# Vue\_02

# 目标

* 掌握vue指令
* 能够定义vue自定义指令
* 使用计算属性和侦听器
* 使用混入完成方法添加
* 掌握生命周期

# 一、常用指令

指令扩展了html标签的功能、大部分的指令的值是js的表达式，取代了DOM操作。

## 1.4、v-if和v-show

根据表达式的布尔值(true/false)进行判断是否渲染该元素

<div id="app">

<!--

v-if是对dom元素的移除和创建

v-show是通过修改标签的display值

-->

<!-- 如果isShow的值是true ,就显示p标签 -->

<p v-if="isShow">我是p标签中的内容</p>

<p v-show="isShow">我是p标签中的内容</p>

</div>

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

<script>

const vm = new Vue({

el: '#app',

data: {

isShow: true

}

})

</script>

注：v-if 有更高的切换开销，而 v-show 有更高的初始渲染开销。因此，如果需要非常频繁地切换，则使用 v-show 较好；如果在运行时条件很少改变，则使用 v-if 较好。

# 多条件判断

v-if … v-else-if v-else



## 1.5、v-bind指令

动态地绑定一个或多个属性

<!-- v-bind绑定href属性值 -->

<a v-bind:href='url'>跳转</a>

<!-- v-bind绑定href属性值（简写形式） -->

<a :href='url'>跳转</a>





## 1.6、v-for

根据一组数组或对象的选项列表进行渲染。

v-for指令需要使用 (item,index) in 数组或对象 形式的特殊语法，同时还需要指定key值,key的作用在vue进行新旧数据比对渲染页面里，如果有**key值会提升比对性能**，加快渲染，key使用唯一的值来赋值。

<div id="app">

<ul>

<!-- 数组循环 -->

<li v-for="u in users" :key="u">{{ u }}</li>

</ul>

<hr>

<ul>

<!-- 对象循环 -->

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

</ul>

</div>

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

<script>

const vm = new Vue({

el: '#app',

data: {

users: [1, 2, 3],

obj: {

username: 'zhangsan',

age: 28,

gender: 'male'

