# Vue

先安装node.js

node.js:运行在服务端的 JavaScript。

查看版本

node -v

新建helloworld.js 写入： console.log("Hello World");

用node命令执行本地的js文件

node helloworld.js

1.引入vue.js

2. NPM 很好地和诸如 Webpack 或 Browserify 模块打包器配合使用

NPM是随同NodeJS一起安装的包管理工具，能解决NodeJS代码部署上的很多问题，常见的使用场景有以下几种：

允许用户从NPM服务器下载别人编写的第三方包到本地使用。

允许用户从NPM服务器下载并安装别人编写的命令行程序到本地使用。

允许用户将自己编写的包或命令行程序上传到NPM服务器供别人使用。

由于新版的nodejs已经集成了npm，所以之前npm也一并安装好了。同样可以通过输入 "npm -v" 来测试是否成功安装。命令如下，出现版本提示表示安装成功

升级

npm install npm -g

或者

淘宝镜像的命令 cnpm install npm -g

安装模块 npm install <Module Name> （npm install express） 代码引用var express = require('express');

全局安装与本地安装

npm install express # 本地安装

npm install express -g # 全局安装

查看所有全局安装的模块

npm list -g

报错:

npm ERR! error no such file or directory, open 'C:\Program Files\nodejs\node\_modules\npm-debug.log\package.json

查看某个模块的版本号

npm list grunt

package.json 位于模块的目录下，用于定义包的属性

name - 包名。

version - 包的版本号。

description - 包的描述。

homepage - 包的官网 url 。

author - 包的作者姓名。

contributors - 包的其他贡献者姓名。

dependencies - 依赖包列表。如果依赖包没有安装，npm 会自动将依赖包安装在 node\_module 目录下。

repository - 包代码存放的地方的类型，可以是 git 或 svn，git 可在 Github 上。

main - main 字段是一个模块ID，它是一个指向你程序的主要项目。就是说，如果你包的名字叫 express，然后用户安装它，然后require("express")。

keywords - 关键字

卸载模块

npm uninstall express

/node\_modules/ 目录下查看包是否还存在，或者使用以下命令查看：

$ npm ls

更新模块

npm update express

创建模块

创建模块，package.json 文件是必不可少的。我们可以使用 NPM 生成 package.json 文件

npm init

接下来我们可以使用以下命令在 npm 资源库中注册用户（使用邮箱注册）：

$ npm adduser

Username: mcmohd

Password:

Email: (this IS public) mcmohd@gmail.com

接下来我们就用以下命令来发布模块：

$ npm publish

使用淘宝 NPM 镜像

大家都知道国内直接使用 npm 的官方镜像是非常慢的，这里推荐使用淘宝 NPM 镜像。

淘宝 NPM 镜像是一个完整 npmjs.org 镜像，你可以用此代替官方版本(只读)，同步频率目前为 10分钟 一次以保证尽量与官方服务同步。

你可以使用淘宝定制的 cnpm (gzip 压缩支持) 命令行工具代替默认的 npm:

$ npm install -g cnpm --registry=https://registry.npm.taobao.org

这样就可以使用 cnpm 命令来安装模块了：

$ cnpm install [name]

C:\Users\Administrator>npm install vue

C:\Users\Administrator

`-- vue@2.3.4

npm WARN enoent ENOENT: no such file or directory, open 'C:\Users\Administrator\package.json'

npm WARN Administrator No description

npm WARN Administrator No repository field.

npm WARN Administrator No README data

npm WARN Administrator No license field.

这里面的依赖包可能版本或名字不对

npm init -f

npm install vue

npm WARN hebaisheng@3.10.10 No description

npm WARN hebaisheng@3.10.10 No repository field. //缺少项目的仓库字段

仓库字段

"repository": {

"type": "git",

"url": "http://baidu.com"

},

这种方式把项目声明为私有。

"private": true,

应该是这样的：

C:\Users\Administrator>npm install vue

hebaisheng@3.10.10 C:\Users\Administrator

`-- vue@2.3.4

vue:

读音: v-u-e

view

vue到底是什么?

