**兼容性**

Vue **支持** IE9 及上版本，因为 Vue 使用了 IE8 无法模拟的 ECMAScript 5 特性。但它支持所有**[兼容 ECMAScript 5 的浏览器](https://caniuse.com/" \l "feat=es5" \t "_blank)**。

因为它使用的是对象的getter,setter。这些特性ie9以上才支持，其他非ie系浏览器基本都支持

**数据绑定**

**绑定文本**：{{绑定键名}}

**绑定属性**：v-bind:属性名=” 绑定键名” ，它有一种缩写：“:属性名 = 绑定键名”

**绑定DOM**：v-if=”绑定键名(必须是boolean类型)”

**组件化**

在一个大型的应用中，不同的页面中肯定有重复的内容，那么将这些重复的内容封装成组件，也就更便于维护了。以下是创建一个组件的例子：

JS文件：

Vue.component('todo-item', {

props: ['todo'], //todo是传入这个组件的参数

template: '<li>{{ todo }}</li>' //模板

})

var app7 = new Vue({

el: '#app-7',

data: {

groceryList: [

'蔬菜' ,

'奶酪' ,

'随便其它什么人吃的东西',

]

}

})

HTML文件：

<!--

v-bind:todo传递参数

-->

<todo-item v-for="item in groceryList"

v-bind:todo="item">

</todo-item>

**选项/数据**

当我们创建一个Vue实例时，你可以传入一个选项对象，下面对它们一一介绍：

**#data**

**类型：Object | Function**

data是Vue实例的数据对象。Vue会将data属性中的值转成getter/setter，从而让data的值能够响应数据的变化。值得注意的是只有当实例被创建时data中存在的属性才是响应式的。也就是说如果在页面加载完毕后新增一个属性就不会触发更新。解决方法是，将那些一开始值为空的属性设置为初始值。

实例创建之后，可以通过 vm.$data 访问原始数据对象。Vue 实例也代理了 data 对象上所有的属性，因此访问 vm.a 等价于访问 vm.$data.a。

当一个**组件**被定义，data 必须声明为返回一个初始数据对象的函数，因为组件可能被用来创建多个实例。如果 data 仍然是一个纯粹的对象，则所有的实例将**共享引用**同一个数据对象！通过提供 data 函数，每次创建一个新实例后，我们能够调用 data 函数，从而返回初始数据的一个全新副本数据对象。

**#props**

**类型：Array<string> | Object**

props 可以是数组或对象，用于接收来自父组件的数据。props 可以是简单的数组，或者使用对象作为替代，对象允许配置高级选项，如类型检测、自定义校验和设置默认值。

**#methods**

**类型：{key<string> : Function}**

可以用v-on指令监听DOM事件，该函数的第一个参数为event对象。

**#watch**

**类型：{key<string> : string | Function | Object | Array }**

可以用watch观察选项的Data对象，回调函数的参数可以是新值和旧值

**计算属性与监听器**

当我们需要对普通属性中值进行一些复杂的逻辑运算时，可以使用computed(计算属性)来绑定普通属性。

计算属性的结果会被缓存，除非普通属性发生改变需要重新计算。计算属性相当于声明了一个getter函数，使它更易于维护和理解。

例如：HTML文件：

<tr>

<td>用户名：</td>

<td><input type="text" v-model="UserName"/></td>

<td :class="ValidUserName.font">**{{**ValidUserName.msg**}}**</td>

</tr>

JS文件：

data: {

UserName: "",

},

computed: {

ValidUserName: function () {

if (this.UserName.length <= 0) {

return {

state: 0,

font:"font-red",

msg : "用户名不能为空"

};

}}

当需要在数据变化时执行异步或开销较大的操作时，我们需要自定义一个监听器，Vue通过watch选项提供了一个通用的方法来响应数据。

**Class和Style绑定**

绑定HTML Class

我们可以传给v-bind:class一个对象，以动态切换class，至于里面的class是否显示，则又bool值来确定。它有两种显示设置方式：

1. 在HTML中放置键值对

<input type=”button” v-bind:class=”{Active:true}”/>

* 转化为HTML：<input type=”button” class=”Active”/>

1. 在HTML中方式对象键

<input type="button" v-bind:class="classObj">

JS:

