**[Vue.js——60分钟快速入门](https://www.cnblogs.com/rik28/p/6024425.html)**

[**Vue.js**](https://github.com/vuejs/vue)**是当下很火的一个JavaScript MVVM库，它是以数据驱动和组件化的思想构建的。相比于Angular.js，Vue.js提供了更加简洁、更易于理解的API，使得我们能够快速地上手并使用Vue.js。**

本文摘自：http://www.cnblogs.com/keepfool/p/5619070.html

如果你之前已经习惯了用jQuery操作DOM，学习Vue.js时请先抛开手动操作DOM的思维，因为Vue.js是数据驱动的，你无需手动操作DOM。它通过一些特殊的HTML语法，将DOM和数据绑定起来。一旦你创建了绑定，DOM将和数据保持同步，每当变更了数据，DOM也会相应地更新。

当然了，在使用Vue.js时，你也可以结合其他库一起使用，比如jQuery。

本文的Demo和源代码已放到GitHub，如果您觉得本篇内容不错，请点个赞，或在GitHub上加个星星！

[v-for Demo](https://keepfool.github.io/vue-tutorials/01.GettingStarted/v-for.html) [v-bind Demo](https://keepfool.github.io/vue-tutorials/01.GettingStarted/v-bind.html) [Page Demo](https://keepfool.github.io/vue-tutorials/01.GettingStarted/simple-demo.html) [GitHub Source](https://github.com/keepfool/vue-tutorials" \t "_blank)

**MVVM模式**

下图不仅概括了MVVM模式（Model-View-ViewModel），还描述了在Vue.js中ViewModel是如何和View以及Model进行交互的。

**ViewModel是Vue.js的核心，它是一个Vue实例。**Vue实例是作用于某一个HTML元素上的，这个元素可以是HTML的body元素，也可以是指定了id的某个元素。

当创建了ViewModel后，双向绑定是如何达成的呢？

首先，我们将上图中的DOM Listeners和Data Bindings看作两个工具，它们是实现双向绑定的关键。  
从View侧看，ViewModel中的DOM Listeners工具会帮我们监测页面上DOM元素的变化，如果有变化，则更改Model中的数据；  
从Model侧看，当我们更新Model中的数据时，Data Bindings工具会帮我们更新页面中的DOM元素。

**Hello World示例**

了解一门语言，或者学习一门新技术，编写Hello World示例是我们的必经之路。  
这段代码在画面上输出"Hello World!"。

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title></title>

</head>

<body>

<!--这是我们的View-->

<div id="app">

{{ message }}

</div>

</body>

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

<script>

// 这是我们的Model

var exampleData = {

message: 'Hello World!'

}

// 创建一个 Vue 实例或 "ViewModel"

// 它连接 View 与 Model

new Vue({

el: '#app',

data: exampleData

})

</script>

</html>

使用Vue的过程就是定义MVVM各个组成部分的过程的过程。

1. **定义View**
2. **定义Model**
3. **创建一个Vue实例或"ViewModel"，它用于连接View和Model**

在创建Vue实例时，需要传入一个**选项对象**，选项对象可以包含数据、挂载元素、方法、模生命周期钩子等等。

在这个示例中，**选项对象**的**el属性**指向View，el: '#app'表示该Vue实例将挂载到<div id="app">...</div>这个元素；**data属性**指向Model，data: exampleData表示我们的Model是exampleData对象。  
Vue.js有多种数据绑定的语法，最基础的形式是文本插值，使用一对大括号语法，在运行时{{ message }}会被数据对象的message属性替换，所以页面上会输出"Hello World!"。