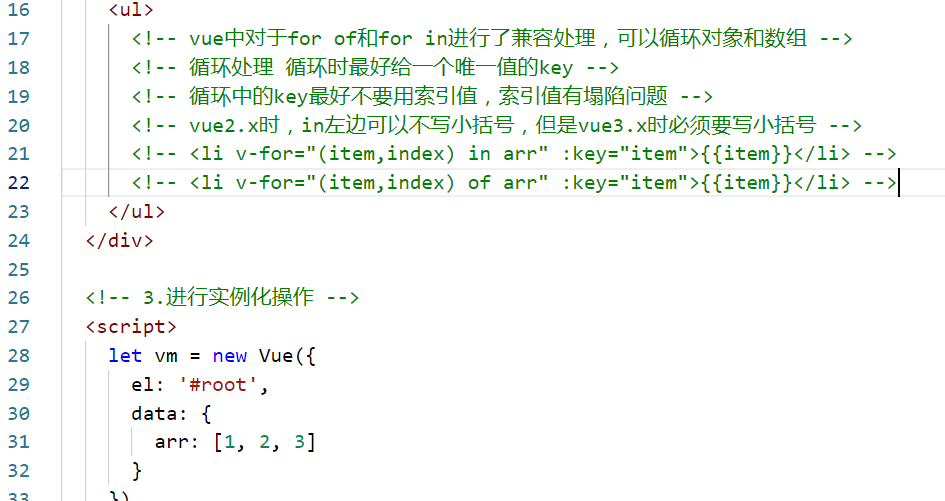
}

}

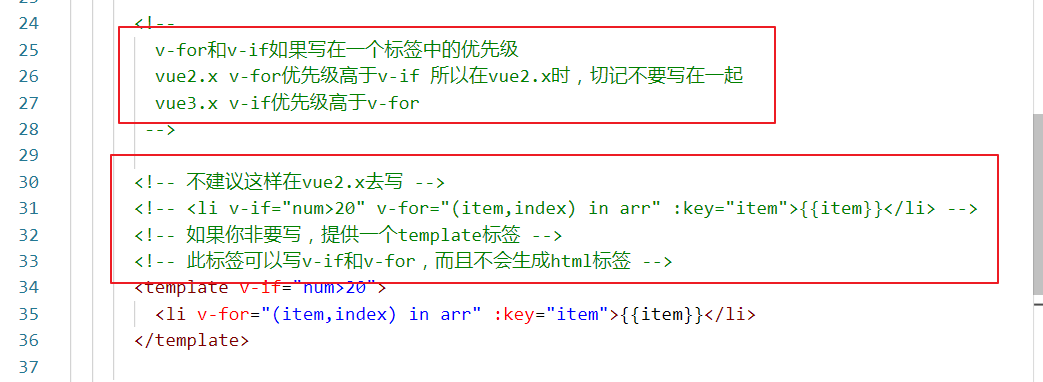
})

</script>

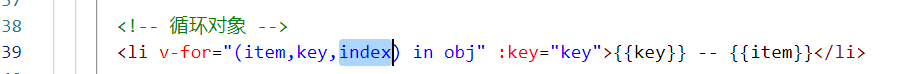
循环数组



v-if和v-for同时存在时的优先级处理



循环对象





## 1.7、v-on

绑定事件监听器（事件绑定）

<!-- 常规写法 -->

<button v-on:click="事件实现方法()"></button>

<!-- 缩写 -->

<button @click="事件实现方法()"></button>

# 绑定好事件实现方法后需要在Vue对象中的methods对象中实现对应的绑定方法

methods: {

functionName(arg1,arg2,arg3,...){

// something to do

},

....

}



* 事件处理函数传参 -- 事件对象--- event对象

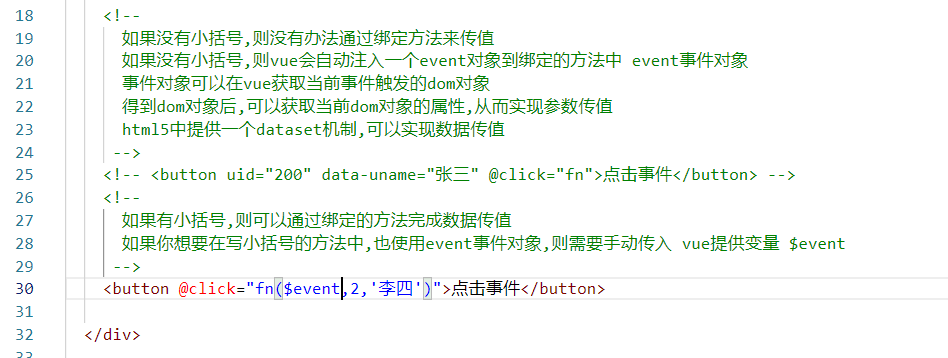
<!-- 事件处理函数调用：直接写函数名 -->

<button @click="say"></button>

<!-- 事件处理函数调用：常规调用 -->

<button @click="say($event)"></button>

# 注：如果没有参数时，可以不写小括号，默认就会把event事件对象绑定到实现的方法中，如果需要传参数时，则通过 $event 来手动传递到实现的方法中





* 事件修饰符

用来处理事件的特定行为

<!-- 阻止冒泡 -->

<button @click.stop="doThis"></button>

<!-- 阻止默认行为 -->

<a @click.prevent="doThis"></a>

<!-- 只执行一次 -->

<div @click.once="incr()">自增一下</div>

<!-- 绑定的元素本身触发时才触发回调 -->

<ul @click.self="incr()">

<li>你好世界</li>

</ul>

<!-- 串联修饰符 -->

<button @click.stop.prevent="doThis"></button>

* 按键修饰符

在监听键盘事件时，我们经常需要检查详细的按键。Vue 允许为 v-on 在监听键盘事件时添加按键修饰符。

<!-- 只有在 `key` 是 `Enter` 回车键的时候调用 -->

<input @keyup.enter="submit">

<!-- Ctrl + Click -->

<div @click.ctrl="doSomething">Do something</div>

你还可以通过全局 config.keyCodes 对象自定义按键修饰符别名：

// 可以使用 `v-on:keyup.f1`

Vue.config.keyCodes = {f2:113}

<input @keyup.f2="add()" value="aaaa">

* todolist

## 1.8、样式绑定

### 1.8.1、class样式处理

* 对象语法

.active {

color: red;