一个mvvm框架(库)、和angular类似

比较容易上手、小巧

mvc:

mvp

mvvm

mv\*

mvx

官网:http://cn.vuejs.org/

手册： http://cn.vuejs.org/api/

vue和angular区别?

vue——简单、易学

指令以 v-xxx

一片html代码配合上json，在new出来vue实例

个人维护项目

适合: 移动端项目,小巧

vue的发展势头很猛，github上start数量已经超越angular

angular——上手难

指令以 ng-xxx

所有属性和方法都挂到$scope身上

angular由google维护

合适: pc端项目

共同点: 不兼容低版本IE

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

vue基本雏形:

angular展示一条基本数据:

var vm=angular.module('vm',[]);

vm.controller('xxx',function($scope){ //C

$scope.msg='welcome'

})

html:

div ng-controller="xxx"

{{msg}}

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

vue:

html:

<div id="box">

{{msg}}

</div>

var c=new Vue({

el:'#box', //选择器 class tagName

data:{

msg:'welcome vue'

}

});

常用指令:

angular:

ng-model ng-controller

ng-repeat

ng-click

ng-show

$scope.show=function(){}

指令: 扩展html标签功能,属性

v-model 一般表单元素(input) 双向数据绑定

循环:

v-for="name in arr"

{{$index}}

v-for="name in json"

{{$index}} {{$key}}

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

v-for="(k,v) in json"

事件:

v-on:click="函数"

v-on:click/mouseout/mouseover/dblclick/mousedown.....

new Vue({

el:'#box',

data:{ //数据

arr:['vmle','banana','orange','pear'],

json:{a:'vmle',b:'banana',c:'orange'}

},

methods:{

show:function(){ //方法

alert(1);

}

}

});

显示隐藏:

v-show=“true/false”

bootstrap+vue简易留言板(todolist):

bootstrap: css框架 跟jqueryMobile一样

只需要给标签 赋予class，角色

依赖jquery

确认删除？和确认删除全部么?

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

事件:

v-on:click/mouseover......

简写的:

@click="" 推荐

事件对象:

@click="show($event)"

事件冒泡:

阻止冒泡:

a). ev.cancelBubble=true;

b). @click.stop 推荐

默认行为(默认事件):

阻止默认行为:

a). ev.preventDefault();

b). @contextmenu.prevent 推荐

键盘:

@keydown $event ev.keyCode

@keyup

常用键:

回车

a). @keyup.13

b). @keyup.enter

上、下、左、右

@keyup/keydown.left

@keyup/keydown.right

@keyup/keydown.up

@keyup/keydown.down

.....

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

属性:

v-bind:src=""

width/height/title....

简写:

:src="" 推荐

<img src="{{url}}" alt=""> 效果能出来，但是会报一个404错误

<img v-bind:src="url" alt=""> 效果可以出来，不会发404请求

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

class和style:

:class="" v-bind:class=""

:style="" v-bind:style=""

:class="[red]" red是数据

:class="[red,b,c,d]"

:class="{red:a, blue:false}"

:class="json"

data:{

json:{red:a, blue:false}

}

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

style:

:style="[c]"

:style="[c,d]"

注意: 复合样式，采用驼峰命名法

:style="json"

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

模板:

{{msg}} 数据更新模板变化

{{\*msg}} 数据只绑定一次

{{{msg}}} HTML转意输出

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

过滤器:-> 过滤模板数据

系统提供一些过滤器:

{{msg| filterA}}

{{msg| filterA | filterB}}

uppercase eg: {{'welcome'| uppercase}}

lowercase

capitalize

currency 钱

{{msg| filterA 参数}}

....

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

交互:

$http （ajax）

如果vue想做交互

引入: vue-resouce

get:

获取一个普通文本数据:

this.$http.get('aa.txt').then(function(res){

alert(res.data);

},function(res){

alert(res.status);

});

给服务发送数据:√

this.$http.get('get.php',{

a:1,

b:2

}).then(function(res){

alert(res.data);

},function(res){

alert(res.status);

});

post:

this.$http.post('post.php',{

a:1,

b:20

},{

emulateJSON:true

}).then(function(res){

alert(res.data);

},function(res){

alert(res.status);

});

jsonp:

https://sug.so.360.cn/suggest?callback=suggest\_so&word=a

https://sp0.baidu.com/5a1Fazu8AA54nxGko9WTAnF6hhy/su?wd=a&cb=jshow

this.$http.jsonp('https://sp0.baidu.com/5a1Fazu8AA54nxGko9WTAnF6hhy/su',{

wd:'a'

},{

jsonp:'cb' //callback名字，默认名字就是"callback"