Vue.js已经更新到2.0版本了，但由于还不是正式版，本文的代码都是1.0.25版本的。

**双向绑定示例**

MVVM模式本身是实现了双向绑定的，在Vue.js中可以使用v-model指令在表单元素上创建双向数据绑定。

<!--这是我们的View-->

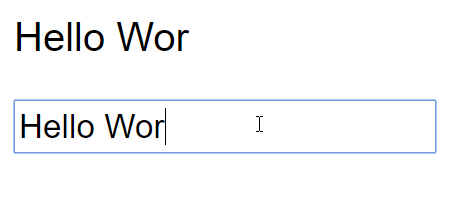
<div id="app">

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

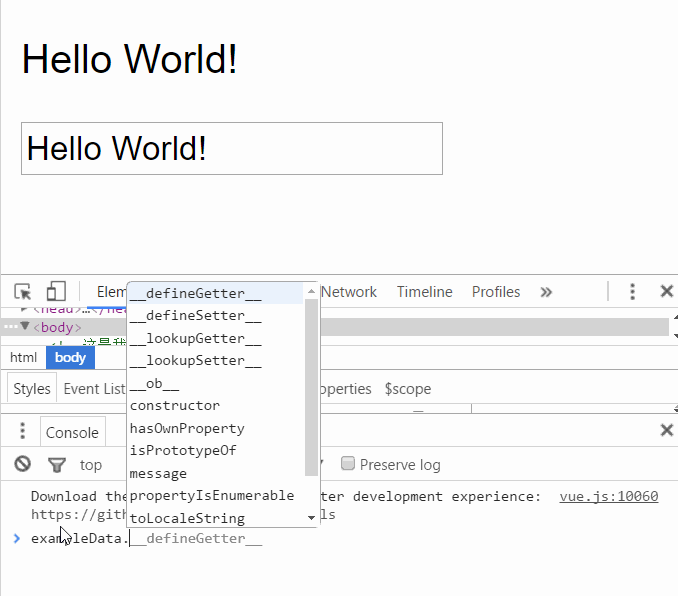
<input type="text" v-model="message"/>

</div>

将message绑定到文本框，当更改文本框的值时，<p>{{ message }}</p> 中的内容也会被更新。

[](http://images2015.cnblogs.com/blog/341820/201606/341820-20160627065309718-2118753556.gif)

反过来，如果改变message的值，文本框的值也会被更新，我们可以在Chrome控制台进行尝试。

[](http://images2015.cnblogs.com/blog/341820/201606/341820-20160627065312015-708112076.gif)

Vue实例的data属性指向exampleData，它是一个引用类型，改变了exampleData对象的属性，同时也会影响Vue实例的data属性。

**Vue.js的常用指令**

上面用到的v-model是Vue.js常用的一个指令，那么指令是什么呢？

**Vue.js的指令是以v-开头的，它们作用于HTML元素，指令提供了一些特殊的特性，将指令绑定在元素上时，指令会为绑定的目标元素添加一些特殊的行为，我们可以将指令看作特殊的HTML特性（attribute）。**

Vue.js提供了一些常用的内置指令，接下来我们将介绍以下几个内置指令：

* v-if指令
* v-show指令
* v-else指令
* v-for指令
* v-bind指令
* v-on指令

Vue.js具有良好的扩展性，我们也可以开发一些自定义的指令，后面的文章会介绍自定义指令。

**v-if指令**

v-if是条件渲染指令，它根据表达式的真假来删除和插入元素，它的基本语法如下：

v-if="*expression*"

expression是一个返回bool值的表达式，表达式可以是一个bool属性，也可以是一个返回bool的运算式。例如：

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title></title>

</head>

<body>

<div id="app">

<h1>Hello, Vue.js!</h1>

<h1 v-if="yes">Yes!</h1>

<h1 v-if="no">No!</h1>

<h1 v-if="age >= 25">Age: {{ age }}</h1>

<h1 v-if="name.indexOf('jack') >= 0">Name: {{ name }}</h1>

</div>

</body>

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

<script>

var vm = new Vue({

el: '#app',

data: {

yes: true,

no: false,

age: 28,

name: 'keepfool'

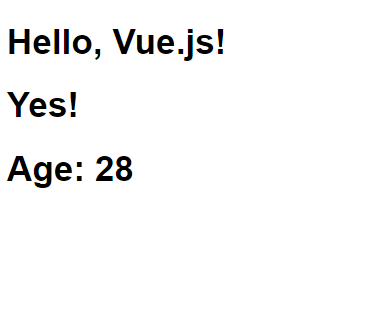
}

})

</script>

</html>

**注意：**yes, no, age, name这4个变量都来源于Vue实例选项对象的data属性。

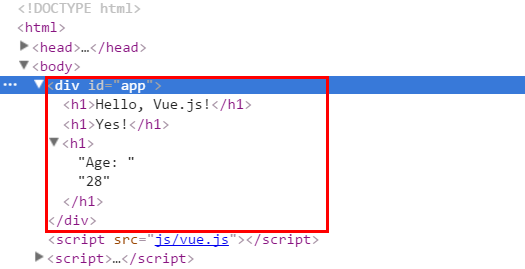
[](http://images2015.cnblogs.com/blog/341820/201606/341820-20160627065313781-1005102718.png)