…

data:{

classObj:{

bg\_red:true

}

}

…

绑定内联样式

我们可以用v-bind:style来绑定CSS，CSS 属性名可以用驼峰式 (camelCase) 或短横线分隔 (kebab-case，记得用单引号括起来) 来命名。但不推荐这种写法。

流程控制

v-if v-else-if v-else

它们的用法和其他语言的if...else...if…else类似

<span v-if="hour >= 5 && hour < 12">早晨好</span>

<span v-else-if="hour >= 12 && hour < 19">下午好</span>

<span v-else-if="(hour >= 19 && hour < 24) || (hour < 5 && hour >= 0)">晚上好</span>

<span v-else>输出错误</span>

如果需要包裹多个元素，可以使用<template>标签

<template v-if="hour >= 5 && hour < 12">

早晨好

</template>

<template v-else-if="hour >= 12 && hour < 19">

下午好

</template>

<template v-else-if="(hour >= 19 && hour < 24) || (hour < 5 && hour >= 0)">

晚上好

</template>

<template v-else>

输出错误

</template>

v-show

另一个用于控制显示的标签是**v-show**

<span v-show="hour >= 5 && hour < 12">早晨好</span>

<span v-show="hour >= 12 && hour < 19">下午好</span>

<span v-show="(hour >= 19 && hour < 24) || (hour < 5 && hour >= 0)">晚上好</span>

与v-if不同的是，v-show是通过修改元素的display属性来控制显示，而if则是通过DOM来控制

列表渲染

v-for

Vue中我们可以用v-for对一组数组或对象进行遍历。语法格式为item in arr

遍历数组时，最大支持两个参数：值和索引。

遍历对象时，最大支持三个参数：值，键名，索引。

以下是数据源：

data: {

Student:

[

{

ID:"1",

FirstName: "Wang",

LastName: "Di",

Age: 12,

},

{

ID:2,

FirstName: "Feng",

LastName: "Yan",

Age: 12,

},

]

}

以下是HTML代码：

<div v-for="(item, index) in Student" :key="item.ID"> <!--遍历数组-->

**{{** index **}}**.

<span v-for="(value,key,index) in item"> <!--遍历对象-->

**{{**value**}}**

</span>

</div>

为了给 Vue 一个提示，以便它能跟踪每个节点的身份，从而重用和重新排序现有元素，你需要为每项提供一个唯一 key 属性。理想的 key 值是每项都有的且唯一的 id。但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。

建议尽可能在使用 v-for 时提供 key，除非遍历输出的 DOM 内容非常简单，或者是刻意依赖默认行为以获取性能上的提升。

你可以对数组调用JS的数组方法，这些改变会立即反应到视图中。

常用的方法如下：

|  |  |  |
| --- | --- | --- |
| 方法名 | 作用 | 例子 |
| push(obj1…objN) | 往数组尾部添加元素 |  |
| pop() | 从数组尾部删除元素 |  |
| unshift(obj1…objN) | 往数组头部添加元素 |  |
| shift() | 从数组头部删除元素 |  |
| Splice(index,howMany,obj1…objN) | 向数组中添加或删除元素，index表示插入位置，howMany表示删除数量，obj表示添加的素 |  |
| Sort(Func) | 对数组进行排序，如果调用该方法时没有使用参数，将按字母顺序对数组中的元素进行排序，说得更精确点，是按照字符编码的顺序进行排序。要实现这一点，首先应把数组的元素都转换成字符串（如有必要），以便进行比较。  如果想按照其他标准进行排序，就需要提供比较函数，该函数要比较两个值 | app.Student.sort(function(a,b){  return a.Age< b.Age;  });  //对Student对象进行排序，以Age倒序排序。 |
| Reverse() | 对数组进行倒排 |  |
| filter (Func) | 返回数组中的满足回调函数中指定的条件的元素。 | example1.items = example1.items.filter(function (item) {  return item.message.match(/Foo/) }) |
| Concat(arr1…arrN) | 连接两个或多个数组。 |  |
| Slice(startIndex,endIndex) | 从已有的数组中返回选定的元素。 |  |

