# Vue.js

## 安装vue环境

由于 npm 安装速度慢，本教程使用了淘宝的镜像及其命令 cnpm

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

在用 Vue.js 构建大型应用时推荐使用 NPM 安装：

# 最新稳定版

$ cnpm install vue

# 全局安装 vue-cli

$ cnpm install --global vue-cli

创建并进入项目，安装并运行：

# 创建一个基于 webpack 模板的新项目

$ vue init webpack my-project

# 这里需要进行一些配置，默认回车Yes即可

$ cd my-project

$ cnpm install

$ cnpm run dev

## 模板语法

### 文本引用和html引用

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>Vue 测试实例 - 菜鸟教程(runoob.com)</title>

<script src="https://cdn.bootcss.com/vue/2.2.2/vue.min.js"></script>

</head>

<body>

<div id="app">

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

<div v-html="html\_message"></div>

</div>

<script>

new Vue({

el: '#app',

data: {

message: 'Hello Vue.js!',

html\_message:'<h1>Hello Vue.js!</h1>'

}

})

</script>

</body>

</html>

### 指令控制

<script src="https://cdn.bootcss.com/vue/2.2.2/vue.min.js"></script>

<div id="app">

<p v-if="seen">现在你看到我了</p>

<template v-if="ok">

<h1>菜鸟教程</h1>

<p>学的不仅是技术，更是梦想！</p>

<p>哈哈哈，打字辛苦啊！！！</p>

</template>

</div>

<script>

new Vue({

el: '#app',

data: {

seen: true,

ok: true

}

})

</script>

### 参数绑定（对html标签内的原有属性进行绑定）

<script src="https://cdn.bootcss.com/vue/2.2.2/vue.min.js"></script>

<div id="app">

<pre><a v-bind:href="url">菜鸟教程</a></pre>

</div>

<script>

new Vue({

el: '#app',

data: {

url: 'http://www.runoob.com'

}

})

</script>

### 数据双向绑定

<div id="app">

{{message}}

<input type="text" v-model="message" />

</div>

<script>

new Vue({

el:"#app",

data:{

message:"111"

}

})

</script>

### 事件绑定

<div id="app">

{{message}}

<button v-on:click="reservemessage">字符串翻转</button>

</div>

<script>

new Vue({

el:"#app",

data:{

message:"123"

},

methods:{

reservemessage:function(){

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

}

}

})

</script>

### 参数过滤处理

<div id="app">

{{ message | capitalize("arg1") }}

</div>

<script>

new Vue({

el: '#app',

data: {

message: 'runoob'

},

filters: {

capitalize: function(value,arg1) {

alert(arg1)

if (!value) return ''

value = value.toString()

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

}

}

})

</script>

过滤器可以串联：

{{ message | filterA | filterB }}

过滤器是 JavaScript 函数，因此可以接受参数：

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

这里，message 是第一个参数，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

<!-- 完整语法 -->

<a v-bind:href="url"></a>

<!-- 缩写 -->

<a :href="url"></a>

v-on 缩写

<!-- 完整语法 -->

<a v-on:click="doSomething"></a>

<!-- 缩写 -->

<a @click="doSomething"></a>

### Vue实例

<div id="app">

{{message}}

<input type="text" v-model="message" />

</div>

<script>

var data = {message:"这是message"};

var v = new Vue({

el:"#app",

data:data

})

v.$watch('message',function(newVal,oldVal){

alert("从 "+oldVal+" 变成了 "+newVal)

})

</script>

每个 Vue 实例都会代理其 data 对象里所有的属性：

var data = { a: 1 }

var vm = new Vue({

data: data

})

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

// 设置属性也会影响到原始数据

vm.a = 2

data.a // -> 2

// ... 反之亦然

data.a = 3

vm.a // -> 3

Vue中加上$属性区分

var data = { a: 1 }

var vm = new Vue({

el: '#example',

data: data

})

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

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

// $watch 是一个实例方法

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

// 这个回调将在 `vm.a` 改变后调用})

## 条件语句

### v-if v-else

<div id="app">

<div v-if="message[0].number > 0.5">

{{message[0].number}}

</div>

<div v-else-if="message[1].number > 2.1">

{{message[1].number}}

</div>

<div v-else>

{{message[2].number}}

</div>

</div>

<script>

new Vue({

el: '#app',

data:{

message:[{number:0.4},{number:2},{number:3}]

}

})

</script>

### v-show

<div id="app">

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

</div>

<script>

new Vue({

el: '#app',

data: {

ok: true

}

})