这段代码使用了4个表达式：

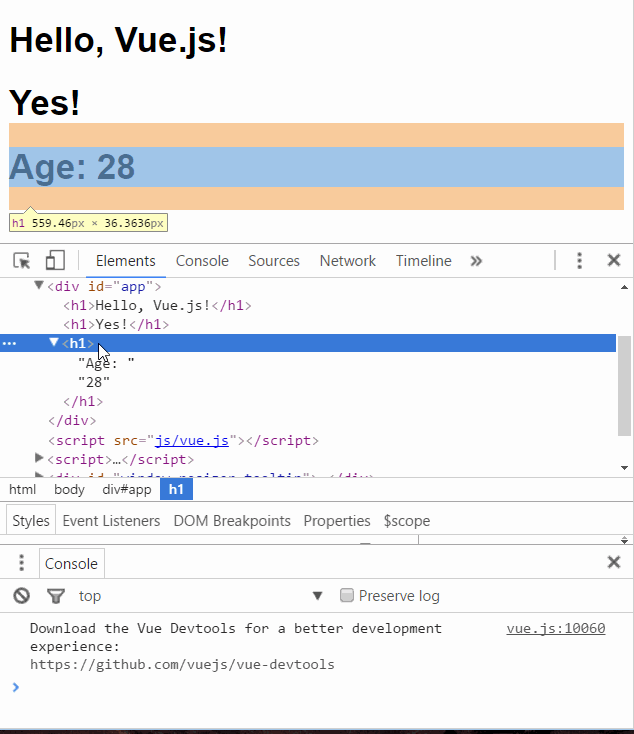
* 数据的yes属性为true，所以"Yes!"会被输出；
* 数据的no属性为false，所以"No!"不会被输出；
* 运算式age >= 25返回true，所以"Age: 28"会被输出；
* 运算式name.indexOf('jack') >= 0返回false，所以"Name: keepfool"不会被输出。

**注意：**v-if指令是根据条件表达式的值来执行**元素的插入或者删除行为。**

这一点可以从渲染的HTML源代码看出来，面上只渲染了3个<h1>元素，v-if值为false的<h1>元素没有渲染到HTML。

[](http://images2015.cnblogs.com/blog/341820/201606/341820-20160627065315515-1776406933.png)

为了再次验证这一点，可以在Chrome控制台更改age属性，使得表达式age >= 25的值为false，可以看到<h1>Age: 28</h1>元素被删除了。

[](http://images2015.cnblogs.com/blog/341820/201606/341820-20160627065317577-1183006666.gif)

age是定义在选项对象的data属性中的，为什么Vue实例可以直接访问它呢？  
这是因为**每个Vue实例都会代理其选项对象里的data属性。**

**v-show指令**

v-show也是条件渲染指令，和v-if指令不同的是，使用v-show指令的元素始终会被渲染到HTML，它只是简单地为元素设置CSS的style属性。

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title></title>

</head>

<body>

<div id="app">

<h1>Hello, Vue.js!</h1>

<h1 v-show="yes">Yes!</h1>

<h1 v-show="no">No!</h1>

<h1 v-show="age >= 25">Age: {{ age }}</h1>

<h1 v-show="name.indexOf('jack') >= 0">Name: {{ name }}</h1>

</div>

</body>

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

<script>

var vm = new Vue({

el: '#app',

data: {

yes: true,

no: false,

age: 28,

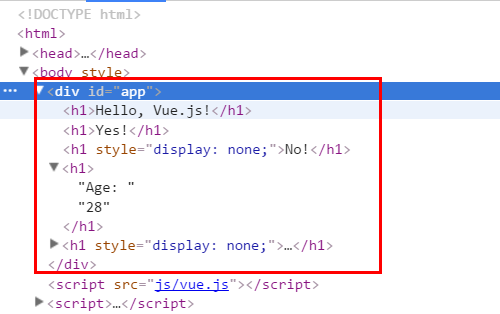
name: 'keepfool'