}

<div v-bind:class="{active: isActive}">class样式</div>

data: {

// 如果为true时，样式生效，如果为false不生效

isActive: true

}



* 数组语法

.active {

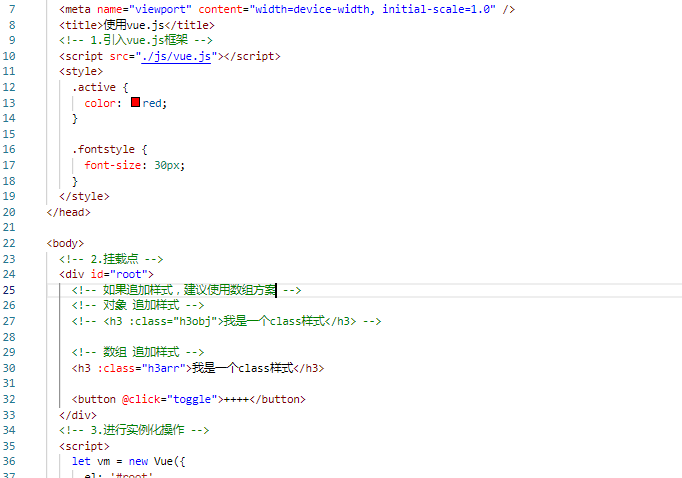
color: red;

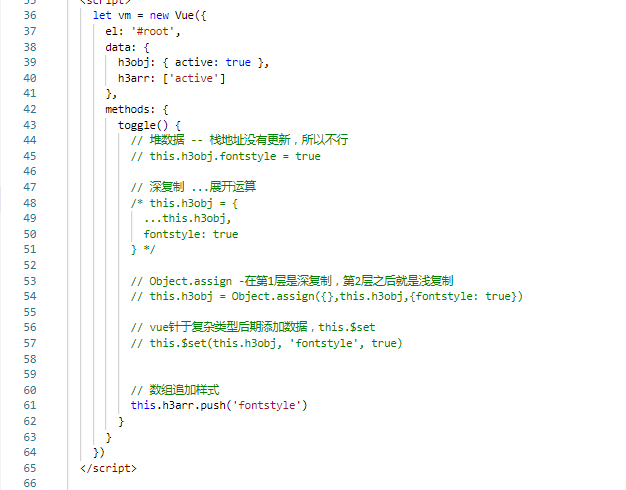
}

<div v-bind:class="['active']">数组写法</div>



追加样式





### 1.8.2、绑定style

* 对象语法

<div :style="{color: redColor, fontSize: '20px'}">对象写法</div>

data: {

redColor: 'red'

}

* 数组语法

<div v-bind:style="[color, fontSize]">数组写法</div>

data: {

color: {

color: 'red'

},

fontSize: {

'font-size': '20px'

}

}



## 1.9、v-model

作用: 表单元素的绑定，实现了双向数据绑定，通过表单项可以更改数据。

v-model会忽略所有表单元素的value、checked、selected特性的初始值,而总是将Vue实例的数据作为数据来源，应该在data选项中声明初始值。