}).then(function(res){

alert(res.data.s);

},function(res){

alert(res.status);

});

https://www.baidu.com/s?wd=s

作业:

1. 简易留言-> 确认删除? 确认删除全部

2. 用vue get 写个例子 weibo

//vue

-------------------文本插值

模板语法来声明式的将数据渲染进 DOM

DOM:

<div id="vm">

{{ message }}

//Mustache” 语法 双大括号

// v-once 指令 ：执行一次性地插值

//输出真正的 HTML: v-html="rawHtml"

//模版的data属性 rawHtml:'<p>内容将会被替换成为属性值 rawHtml</p>'

</div>

模板语法:

//Vue 实例

//传入的对象可以是：数据、模板、挂载元素、方法、生命周期钩子

var vm = new Vue(object) //ViewModel 的简称 这个变量名表示 Vue 实例

var vm = new Vue({

el:'#vm' //class id elename

,data{

mode-name:'hello vue'

}

})

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

//html属性

Mustache 不能在 HTML 属性中使用 应使用 v-bind 指令

v-bind:id="dynamicId"

v-bind:disabled="isButtonDisabled"

使用 JavaScript 表达式

绑定都只能包含单个表达式

{{ number + 1 }}

{{ ok ? 'YES' : 'NO' }}

{{ message.split('').reverse().join('') }}

模板表达式都被放在沙盒中，只能访问全局变量的一个白名单，如 Math 和 Date 。你不应该在模板表达式中试图访问用户定义的全局变量。

指令：v-on v-bind

参数:在指令后以冒号指明 v-bind:href v-on:click //监听的事件名

修饰符：v-on:submit.prevent //. 指明的特殊后缀

过滤器：过滤器应该被添加在 JavaScript 表达式的尾部，由“管道”符指示：

<!-- in mustaches -->

{{ message | capitalize }}

<!-- in v-bind -->

<div v-bind:id="rawId | formatId"></div>

//可以串联

{{ message | filterA | filterB }}

//过滤器函数

new Vue({

// ...

filters: {

capitalize: function (value) {

if (!value) return ''

value = value.toString()

return value.charAt(0).toUpperCase() + value.slice(1)

}

}

})

{{ message | filterA('arg1', arg2) }}

//表达式message的值作为第一个参数 ，字符串 'arg1' 将传给过滤器作为第二个参数， arg2 表达式的值将被求值然后传给过滤器作为第三个参数。

缩写：

v-bind 缩写 ：

v-on 缩写：@

--------------------------计算属性

DOM:

<div id="example">

<p>Original message: "{{ message }}"</p>

<p>Computed reversed message: "{{ reversedMessage }}"</p>

</div>

模版语法：

var vm = new Vue({

el: '#example',

data: {

message: 'Hello'

},

computed: {

// a computed getter

//计算属性 reversedMessage

reversedMessage: function () {

// `this` points to the vm instance

return this.message.split('').reverse().join('')

}

}

})

console.log(vm.reversedMessage) // -> 'olleH'

vm.message = 'Goodbye'

console.log(vm.reversedMessage) // -> 'eybdooG'

//计算属性 对比 Methods

我们为什么需要缓存？假设我们有一个性能开销比较大的的计算属性 A ，它需要遍历一个极大的数组和做大量的计算。然后我们可能有其他的计算属性依赖于 A 。如果没有缓存，我们将不可避免的多次执行 A 的 getter！如果你不希望有缓存，请用 method 替代。

message改变，才会执行计算属性

Computed 属性 对比 Watched 属性

当你有一些数据需要随着其它数据变动而变动时.通常更好的想法是使用 computed 属性.

计算 setter

计算属性默认只有 getter

不过在需要时你也可以提供一个 setter

DOM:

<div id="demo">{{ fullName }}</div>

模版语法：

data: {

firstName: 'Foo',

lastName: 'Bar'

},

computed: {

fullName: {

// getter

get: function () {

return this.firstName + ' ' + this.lastName

},

// setter

set: function (newValue) {

var names = newValue.split(' ')

this.firstName = names[0]

this.lastName = names[names.length - 1]

}

}

}

观察 watch

当你想要在数据变化响应时，执行异步操作或开销较大的操作

DOM：

<input v-model="question">

<p>{{ answer }}</p>

模版语法：

data: {

question: '',

answer: 'I cannot give you an answer until you ask a question!'