}

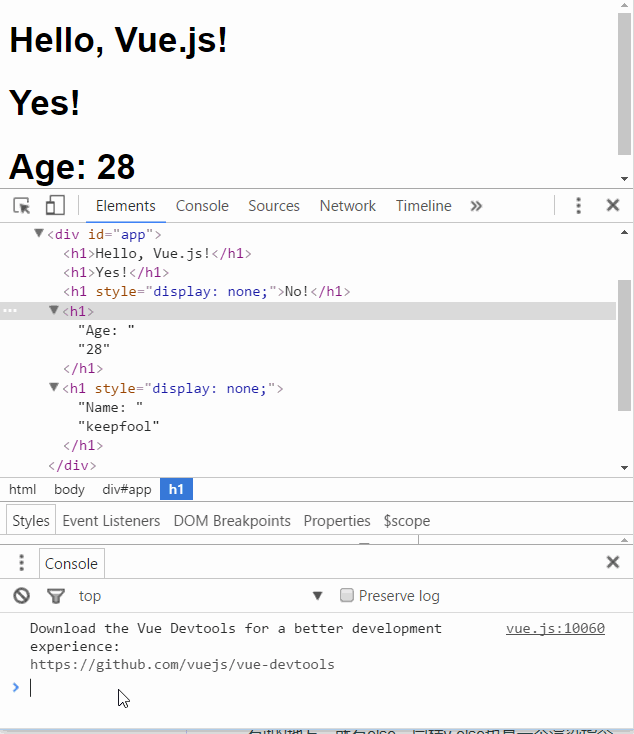
})

</script>

</html>

**[](http://images2015.cnblogs.com/blog/341820/201606/341820-20160627065320577-322007545.png)**

在Chrome控制台更改age属性，使得表达式age >= 25的值为false，可以看到<h1>Age: 24</h1>元素被设置了style="display:none"样式。

[](http://images2015.cnblogs.com/blog/341820/201606/341820-20160627065322656-1570128969.gif)

**v-else指令**

可以用v-else指令为v-if或v-show添加一个“else块”。v-else元素必须立即跟在v-if或v-show元素的后面——否则它不能被识别。

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title></title>

</head>

<body>

<div id="app">

<h1 v-if="age >= 25">Age: {{ age }}</h1>

<h1 v-else>Name: {{ name }}</h1>

<h1>---------------------分割线---------------------</h1>

<h1 v-show="name.indexOf('keep') >= 0">Name: {{ name }}</h1>

<h1 v-else>Sex: {{ sex }}</h1>

</div>

</body>

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

<script>

var vm = new Vue({

el: '#app',

data: {

age: 28,

name: 'keepfool',

sex: 'Male'

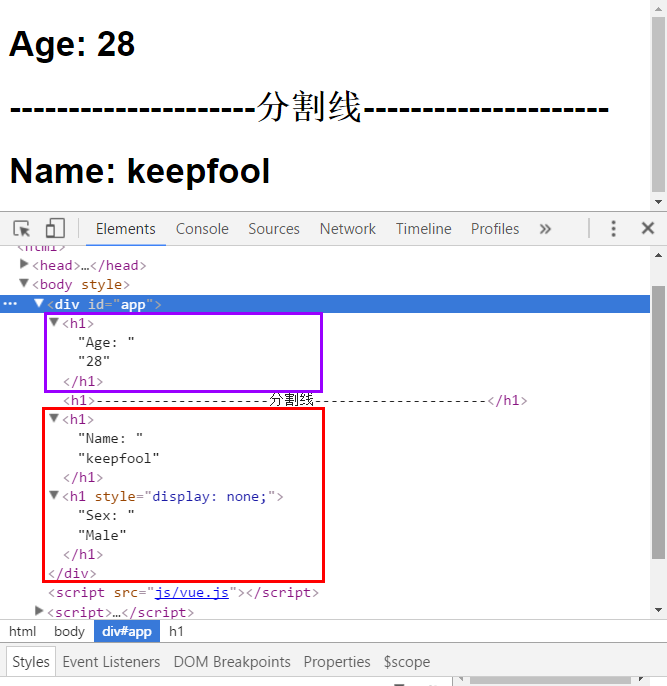
}

})

</script>

</html>

v-else元素是否渲染在HTML中，取决于前面使用的是v-if还是v-show指令。  
这段代码中v-if为true，后面的v-else不会渲染到HTML；v-show为tue，但是后面的v-else仍然渲染到HTML了。

[](http://images2015.cnblogs.com/blog/341820/201606/341820-20160627065325609-935780863.png)