* 绑定文本框

<p>{{message}}</p>

<input type='text' v-model='message'>

<!-- v-model其实是语法糖,它是下面这种写法的简写 -->

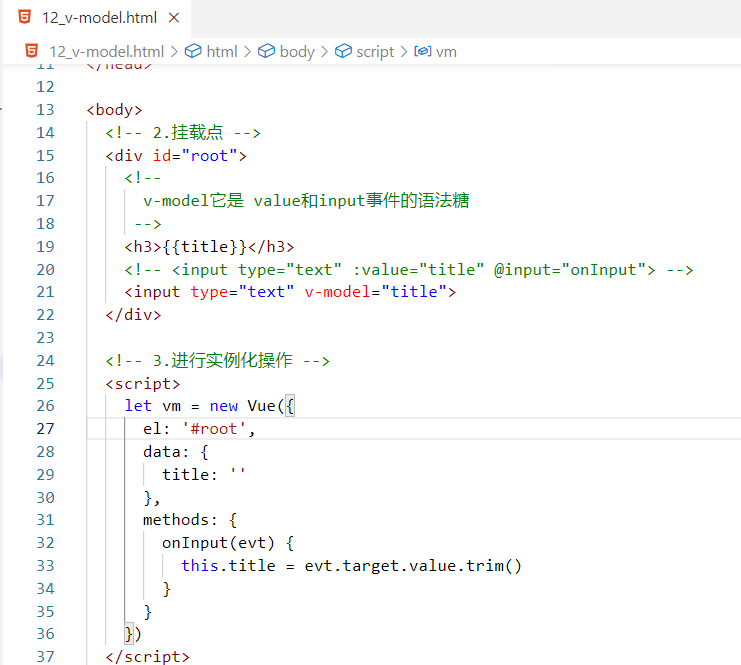
<!-- 语法糖：这种语法对语言的功能并没有影响，但是更方便程序员使用 -->

<input v-bind:value='message' v-on:input='message = $event.target.value' />

data: {

message: 'message默认值'

}



* 绑定多行文本框

<textarea v-model="message"></textarea>

data: {

message: '我是多行文本内容'

}

注意：多行文本框中使用插值表达式 无效



* 绑定单个复选框

# 绑定一个复选框

<input type="checkbox" v-model="checked">

data:{

// 布尔值 false/true true勾选

checked:true

}

* 绑定多个复选框

此种方式需要input标签提供value属性

<input type="checkbox" value="html" v-model="checkedNames">

<input type="checkbox" value="css" v-model="checkedNames">

<input type="checkbox" value="js" v-model="checkedNames">

data:{

// 如果数组中有对应的value值，则此checkbox会被选中

checkedNames:[]

}



* 绑定单选框

需要input提供value属性的值

男<input type="radio" value="男" v-model="sex">

女<input type="radio" value="女" v-model="sex">

data: {

sex: ''

}

* 绑定下拉框

<select v-model="selected">

<option>请选择</option>

<option>HTML</option>

<option>CSS</option>

<option>JS</option>

</select>

data: {

selected: ''

}



* 修饰符

.lazy 失去焦点时触发(延时更新数据源)

.number 输入值转为数值类型

.trim 自动过滤用户输入的首尾空白字符



# 二、自定义指令

<https://cn.vuejs.org/v2/guide/custom-directive.html>

除了核心功能默认内置的指令，Vue也允许注册自定义指令。有的情况下，对普通 DOM 元素进行底层操作，这时候就会用到自定义指令绑定到元素上执行相关操作。

自定义指令分为：全局指令和局部指令，当全局指令和局部指令同名时以局部指令为准。

自定义指令常用钩子函数

* bind 第一次绑定到元素时调用
* inserted 被绑定元素插入父节点时调用 (仅保证父节点存在，但不一定已被插入文档中)
* update 数据更新时调用
* componentUpdated 指令所在组件的 VNode 及其子 VNode 全部更新后调用。
* unbind 只调用一次，指令与元素解绑时调用。

# 全局

Vue.directive('test',{

bind(el,bind){

console.log(el)

}

})

# 局部

new Vue({

directives: {

test:{

bind(el,bind){}

},

// bind/update

test2(el,bind){}

}

})

# 三、计算属性 - computed

模板中放入太多的逻辑会让模板过重且难以维护，使用计算属性可以让模板变得简洁易于维护。计算属性是基于它们的响应式依赖进行缓存的，计算属性比较适合对多个变量或者对象进行处理后返回一个结果值，也就是数多个变量中的某一个值发生了变化则我们监控的这个值也就会发生变化。

计算属性定义在Vue对象中，通过关键词computed属性对象中定义一个个函数，并返回一个值，使用计算属性时和data中的数据使用方式一致

<div id="app">

<!-- 当多次调用 cfn计算属性时只要里面的 num值不改变,它会把第一次计算的结果直接返回。

直到data中的num值改变 计算属性才会重新发生计算 -->

<div>{{ cfn }}</div>

<div>{{ cfn }}</div>

<!-- 调用methods中的方法的时候 他每次会重新调用 -->

<div>{{ fn() }}</div>

<div>{{ fn() }}</div>

</div>

<script type="text/javascript">

const vm = new Vue({

el: '#app',

data: {

num: 10

},

// 方法

methods: {

fn() {

console.log('methods')

return this.num

}

},

// 计算属性

computed: {

cfn() {

console.log('computed')

return this.num

}

}

})