},

watch: {

// 如果 question 发生改变，这个函数就会运行

question: function (newQuestion) {

this.answer = 'Waiting for you to stop typing...'

this.getAnswer()

}

},

methods: {

getAnswer:function (){

}

}

-----------------------------------Class 与 Style 绑定

v-bind:class 指令可以与普通的 class 属性共存。

绑定 HTML Class:

对象:

<div class="static"

v-bind:class="{ active: isActive, 'text-danger': hasError }">

</div>

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

data: {

isActive: true,

hasError: false

}

绑定数据里的一个对象：

dom:

<div v-bind:class="classObject"></div>

模版语法：

data: {

classObject: {

active: true,

'text-danger': false

}

}

绑定返回对象的计算属性，进行逻辑处理：

<div v-bind:class="classObject"></div>

data: {

isActive: true,

error: null

},

computed: {

classObject: function () {

var obj = {

active: this.isActive && !this.error,

'text-danger': this.error && this.error.type === 'fatal'

}

return obj;

}

}

数组语法:

<div v-bind:class="[activeClass, errorClass]">

data: {

activeClass: 'active',

errorClass: 'text-danger'

}

切换列表中的 class

<div v-bind:class="[isActive ? activeClass : '', errorClass]">

用在组件上

当你在一个定制的组件上用到 class 属性的时候，这些类将被添加到根元素上面，这个元素上已经存在的类不会被覆盖。

绑定内联样式

v-bind:style 的对象语法十分直观——看着非常像 CSS ，其实它是一个 JavaScript 对象。

CSS 属性名可以用驼峰式 (camelCase) 或 (配合引号的) 短横分隔命名 (kebab-case)：

<div v-bind:style="{ color: activeColor, fontSize: fontSize + 'px' }"></div>

data: {

activeColor: 'red',

fontSize: 30

}

绑定到一个样式对象通常更好

<div v-bind:style="styleObject"></div>

data: {

styleObject: {

color: 'red',

fontSize: '13px'

}

}

数组语法

v-bind:style 的数组语法可以将多个样式对象应用到一个元素上：

<div v-bind:style="[baseStyles, overridingStyles]">

自动添加前缀,无需 transform

-----------------------------------Vue的属性与方法

每个 Vue 实例都会代理其 data 对象里所有的属性：

var data = { key: 1 }//代理的

var vm = new Vue({

data: data

})

vm.key === data.key // -> true

//如果在实例创建之后添加新的属性到实例上，它不会触发视图更新

属性1

vm.$data === data // -> true

属性2

vm.$el === document.getElementById('example') // -> true

属性3

// $watch 是一个实例方法

vm.$watch('key', function (newVal, oldVal) {

// 这个回调将在 `vm.key` 改变后调用

})

-----------------------------vue实例生命周期

实例化过程中会调用一些 生命周期钩子

created 这个钩子：

var vm = new Vue({

data: {

a: 1

},

created: function () {

// `this` 指向 vm 实例

//这个钩子在实例被创建之后被调用

console.log('a is: ' + this.a)

}

})

// mounted、updated、destroyed。钩子的 this 指向调用它的 Vue 实例

--------------------------------------vue生命周期

new Vue() --->created之前需要 配置数据观测，编译模版，挂载实例到 DOM，在数据变化时更新 DOM

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

绑定 DOM 元素属性:

例如鼠标悬停：v-bind:title

DOM:

<span v-bind:title="message">

鼠标悬停几秒钟查看此处动态绑定的提示信息！

</span>

模版语法：

message: '页面加载于 ' + new Date()

--------------------条件

DOM:

<p v-if="seen">现在你看到我了</p>

模版语法：

seen: true

非模版设置：

vm.seen = false

--------------------循环

DOM:

<ol>

<li v-for="todo in todos">

{{ todo.text }}

</li>

</ol>

模版语法：

todos: [

{ text: '学习 JavaScript' },

{ text: '学习 Vue' },

{ text: '整个牛项目' }

]

非模版控制：

vm.todos.push({ text: '新项目' })

------------------------处理用户输入

v-on 指令绑定一个事件监听器

通过它调用我们 Vue 实例中定义的方法

DOM：

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

<button v-on:click="onMessage">逆转消息</button>

模版语法：

data: {

message: 'Hello Vue.js!'