**v-for指令**

v-for指令基于一个数组渲染一个列表，它和JavaScript的遍历语法相似：

v-for="item in items"

items是一个数组，item是当前被遍历的数组元素。

隐藏代码

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title></title>

<link rel="stylesheet" href="styles/demo.css" />

</head>

<body>

<div id="app">

<table>

<thead>

<tr>

<th>Name</th>

<th>Age</th>

<th>Sex</th>

</tr>

</thead>

<tbody>

<tr v-for="person in people">

<td>{{ person.name }}</td>

<td>{{ person.age }}</td>

<td>{{ person.sex }}</td>

</tr>

</tbody>

</table>

</div>

</body>

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

<script>

var vm = new Vue({

el: '#app',

data: {

people: [{

name: 'Jack',

age: 30,

sex: 'Male'

}, {

name: 'Bill',

age: 26,

sex: 'Male'

}, {

name: 'Tracy',

age: 22,

sex: 'Female'

}, {

name: 'Chris',

age: 36,

sex: 'Male'

}]

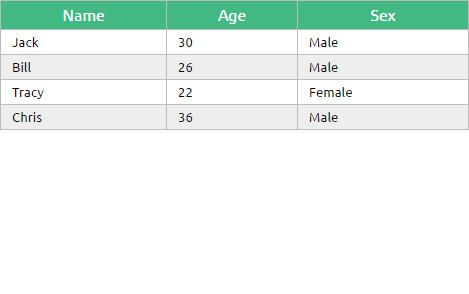
}

})

</script>

</html>

我们在选项对象的data属性中定义了一个people数组，然后在#app元素内使用v-for遍历people数组，输出每个person对象的姓名、年龄和性别。

[](http://images2015.cnblogs.com/blog/341820/201606/341820-20160627065327202-406012811.png)

[View Demo](https://keepfool.github.io/vue-tutorials/01.GettingStarted/v-for.html)

**v-bind指令**

v-bind指令可以在其名称后面带一个参数，中间放一个冒号隔开，这个参数通常是HTML元素的特性（attribute），例如：v-bind:class

v-bind:*argument*="*expression*"

下面这段代码构建了一个简单的分页条，v-bind指令作用于元素的class特性上。  
这个指令包含一个表达式，表达式的含义是：高亮当前页。

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title></title>

<link rel="stylesheet" href="styles/demo.css" />

</head>

<body>

<div id="app">

<ul class="pagination">

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

<a href="javascripit:void(0)" v-bind:class="activeNumber === n + 1 ? 'active' : ''">{{ n + 1 }}</a>

</li>

</ul>

</div>

</body>

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

<script>

var vm = new Vue({

el: '#app',

data: {

activeNumber: 1,

pageCount: 10

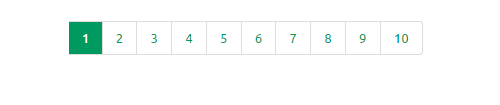
}

})

</script>

</html>

注意v-for="n in pageCount"这行代码，pageCount是一个整数，遍历时n从0开始，然后遍历到pageCount –1结束。

[](http://images2015.cnblogs.com/blog/341820/201606/341820-20160627065328499-156638431.png)

[View Demo](https://keepfool.github.io/vue-tutorials/01.GettingStarted/v-bind.html)

**v-on指令**

v-on指令用于给监听DOM事件，它的用语法和v-bind是类似的，例如监听<a>元素的点击事件：

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

有两种形式调用方法：**绑定一个方法（让事件指向方法的引用），或者使用内联语句。**Greet按钮将它的单击事件直接绑定到greet()方法，而Hi按钮则是调用say()方法。

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title></title>

</head>

<body>

<div id="app">

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

<p>

<!--click事件直接绑定一个方法-->

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

</p>

<p>

<!--click事件使用内联语句-->

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

</p>

</div>

</body>

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

<script>

var vm = new Vue({

el: '#app',

data: {

message: 'Hello, Vue.js!'

},

// 在 `methods` 对象中定义方法

methods: {

greet: function() {

// // 方法内 `this` 指向 vm

alert(this.message)

},

say: function(msg) {

alert(msg)

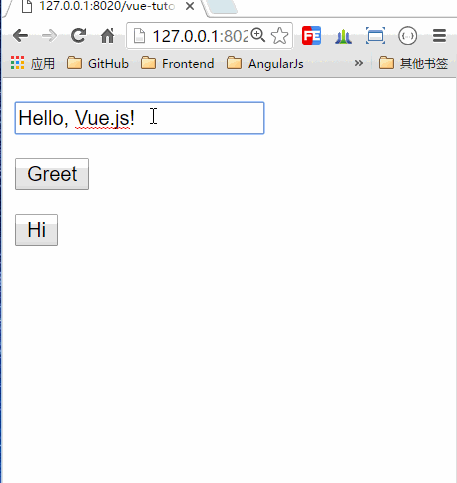
}

}

})