</script>

## 循环语句

### 迭代对象

<div id="app">

<ul>

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

{{ value }}

</li>

</ul>

</div>

<script>

new Vue({

el: '#app',

data: {

object: {

name: '菜鸟教程',

url: 'http://www.runoob.com',

slogan: '学的不仅是技术，更是梦想！'

}

}

})

</script>

### 迭代数组

<div id="app">

<ol>

<li v-for="site in sites">

{{ site.name }}

</li>

</ol>

</div>

<script>

new Vue({

el: '#app',

data: {

sites: [{name: 'Runoob'}, {name: 'Google'}, {name: 'Taobao'}]

}

})

</script>

键值和value值和索引

<div id="app">

<ul>

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

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

</li>

</ul>

</div>

### 迭代number

<div id="app">

<ul>

<li v-for="n in 10">

{{ n }}

</li>

</ul>

</div>

<script>

new Vue({

el: '#app'

})

</script>

## 计算属性

### computed和methods

但是 computed 是基于它的依赖缓存，只有相关依赖发生改变时才会重新取值。而使用 methods ，在重新渲染的时候，函数总会重新调用执行。

<div id="app">

<p>原始字符串: {{ message }}</p>

<p>计算后反转字符串: {{ reversedMessage }}</p>

<p>使用方法后反转字符串: {{ reversedMessage2() }}</p>

</div>

<script>

var vm = new Vue({

el: '#app',

data: {

message: 'Runoob!'

},

computed: {

reversedMessage: function() {

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

}

},

methods: {

reversedMessage2: function() {

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

}

}

})

</script>

### get和set

<div id="app">

<p>{{ site }}</p>

</div>

<script>

var vm = new Vue({

el: '#app',

data: {

name: 'Google',

url: 'http://www.google.com'

},

computed: {

site: {

// getter

get: function() {

return this.name + ' ' + this.url

},

// setter

set: function(newValue) {

alert(newValue)

var names = newValue.split(' ')

this.name = names[0]

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

}

}

}

})

// 调用 setter， vm.name 和 vm.url 也会被对应更新

vm.site = '菜鸟教程 http://www.runoob.com';

document.write('name: ' + vm.name);

document.write('<br>');

document.write('url: ' + vm.url);

</script>

## 样式绑定

### class属性绑定

当classstate为true时使用样式 否则不使用

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>Vue 测试实例 - 菜鸟教程(runoob.com)</title>

</head>

<style>

.classstyle {

background: #444;

color: #eee;

}

</style>

<body>

<script src="https://cdn.bootcss.com/vue/2.2.2/vue.min.js"></script>

<div id="app">

<label for="r1">修改颜色</label>

<input type="checkbox" v-model="classstate" id="r1">

<br>

<br>

<div v-bind:class="{'classstyle': classstate}">

directiva v-bind:class

</div>

</div>

<script>

new Vue({

el: '#app',

data: {

classstate: false

}

});

</script>

</body>

</html>

### 控制样式

<style>

.text-danger {

width: 100px;

height: 100px;

background: red;

}

.active {

width: 100px;

height: 100px;

background: green;

}

</style>

</head>

<body>

<div id="app">

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

</div>

<script>

new Vue({

el: '#app',

data: {

isActive: false,

activeClass: 'active',

errorClass: 'text-danger'

},

computed: {

controlClass: function() {

if (this.isActive) return this.activeClass

return this.errorClass

}

}

})

</script>

</body>

### 内联样式

①

<div id="app">

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

</div>

<script>

new Vue({

el: '#app',

data: {

activeColor: 'green',

fontSize: 30

}

})

</script>

②

<div id="app">

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

</div>

<script>

new Vue({

el: '#app',

data: {

styleObject: {

color: 'green',

fontSize: '30px'

}

}

})

</script>

## 事件处理器

### 点击调用和js内调用

<div id="app">

<!-- `greet` 是在下面定义的方法名 -->

<button v-on:click="greet('666')">Greet</button>

</div>

<script>

var app = new Vue({

el: '#app',

data: {

name: 'Vue.js'

},

// 在 `methods` 对象中定义方法

methods: {

greet: function(message,event) {

// `this` 在方法里指当前 Vue 实例

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

// `event` 是原生 DOM 事件

if (event) {

alert(event.target.tagName)

}

}

}

})

// 也可以用 JavaScript 直接调用方法

app.greet('666')