},

//方法

methods: {

//自定义

reverseMessage: function () {

//渲染dom

//字符串处理

//DOM 操作都由 Vue 来处理

this.message = this.message.split('').reverse().join('')

}

}

----------------------------表单输入和应用状态之间的双向绑定

v-model指令

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

<input v-model="message">

模版语法：

message: 'Hello Vue!'

//你在input中输入什么文本值也会变化，双向绑定

--------------------------Vue 中注册组件 Vue 组件非常类似于自定义元素

DOM:

<ol>

<todo-item v-for="item in groceryList" v-bind:todo="item" v-bind:key="item.id" ></todo-item>

</ol>

模版语法：

data: {

groceryList: [

{ id: 0, text: '蔬菜' },

{ id: 1, text: '奶酪' },

{ id: 2, text: '随便其他什么人吃的东西' }

]

}

构建另一个组件模板:

// 定义名为 todo-item 的新组件

Vue.component('todo-item', {

props: ['todo'],//自定义属性 对应v-bind:todo

template: '<li>{{ todo.text }}</li>'

})

局部注册：

var Child = {

template: '<div>A custom component!</div>'

}

new Vue({

// ...

components: {

// <my-component> 将只在父模板可用

'my-component': Child

}

})

---------------------------扩展 Vue 构造器

//预定义选项 创建 可复用的组件构造器

var MyComponent = Vue.extend({

// 扩展选项

})

// 所有的 `MyComponent` 实例都将以预定义的扩展选项被创建

var myComponentInstance = new MyComponent()

------------------------------ v-if 根据条件展示元素

v-else 元素必须紧跟在 v-if 或者 v-else-if 元素的后面——否则它将不会被识别。

<div v-if="type === 'A'">

A

</div>

<div v-else-if="type === 'B'">

B

</div>

<div v-else-if="type === 'C'">

C

</div>

<div v-else>

Not A/B/C

</div>

用 key 管理可复用的元素 <label> 元素仍然会被高效地复用，因为它们没有添加 key 属性。

<template v-if="loginType === 'username'">

<label>Username</label>

<input placeholder="Enter your username" key="username-input">

</template>

<template v-else>

<label>Email</label>

<input placeholder="Enter your email address" key="email-input">

</template>

-------------------------------v-show 根据条件展示元素

<h1 v-show="ok">Hello!</h1>

不同的是带有 v-show 的元素始终会被渲染并保留在 DOM 中。v-show 是简单地切换元素的 CSS 属性 display 。

注意， v-show 不支持 <template> 语法，也不支持 v-else。

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；如果在运行时条件不太可能改变，则使用 v-if 较好。

当 v-if 与 v-for 一起使用时，v-for 具有比 v-if 更高的优先级。

------------------------------------------列表渲染 v-for

<ul id="example-1">

<li v-for="item in items">

{{ item.message }}

</li>

</ul>

var example1 = new Vue({

el: '#example-1',

data: {

items: [

{message: 'Foo' },

{message: 'Bar' }

]

}

})

在 v-for 块中，我们拥有对父作用域属性的完全访问权限:parentMessage。 v-for 还支持一个可选的第二个参数为当前项的索引 :index。下标从0开始，

你也可以用 of 替代 in 作为分隔符，因为它是最接近 JavaScript 迭代器的语法：

<div v-for="item of items"></div>

<ul id="example-2">

<li v-for="(item, index) in items">

{{ parentMessage }} - {{ index }} - {{ item.message }}

</li>

</ul>

var example2 = new Vue({

el: '#example-2',

data: {

parentMessage: 'Parent',

items: [

{ message: 'Foo' },

{ message: 'Bar' }

]

}

})

对象迭代 v-for 对象的属性来迭代

<ul id="repeat-object" class="demo">

<li v-for="value in object">

{{ value }}

</li>

</ul>

new Vue({

el: '#repeat-object',

data: {

object: {

firstName: 'John',

lastName: 'Doe',

age: 30

}

}

})