</script>

注：只要依赖的数据源不发生改变，我就调用1次，其它使用缓存

# 四、侦听器 - watch

使用watch来侦听data中数据的变化,watch中的属性一定是data 中已经存在的数据。

当需要监听一个对象的改变时，普通的watch方法无法监听到对象内部属性的改变，只有data中的数据才能够监听到变化，此时就需要deep属性对对象进行深度监听。

<div id="app">

<input type="text" v-model="username">

</div>

<script>

const vm = new Vue({

el: '#app',

data: {

username: 'aaaa'

},

watch: {

username(newVal, oldVal) {

console.log(newVal, oldVal)

}

}

})

</script>

-----------------------------------------------------------

# 对象侦听

<div id="app">

<input type="text" v-model="obj.username">

</div>

<script type="text/javascript">

const vm = new Vue({

el: '#app',

data: {

obj: {

username: '张三'

}

},

computed:{

obj1:function(){

return JSON.parse(JSON.stringify(this.obj))

}

},

watch: {

obj1: {

handler(newVal, oldVal) {

console.log(newVal, oldVal)

},

// 深度侦听

deep: true

}

}

})

</script>

# 五、过滤器 - filters

在数据被渲染之前，可以对其进行进一步处理，比如将字符截取或者将小写统一转换为大写等等，过滤器本身就是一个方法。

过滤器可以定义全局或局部

# 全局

// 回调函数中的参数1永久是绑定的数据

Vue.filter(过滤器名称,(value,[…args])=>{})

# 局部过滤器定义

此过滤器定义在Vue对象的配置中的filters配置选项中，以方法形式来定义，参数1永久是绑定的数据

new Vue({

filters:{

plusNum:function(value){

return value\*10

},

// 脱敏

tuomin:function(phoneNum){

return `${phoneNum.substring(0,3)}\*\*\*\*${phoneNum

.substring(7)}`

}

}

})

# 六、混入 - Mixins

混入(mixins)是一种分发Vue组件中可复用功能的非常灵活的方式。混入对象可以包含任意组件选项。当组件使用混入对象时，所有混入对象的选项将被混入该组件本身的选项。

混入分为：全局和局部

# 全局混入

Vue.mixin({

created(){},

methods:{}

})

# 局部混入

let mixinObj = {}

new Vue({

mixins:[mixinObj]

})

# 七、实例生命周期

每个 Vue 实例在被创建之前都要经过一系列的初始化过程。例如需要设置数据监听、编译模板、挂载实例到 DOM、在数据变化时更新 DOM 等。同时在这个过程中也会运行一些叫做生命周期钩子的函数，给予用户机会在一些特定的场景下添加他们自己的代码。

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | beforeCreate | 在实例初始化之后，数据观测和事件配置之前被调用 此时data 和 methods 以及页面的DOM结构都没有初始化 什么都做不了 执行1次 | | created | 在实例创建完成后被立即调用此时data 和 methods已经可以使用 但是页面还没有渲染出来 执行1次 用this对象 | | beforeMount | 在挂载开始之前被调用 此时页面上还看不到真实数据 只是一个模板页面而已 执行1次 | | mounted | el被新创建的vm.$el替换，并挂载到实例上去之后调用该钩子。 数据已经真实渲染到页面上 在这个钩子函数里面可以进行数据请求等 执行1次 | | beforeUpdate | 数据更新时调用，页面上数据还是旧的 n次 | | updated | 由于数据更新完毕，页面上数据已经替换成最新的 n次 | | beforeDestroy | 实例销毁之前调用 执行1次 | | destroyed | 实例销毁后调用 执行1次 | | activated | keep-alive 组件激活时调用 | | deactivated | keep-alive 组件停用时调用 | | errorCaptured | 当捕获一个来自子孙组件的错误时被调用 | |