</script>

## 表单

### 多选

<div id="app">

<p>单个复选框：</p>

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

<label for="checkbox">{{ checked }}</label>

<p>多个复选框：</p>

<input type="checkbox" id="runoob" value="Runoob" v-model="checkedNames">

<label for="runoob">Runoob</label>

<input type="checkbox" id="google" value="Google" v-model="checkedNames">

<label for="google">Google</label>

<input type="checkbox" id="taobao" value="Taobao" v-model="checkedNames">

<label for="taobao">taobao</label>

<br>

<span>选择的值为: {{ checkedNames }}</span>

</div>

<script>

new Vue({

el: '#app',

data: {

checked: false,

checkedNames: []

}

})

</script>

### 单选

<div id="app">

<input type="radio" id="runoob" value="Runoob" v-model="picked">

<label for="runoob">Runoob</label>

<br>

<input type="radio" id="google" value="Google" v-model="picked">

<label for="google">Google</label>

<br>

<span>选中值为: {{ picked }}</span>

</div>

<script>

new Vue({

el: '#app',

data: {

picked: 'Runoob'

}

})

</script>

### 下拉

<div id="app">

<select v-model="selected" name="fruit">

<option value="">选择一个网站</option>

<option value="www.runoob.com">Runoob</option>

<option value="www.google.com">Google</option>

</select>

<div id="output">

选择的网站是: {{selected}}

</div>

</div>

<script>

new Vue({

el: '#app',

data: {

selected: ''

}

})

</script>

## 组件

### 全局组件

<div id="app">

<runoob></runoob>

</div>

<script>

// 注册

Vue.component('runoob', {

template: '<h1>自定义组件!</h1>'

})

// 创建根实例

new Vue({

el: '#app'

})

</script>

### 局部组件

<div id="app">

<runoob></runoob>

</div>

<script>

var Child = {

template: '<h1>自定义组件!</h1>'

}

// 创建根实例

new Vue({

el: '#app',

components: {

// <runoob> 将只在父模板可用

'runoob': Child

}

})

</script>

### Props

<div id="app">

<child message="hello!"></child>

</div>

<script>

// 注册

Vue.component('child', {

// 声明 props

props: ['message'],

// 同样也可以在 vm 实例中像 “this.message” 这样使用

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

})

// 创建根实例

new Vue({

el: '#app'

})

</script>

### 动态props

<div id="app">

<div>

<input v-model="parentMsg">

<br>

<child v-bind:message="parentMsg"></child>

</div>

</div>

<script>

// 注册

Vue.component('child', {

// 声明 props

props: ['message'],

// 同样也可以在 vm 实例中像 “this.message” 这样使用

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

})

// 创建根实例

new Vue({

el: '#app',

data: {

parentMsg: '父组件内容'

}

})

</script>

### 组件数组循环

<div id="app">

<ol>

<todo-item v-for="item in sites" v-bind:todo="item"></todo-item>

</ol>

</div>

<script>

Vue.component('todo-item', {

props: ['todo'],

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

})

new Vue({

el: '#app',

data: {

sites: [{

text: 'Runoob'

}, {

text: 'Google'

}, {

text: 'Taobao'

}]

}

})

</script>

### 自定义事件

<div id="app">

<div id="counter-event-example">

<p>{{ total }}</p>

<button-counter v-on:increment="incrementTotal"></button-counter>

<button-counter v-on:increment="incrementTotal"></button-counter>

</div>

</div>

<script>

Vue.component('button-counter', {

template: '<button v-on:click="increment">{{ counter }}</button>',

data: function() {

return {

counter: 0

}

},

methods: {

increment: function() {

this.counter += 1

this.$emit('increment')

}

},

})

new Vue({

el: '#counter-event-example',

data: {

total: 0

},

methods: {

incrementTotal: function() {

this.total += 1

}

}

})

</script>

## 自定义指令

### 自动获焦

<div id="app">

<p>页面载入时，input 元素自动获取焦点：</p>

<input v-focus>

</div>

<script>

// 注册一个全局自定义指令 v-focus

Vue.directive('focus', {

// 当绑定元素插入到 DOM 中。

inserted: function(el) {

// 聚焦元素

el.focus()

}

})

// 创建根实例

new Vue({

el: '#app'

})

</script>

### 钩子函数

<div id="app">

<div v-runoob="{ color: 'green', text: '菜鸟教程!' }"></div>

</div>

<script>

Vue.directive('runoob', function(el, binding) {

// 简写方式设置文本及背景颜色

el.innerHTML = binding.value.text

el.style.backgroundColor = binding.value.color

})

new Vue({

el: '#app'

})

</script>