你也可以提供第二个的参数为键名：

<div v-for="(value, key) in object">

{{ key }} : {{ value }}

</div>

第三个参数为索引：

<div v-for="(value, key, index) in object">

{{ index }}. {{ key }} : {{ value }}

</div>

整数迭代 v-for

<div>

<span v-for="n in 10">{{ n }} </span>

</div>

组件 和 v-for

<div id="todo-list-example">

<input

v-model="newTodoText" // data属性

v-on:keyup.enter="addNewTodo" //确认按钮绑定方法

placeholder="Add a todo" //默认显示

>

<ul>

<li

is="todo-item" //组件名

v-for="(todo, index) in todos" 遍历的数组

v-bind:key="index"

v-bind:title="todo" // 对自定义属性进行data属性绑定赋值

v-on:remove="todos.splice(index, 1)" //监听remove方法

></li>

</ul>

</div>

Vue.component('todo-item', {

template: '\

<li>\

{{ title }}\ //data显示

<button v-on:click="$emit(\'remove\')">X</button>\ //移除自己

</li>\

',

props: ['title'] // 自定义属性

})

new Vue({

el: '#todo-list-example',

data: {

newTodoText: '',

todos: [

'Do the dishes',

'Take out the trash',

'Mow the lawn'

]

},

methods: {

addNewTodo: function () { //确认按钮绑定的添加方法

this.todos.push(this.newTodoText)

this.newTodoText = ''

}

}

})

v-for 的优先级比 v-if 更高，这意味着 v-if 将分别重复运行于每个 v-for 循环中。

而如果你的目的是有条件地跳过循环的执行，那么将 v-if 置于包装元素 (或 <template>)上。如:

<ul v-if="shouldRenderTodos">

<li v-for="todo in todos">

{{ todo }}

</li>

</ul>

key 它的工作方式类似于一个属性，所以你需要用 v-bind 来绑定动态值（在这里使用简写）：

建议尽可能使用 v-for 来提供 key

<div v-for="item in items" :key="item.id">

<!-- 内容 -->

</div>

---------------------------------------------------数组更新检测

变异方法

push() //向数组的末尾添加一个或多个元素，并返回新的长度。它直接修改 arrayObject，而不是创建一个新的数组

pop() //删除并返回数组的最后一个元素

shift() //删除并返回数组的第一个元素

unshift() //数组的开头添加一个或多个元素

splice() //删除元素，并向数组添加新元素。

sort() //对数组的元素进行排序

reverse() //颠倒数组中元素的顺序。

var ary = [1,2,3,4];

ary.splice(0,ary.length);//清空数组

重塑数组 非变异方法 不会改变原始数组，但总是返回一个新数组

filter(), concat(), slice()

替换数组中的值

Vue.set(example1.items, 下标, newValue)

example1.items.splice(下标, 1, newValue)

当你修改数组的长度时

example1.items.splice(newLength)

显示过滤/排序结果

显示一个数组的过滤或排序副本，而不实际改变或重置原始数据。在这种情况下，可以创建返回过滤或排序数组的计算属性。

<li v-for="n in evenNumbers">{{ n }}</li>

data: {

numbers: [ 1, 2, 3, 4, 5 ]

},

computed: {

evenNumbers: function () {

return this.numbers.filter(function (number) {

return number % 2 === 0

})

}

}

在计算属性不适用的情况下 (例如，在嵌套 v-for 循环中) 你可以使用一个 method 方法：

<li v-for="n in even(numbers)">{{ n }}</li>

data: {

numbers: [ 1, 2, 3, 4, 5 ]

},

methods: {

even: function (numbers) {

return numbers.filter(function (number) {

return number % 2 === 0

})

}

}

--------------------------------------------事件处理器

监听事件

v-on 指令监听 DOM 事件来触发一些 JavaScript 代码

<div id="example-1">

<button v-on:click="counter += 1">增加 1</button>

<p>这个按钮被点击了 {{ counter }} 次。</p>

</div>

var example1 = new Vue({

el: '#example-1',

data: {

counter: 0

}

})

方法事件处理器

许多事件处理的逻辑都很复杂，所以直接把 JavaScript 代码写在 v-on 指令中是不可行的。因此 v-on 可以接收一个定义的方法来调用。

<div id="example-2">

<!-- `greet` 是在下面定义的方法名 -->

<button v-on:click="greet">Greet</button>

</div>

var example2 = new Vue({

el: '#example-2',

data: {

name: 'Vue.js'

},

// 在 `methods` 对象中定义方法

methods: {

greet: function (event) {

// `this` 在方法里指当前 Vue 实例

alert('Hello ' + this.name + '!')

// `event` 是原生 DOM 事件

if (event) {

alert(event.target.tagName)

}

}

}

})

// 也可以用 JavaScript 直接调用方法

example2.greet() // -> 'Hello Vue.js!'

