加载完全，那么会显示{{msg}}字样，直到vue创建实例，编译模板时，DOM才会被替换，为了解决闪动问题，添加上述CSS代码。

但是工程化的项目里，项目的html结构只有一个空的div元素，剩余内容由路由挂载不同组件完成，不再需要该指令。

v-once也不需要表达式，作用是被定义的元素只渲染一次，包括子节点，首次渲染后，不再随着数据的变化重新渲染，将被视为静态内容。一般用于性能优化。

## 5 v-if与v-show

v-show用来控制元素的显示与隐藏。

二者都可以实现对dom的CSS控制，但是v-show只是简单的CSS属性切换，无论条件真与否，都会被编译。所以v-if适合条件不经常改变的场景，因为开销更大。

v-show不能使用在 <template>上。

## 6 v-for

### 6.1 v-for的基本使用

列表渲染也支持使用of来替代in。v-for的表达式支持一个可选参数作为索引：

v-for=”{book,index} in books”

<div id="app">  
 <ul>  
 <li v-for="book in books">  
 {{book.name}}  
 </li>  
 </ul>  
</div>  
<script src="vue2.5.16.js"></script>  
<script>  
 let app = new Vue({  
 el: '#app',  
 data: {  
 books: [  
 {name: '《aaa》'},  
 {name: '《bbb》'},  
 {name: '《ccc》'}  
 ]  
 }  
 });  
 app.books.push({  
 name: '《ddd》'  
 });  
</script>

### 6.2 数组的更新

vue提供了自己的操作数组的方法，使用它们会造成视图的更新，如：

push() pop() shift() unshift() splice() sort() reverse()

app.books.push({

name: ’《ddd》’

});

使用上述方法会改变被这些方法调用的原始数组，但是有些方法不会改变，它们只是返回一个新数组，我们可以用新数组来替换原始数组达到想要的效果：

filter() concat() slice()

app.books = app.books.filter(function(item)}{

return item.match(/aaa/);

});

注意：vue在检查到数组变化时，并不是重新渲染整个列表，而是最大化的复用dom元素，相同的元素的项不会被重新渲染，所以不用担心性能问题。

### 6.3 不会更新视图的数组变动

下变动的数组，vue是无法检测的，不会触发视图更新：

通过索引直接设置项：app.bookks[3] = {}；

修改数组的长度：app.books.length = 1；

解决第一个问题的办法是使用vue内置的set方法：

Vue.set(

app.books,

3,

{}

);

如果是使用了webpack，且使用了组件化开发的方式，默认是没有导入Vue的，需要使用下列方法解决第一个问题：

this.$set(); //在非webpack模式下可以使用app.$set()

更简便的解决办法：splice，上述两种不会触发视图更新的问题都可以使用splice来解决：

app.books.splice(3,1,{});

app.books,splice(1);

### 6.4 过滤与排序

当你不想改变原始数组，又想通过一个数组副本做过滤、排序，那么可以使用计算属性来返回过滤、排序后的数组。

## 7 v-on

### 7.1 基本用法

用来监听DOM事件的触发：v-on:eventName=’’，可以简写为 @。事件处理函数统一书写在methods中，事件处理函数只有纯粹的逻辑判断，不处理DOM事件的细节，如：阻止冒泡、取消默认行为、判断按键等，如下案例给一个输入框添加输入后回车事件addToDo：

在标签中绑定：**v-on:keyup="addToDo"**

在vue中：

|  |
| --- |
| **methods**: {  addToDo(ev){  **if**(ev.**keyCode** === 13){  **this**.***list***.push({  **title**: ev.**target**.**value** })  }  }  注意这里的addToDo是ES6的写法，ES5写为：  addToDo:function(){} |

我们发现这里我们需要手动判断用户敲击enter键的行为，vue提供了这样的事件修饰符来简化我们的操作：

在标签中绑定：v-on:keyup.enter=”addToDo”

在vue中：

|  |
| --- |
| **methods**: {  addToDo(ev){  **this.*list***.push({  **title**: ev.**target**.**value** })  } } |

如上的代码逻辑不符合vue的设计思想，vue主张操作数据，不到万不得已才去操作DOM，在上述案例中，敲击enter键后，toDoList中添加数据，然后渲染在页面中，我们使用ev.target.value来获取输入的数据，这里已经操作了DOM，我们可以这样设计代码：

在vue的数据中添加一个额外的数据nowData

|  |
| --- |
| **data**: {  **list**: ***listData***,  **nowData**: **''** }, |

这时候我们每次enter，利用双向绑定数据记录在这个nowData中，然后vue直接从数据data中获取数据，不需要再去标签中获取用户输入的数据了，

|  |
| --- |
| **v-model="nowData" v-on:keyup.enter="addToDo"**  **methods**: {  addToDo(ev){  **this**.***list***.push({  **title**: **this**.**nowData** })  **this**.**nowData** = **''** } } |