</script>

</html>

[](http://images2015.cnblogs.com/blog/341820/201606/341820-20160627065330109-1825840680.gif)

**v-bind和v-on的缩写**

Vue.js为最常用的两个指令v-bind和v-on提供了缩写方式。**v-bind指令可以缩写为一个冒号，v-on指令可以缩写为@符号。**

<!--完整语法-->

<a href="javascripit:void(0)" v-bind:class="activeNumber === n + 1 ? 'active' : ''">{{ n + 1 }}</a>

<!--缩写语法-->

<a href="javascripit:void(0)" :class="activeNumber=== n + 1 ? 'active' : ''">{{ n + 1 }}</a>

<!--完整语法-->

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

<!--缩写语法-->

<button @click="greet">Greet</button>

**综合示例**

现在我们已经介绍了一些Vue.js的基础知识了，结合以上知识我们可以来做个小Demo。

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title></title>

<link rel="stylesheet" href="styles/demo.css" />

</head>

<body>

<div id="app">

<fieldset>

<legend>

Create New Person

</legend>

<div class="form-group">

<label>Name:</label>

<input type="text" v-model="newPerson.name"/>

</div>

<div class="form-group">

<label>Age:</label>

<input type="text" v-model="newPerson.age"/>

</div>

<div class="form-group">

<label>Sex:</label>

<select v-model="newPerson.sex">

<option value="Male">Male</option>

<option value="Female">Female</option>

</select>

</div>

<div class="form-group">

<label></label>

<button @click="createPerson">Create</button>

</div>

</fieldset>

<table>

<thead>

<tr>

<th>Name</th>

<th>Age</th>

<th>Sex</th>

<th>Delete</th>

</tr>

</thead>

<tbody>

<tr v-for="person in people">

<td>{{ person.name }}</td>

<td>{{ person.age }}</td>

<td>{{ person.sex }}</td>

<td :class="'text-center'"><button @click="deletePerson($index)">Delete</button></td>

</tr>

</tbody>

</table>

</div>

</body>

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

<script>

var vm = new Vue({

el: '#app',

data: {

newPerson: {

name: '',

age: 0,

sex: 'Male'

},

people: [{

name: 'Jack',

age: 30,

sex: 'Male'

}, {

name: 'Bill',

age: 26,

sex: 'Male'

}, {

name: 'Tracy',

age: 22,

sex: 'Female'

}, {

name: 'Chris',

age: 36,

sex: 'Male'

}]

},

methods:{

createPerson: function(){

this.people.push(this.newPerson);

// 添加完newPerson对象后，重置newPerson对象

this.newPerson = {name: '', age: 0, sex: 'Male'}

},

deletePerson: function(index){

// 删一个数组元素

this.people.splice(index,1);

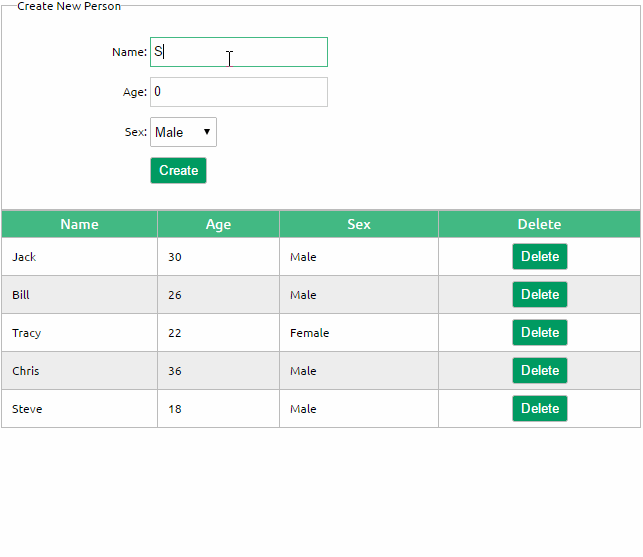
}

}

})

</script>

</html>

[](http://images2015.cnblogs.com/blog/341820/201606/341820-20160627065332359-1941638431.gif)

在我的GitHub Pages查看该Demo：