内联处理器方法

<div id="example-3">

<button v-on:click="say('hi')">Say hi</button>

<button v-on:click="say('what')">Say what</button>

</div>

new Vue({

el: '#example-3',

methods: {

say: function (message) {

alert(message)

}

}

})

或者：

<button v-on:click="warn('Form cannot be submitted yet.', $event)">

Submit

</button>

// ...

methods: {

warn: function (message, event) {

// 现在我们可以访问原生事件对象

if (event) event.preventDefault()

alert(message)

}

}

事件修饰符

通过由点(.)表示的指令后缀来调用修饰符。

//preventDefault() 方法阻止元素发生默认的行为

.stop

.prevent

.capture

.self

.once

<!-- 阻止单击事件冒泡 -->

<a v-on:click.stop="doThis"></a>

<!-- 提交事件不再重载页面 -->

<form v-on:submit.prevent="onSubmit"></form>

<!-- 修饰符可以串联 -->

<a v-on:click.stop.prevent="doThat"></a>

<!-- 只有修饰符 -->

<form v-on:submit.prevent></form>

<!-- 添加事件侦听器时使用事件捕获模式 -->

<div v-on:click.capture="doThis">...</div>

<!-- 只当事件在该元素本身（比如不是子元素）触发时触发回调 -->

<div v-on:click.self="doThat">...</div>

使用修饰符时，顺序很重要；相应的代码会以同样的顺序产生。因此，用 @click.prevent.self 会阻止所有的点击，而 @click.self.prevent 只会阻止元素上的点击。

//event.stopPropagation()停止事件的传播

键值修饰符

监听键盘事件时，我们经常需要监测常见的键值

<!-- 只有在 keyCode 是 13 时调用 vm.submit() -->

<input v-on:keyup.13="submit">

<!-- 同上 -->

<input v-on:keyup.enter="submit">

<!-- 缩写语法 -->

<input @keyup.enter="submit">

全部的按键别名：

.enter

.tab

.delete (捕获 “删除” 和 “退格” 键)

.esc

.space

.up

.down

.left

.right

可以通过全局 config.keyCodes 对象自定义键值修饰符别名：

修饰键

可以用如下修饰符开启鼠标或键盘事件监听，使在按键按下时发生响应。

.ctrl

.alt

.shift

.meta

-------------------------------------表单控件绑定

v-model 指令在表单控件元素上创建双向数据绑定。它会根据控件类型自动选取正确的方法来更新元素

v-model 会忽略所有表单元素的 value、checked、selected 特性的初始值。因为它会选择 Vue 实例数据来作为具体的值。你应该通过 JavaScript 在组件的 data 选项中声明初始值。

文本，input框，多行文本，复选框，单选按钮，选择列表

<select v-model="selected">

<option v-for="option in options" v-bind:value="option.value">

{{ option.text }}

</option>

</select>

<span>Selected: {{ selected }}</span>

new Vue({

el: '...',

data: {

selected: 'A',

options: [

{ text: 'One', value: 'A' },

{ text: 'Two', value: 'B' },

{ text: 'Three', value: 'C' }

]

}

})

value 到 Vue 实例的一个动态属性上，这时可以用 v-bind 实现，并且这个属性的值可以不是字符串。

<input

type="checkbox"

v-model="toggle"

v-bind:true-value="a"

v-bind:false-value="b"

>

// 当选中时

vm.toggle === vm.a

// 当没有选中时

vm.toggle === vm.b

修饰符

.lazy

在默认情况下， v-model 在 input 事件中同步输入框的值与数据 (除了 上述 IME 部分)，但你可以添加一个修饰符 lazy ，从而转变为在 change 事件中同步：

<!-- 在 "change" 而不是 "input" 事件中更新 -->

<input v-model.lazy="msg" >

.number

如果想自动将用户的输入值转为 Number 类型（如果原值的转换结果为 NaN 则返回原值），可以添加一个修饰符 number 给 v-model 来处理输入值：

<input v-model.number="age" type="number">

这通常很有用，因为在 type="number" 时 HTML 中输入的值也总是会返回字符串类型。

.trim

如果要自动过滤用户输入的首尾空格，可以添加 trim 修饰符到 v-model 上过滤输入：

<input v-model.trim="msg">

data 必须是函数

Vue.component('my-component', {

template: '<span>{{ message }}</span>',

data: {

message: 'hello'

}

})