当你需要修改数组的某一个元素时，虽然能修改成功，但是不能反应到视图中去。

例如：我想修改Student里面第一个元素

App.Student[0] = {FirstName:”He”,LastName:”Sheng”,Age:40,ID:1}

这时你会发现视图并没有改变，但在控制台上输出App .Student[0]这个时，却发现值已经改过来了。

那么怎么解决这个问题呢？

答案是使用Vue提供的Set方法，就拿上面的例子来说。

Vue.set(app.Student,0, {FirstName:”He”,LastName:”Sheng”,Age:40,ID:1})

这样就能成功的反应到视图中去了。

和v-if一样，我们可以用<template>来包裹多个需要循环的元素

事件处理

v-on

我们可以用v-on来监听事件，后面跟上 “:事件名称”决定监听哪个事件，在赋值语句中声明可以带上方法名或内联语句。

它还有一种缩写格式：@

如：

方法名<input type="button" value="验证" v-on:click.stop="Valid" />

内联语句：<input type="button" value="验证" v-on:click.stop="Valid($event)" />

我们在一些时候需要阻止默认事件或停止事件冒泡，这是我们常常会用到event.preventDefault() 或 event.stopPropagation()，Vue帮我们封装了这些方法，只需要在事件名称后面添加.prevnet或.stop即可。

如：

<div id="app" v-on:click="FormClick">

<form>

<!--

.stop可以停止事件的冒泡，这样就不会触发Form的FormClick事件了

-->

<input type="button" value="验证" v-on:click.stop="Valid" />

<!--

.prevent阻止默认行为，表单不会提交了

-->

<input type="submit" v-on:click.prevent="Submit" />

</form>

</div>

在2.1.4版本中还新增了**.once**修饰符，它表示只触发一次事件处理函数。

在2.3.0中新增了**.passive**修饰符，它对应addEventListener中的passive选项，**能够提高移动端的性能。**它表示事件处理函数永远不会调用preventDefault()。

[**Passive事件监听器**](https://www.cnblogs.com/ziyunfei/p/5545439.html)

在监听键盘事件时，我们经常要检查常见的键值，一般通过keyCode来判断点击了哪个按钮。Vue中定义了一些常用的按键修饰符如：.enter .tab .esc .space .left .right .delete 等。

[为什么在 HTML 中监听事件?](https://cn.vuejs.org/v2/guide/events.html#%E4%B8%BA%E4%BB%80%E4%B9%88%E5%9C%A8-HTML-%E4%B8%AD%E7%9B%91%E5%90%AC%E4%BA%8B%E4%BB%B6)

你可能注意到这种事件监听的方式违背了关注点分离 (separation of concern) 这个长期以来的优良传统。但不必担心，因为所有的 Vue.js 事件处理方法和表达式都严格绑定在当前视图的 ViewModel 上，它不会导致任何维护上的困难。实际上，使用 v-on 有几个好处：

1. 扫一眼 HTML 模板便能轻松定位在 JavaScript 代码里对应的方法。
2. 因为你无须在 JavaScript 里手动绑定事件，你的 ViewModel 代码可以是非常纯粹的逻辑，和 DOM 完全解耦，更易于测试。
3. 当一个 ViewModel 被销毁时，所有的事件处理器都会自动被删除。你无须担心如何清理它们。

表单输入绑定

我们可以将v-model用在input,textarea,select元素上面创建双向绑定。它的本质是监听用户的输入事件以及更新数据。常见的表单控件包括：文本输入框，多行文本，单选按钮，多选按钮，下拉菜单等。