### 7.2 $event

vue提供了一个特殊变量$event用于访问原生的dom事件。

注意事项：我们在标签中定义事件函数时候，也可以直接带上括号，并传参，此时函数会直接执行，但是如果事件函数用到了事件对象，必须手动将事件对象$event传入：

v-on:keyup.enter = “addToDo(自己的参数，$event)”

### 7.3 事件修饰符

关于事件修饰符：事件处理函数只有纯粹的逻辑判断，不处理DOM事件细节，例如阻止冒泡、取消默认行文、判断按键等，书写方式：

v-on:eventName.修饰符

常见修饰符：stop prevent capture self once

按键修饰符：enter tab delete esc space up down left right ctrl alt shift meta 键值

# 四 表单与v-model

## 1 v-model的基本使用

表单控件如：单选、多选、下拉选择、输入框等，可以完成数据的录入、校验，vue提供了v-model指令用语在表单元素上进行双向绑定。

<div id="app">  
 <input type="text" v-model="msg">  
 <p>输入的内容是：{{msg}}</p>  
</div>  
<script src="vue2.5.16.js"></script>  
<script>  
 let app = new Vue({  
 el: '#app',  
 data: {  
 msg: ''  
 }  
 });  
</script>

使用v-model后，表单控件显示的值只依赖所绑定的数据，不再关心初始化时的value属性，如果是输入汉字，一般只在回车时更新，如果需要实时更新，可以用@input替代v-model：

<div id="app">  
 <input type="text" @input="handleInput">  
 <p>输入的内容是：{{msg}}</p>  
</div>  
<script src="vue2.5.16.js"></script>  
<script>  
 let app = new Vue({  
 el: '#app',  
 data: {  
 msg: ''  
 },  
 methods: {  
 handleInput: function (e) {  
 this.msg = e.target.value;  
 }  
 }  
 });  
</script>

## 2 单选按钮

单独使用单选按钮，不需要v-model，只用v-bind绑定一个布尔类型的值即可，为真时选中，为否时不选。如果是组合使用来实现互斥效果，需要v-model配合value使用：

<div id="app">  
 <input type="radio" v-model="picked" value="html" id="html">  
 <label for="html">HTML</label><br>  
 <input type="radio" v-model="picked" value="css" id="css">  
 <label for="css">CSS</label><br>  
 <p>选择的是：{{picked}}</p>  
</div>  
<script src="vue2.5.16.js"></script>  
<script>  
 let app = new Vue({  
 el: '#app',  
 data: {  
 picked: 'css'  
 }  
 });  
</script>

## 3 复选框

复选框在单独使用时，用v-model绑定一个布尔值。组合使用时，也是v-model与value一起使用。同样，数据都是双向绑定的。

<div id="app">  
 <input type="checkbox" v-model="checked" id="checked">  
 <label for="checked">选择状态是：{{checked}}</label><br>  
</div>  
<script src="vue2.5.16.js"></script>  
<script>  
 let app = new Vue({  
 el: '#app',  
 data: {  
 checked: false  
 }  
 });  
</script>

多个组合：

<div id="app">  
 <input type="checkbox" v-model="checked" value="html" id="html">  
 <label for="html">HTML</label><br>  
 <input type="checkbox" v-model="checked" value="css" id="css">  
 <label for="css">CSS</label><br>  
 <p>选择的项是：{{checked}}</p>  
</div>  
<script src="vue2.5.16.js"></script>  
<script>  
 let app = new Vue({  
 el: '#app',  
 data: {  
 checked: ['html','css']  
 }  
 });  
</script>

## 4 选择列表

选择列表即下拉选择器，同样也分为单选和多选，都需要使用v-model。

<div id="app">  
 <select v-model="selected">  
 <option>html</option>  
 <option value="js">javascript</option>  
 </select>  
 <p>选择的是：{{selected}}</p>  
</div>  
<script src="vue2.5.16.js"></script>  
<script>  
 let app = new Vue({  
 el: '#app',  
 data: {  
 selected: 'html'  
 }  
 });  
</script>

如果要实现多选，只用：  
<select v-mode=’selected” mutiple>

此时data中的selected值改为一个数组。

## 5 绑定值

v-mode有时需要绑定一个动态的值，此时需要v-bind。

<div id="app">  
 <input type="checkbox" v-model="toggle" :true-value="value1" :false-value="value2">  
 <label>复选框</label>  
 <p>{{toggle}}</p>  
</div>  
<script src="vue2.5.16.js"></script>  
<script>  
 let app = new Vue({  
 el: '#app',  
 data: {  
 toggle: false,  
 value1: 'a',  
 value2: 'b'  
 }  
 });  
</script>

## 6 修饰符

在输入框中，v-mode默认是在input事件中同步输入框的数据，使用.lazy会转变为在change事件中同步：

v-mode.lazy=”msg”

此时msg并不是实时改变的，而是在失去焦点或者按回车时候才会改变。

.number修饰符用来将数据转换为number类型，否则虽然输入的是数字，但是类型是String。

.trim可以自动过滤输入的首位空格。