我们也可以用一些修饰符来对它进行一些细节处理：

.lazy => 在change时更新而非input时

.number => 自动将用户输入的值转成数值类型，无法转换则返回原始值

.trim => 自动过滤收尾空格字符

组件基础

组件系统是Vue的重要概念，它允许我们使用小型，独立的可复用组件构造大型应用。在Vue中，一个组件的本质是一个拥有预定义选项的一个Vue实例。

例如：  
最简单的Demo，Basic.html：

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<title></title>

</head>

<body>

<div id="app">

<base-userregister></base-userregister>

</div>

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

<script src="../../../Scripts/vue/Component/Basic.js"></script>

</body>

</html>

定义组件，生成Vue对象，Basic.js：

Vue.component("base-userregister",

{

data: function () {

return {

UserName: "111111",

Password:"hello world"

};

},

methods: {

SubmitClick: function () {

console.log("UserName:"+this.UserName);

}

},

template: `<div>

<table>

<tr>

<td><label>用户名：</label></td>

<td><input type="text" v-model="UserName"/></td>

</tr>

<tr>

<td><label>密码：</label></td>

<td><input type="password" v-model="Password"/></td>

</tr>

<tr>

<td colspan="2"><input type="submit" value="提交" v-on:click='SubmitClick'></td>

</tr>

</table>

</div>`

});

new Vue({

el: "#app",

});

这样就把用户名和密码做成了一个组件。

注意：与普通的Vue对象相比，**组件的data选项是一个Func**，这是为了让多个相同类型的组件不会互相影响

**父框架向组件传递数据**，我们先在组件中注册props选项，然后在把数据通过属性传递到组件中。

例如：

Basic.html

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<title></title>

</head>

<body>

<div id="app">

<base-bbspost title="中美贸易战" ></base-bbspost>

</div>

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

<script src="../../../Scripts/vue/Component/Basic.js"></script>

</body>

</html>

Basic.js

Vue.component("base-bbspost", {

data: function () {

return {};

},

props: {

title: String,

message: String

},

template: `<div>

<h2>{{title}}</h2>

<div>{{message}}</div>

</div>`

});

new Vue({

el: "#app",

});

代码规范：props应该尽可能详细，至少需要指定其类型。它有两个好处：它们写明了组件的API所以很容易看懂组件的用法；在开发环境下，如果向组件提供格式不正确的prop，Vue会告警，以帮助你捕获潜在的错误来源。

反例：

// 这样做只有开发原型系统时可以接受  
props: ['status']

好例子：

props: {  
 status: String  
}

// 更好的做法！  
props: {  
 status: {  
 type: String,  
 required: true,  
 validator: function (value) {  
 return [  
 'syncing',  
 'synced',  
 'version-conflict',  
 'error'  
 ].indexOf(value) !== -1  
 }  
 }  
}

**组件也可以通过注册一个事件供上级调用，并传值。**我们可以通过$emit方法传入事件名称和参数

Basic.html：

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<title></title>

</head>

<body>

<div id="app">

<div v-bind:style="{fontSize:fontSize + 'px'}">达达兔论坛</div>

<base-changefont v-on:font-change="fontChange"></base-changefont>

</div>

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

<script src="../../../Scripts/vue/Component/Basic.js"></script>

</body>

</html>

Basic.js：

Vue.component("base-changefont", {

data: function () {

return {

fontSize: [10, 12, 14, 16, 18, 20],

font:10

};

},

methods: {

fontChange: function (event) {

//调用父控件的方法

this.$emit('font-change', this.font);

}

},

template: `

<div>

<label>字体大小：</label>

<select v-on:change="fontChange" v-model="font">

<option v-for='item in fontSize'>{{item}}</option>

</select>

<label>px</label>

</div>

`

});

new Vue({

el: "#app",

data: {

fontSize:10

},

methods: {

fontChange: function (size) {

this.fontSize = size;

}

}

});