## 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。

学习网址：<http://www.cnblogs.com/rik28/p/6024425.html>

<http://www.runoob.com/w3cnote/vue-js-quickstart.html>

# 1. 下载vue.js(我选择的是开发版本)

<http://vuejs.org/v2/guide/installation.html#CDN>



# 2.第一个helloWorld

## 1.编辑HTML文件

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>菜鸟教程(runoob.com)</title>

<script src="https://static.runoob.com/assets/vue/1.0.11/vue.min.js"></script>

</head>

<body>

<div id="app">

<!-- view层 -->

{{ message }}

</div>

<!-- JavaScript 代码需要放在尾部（指定的HTML元素之后）

model层 -->

<script>

new Vue({

el:'#app',

data: {

message:'Hello World!'

}

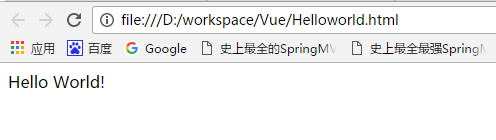
});

</script>

</body>

</html>

## 2.运行



# 3。第二个 v-model这个指令，双向数据绑定

## 1.新建一个HTML文件

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

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

</head>

<body>

<p>修改输入框的值，查看效果：</p>

<div id="app">

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

<input v-model="message">

</div>

<!-- JavaScript 代码需要放在尾部（指定的HTML元素之后） -->

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

</body>

</html>

## 2.导入的model文件vueapp.js的内容

new Vue({

el: '#app',

data: {

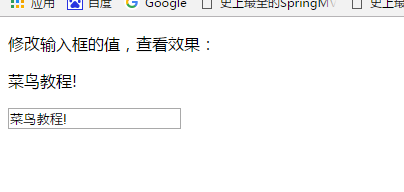
message: '菜鸟教程!'

}

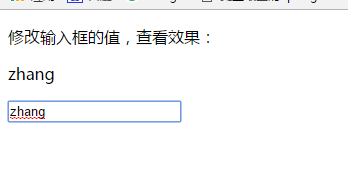
})

## 3.进行测试

### 1.直接打开



### 3.1如果输入框，我们进行自己输入，则上面的值也会发生改变



# 4.第二个 v-model 指令，不进行跟着变化

## 1. {{\* message }} 不会进行变化了

## 2.字符串连接 <!-- 字符串连接 --> message +‘’ 即可

## 3.字符串反转message.split('').reverse().join('')

<meta charset="utf-8">

<title>菜鸟教程(runoob.com)</title>

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

</head>

<body>

<p><b>修改输入框的值，查看效果</b>：</p>

<div id="app">

<p>{{\* message}}</p>

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

<!-- 字符串连接 -->

<p>{{ message + '官网地址：www.runoob.com' }}</p>

<!-- 字符串反转 -->

<p>{{ message.split('').reverse().join('') }}</p>

<input v-model="message">

</div>

<!-- JavaScript 代码需要放在尾部（指定的HTML元素之后） -->

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

</body>

</html>

## 4.直接运行



## 5.修改内容运行



# 5.第三个 v-for

## 1.编辑html文件

### 1.v-for ="todo in todos"表示 todo这个属性的值是在todos里面存放的，在后面的model中会给他进行赋值 通过todeo的text

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

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

</head>

<body>

<div id="app">

<ul>

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

{{ todo.text }}

</li>

</ul>

</div>

<script>

new Vue({

el: '#app',

data: {

todos: [

{ text: '菜鸟教程' },

{ text: 'www.runoob.com' },

{ text: 'www.w3cschool.cc' }

]

}

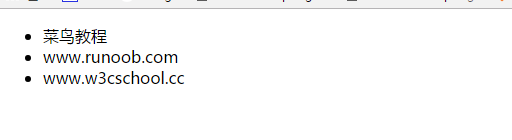
})

</script>

</body>

</html>

## 2.运行测试



## 3.多维数组输出

<div id="app">

<ul id="example-2">

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

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

</li>

</ul>

<script>

new Vue({

el: '#example-2',

data: {

parentMessage: '菜鸟教程官网',

items: [

{ message: 'www.runoob.com' },

{ message: 'www.w3cschool.cc' }

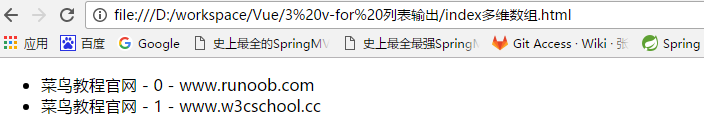
]

}

})

</script>

### 3.1测试结果



# 6、第四个 v-if 条件

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

<h1 v-if="no">No!</h1> 这样的话，在数据源data里就要设置true和flase

<div id="app">

<div v-if="Math.random() > 0.5">

随机数大于 0.5

</div>

<div v-else>

随机数不大于 0.5

</div>

</div>

<script>

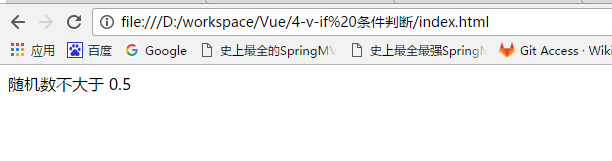
new Vue({

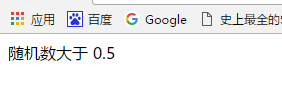
el: '#app'

})

</script>

## 1.测试结果





# 7.过滤器Vue.filter

## 1.{{message | uppercase}}

这样就能输出message的大写了,过滤器也能串联在一起使用:

## 2.{{message | reverse | uppercase}} 讲本身改变之后，再利用里面的值再进行使用过滤器

这里reverse并不是内建的过滤器，我们可以用Vue.filter自定义:

Vue.filter('reverse', function (value) {

return value.split('').reverse().join('')

})

## 3.编写html

<div id="app">

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

<p>{{message | uppercase}}</p>

<p>{{message | reverse | uppercase}}</p>

</div>

<script>

Vue.filter('reverse', function (value) {

return value.split('').reverse().join('')

})

new Vue({

el: '#app',

data: {

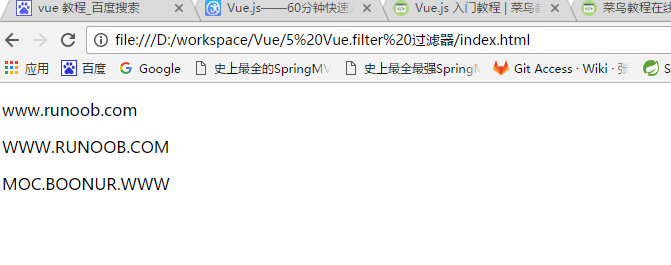
message:'www.runoob.com'

}

})

</script>

## 4.测试

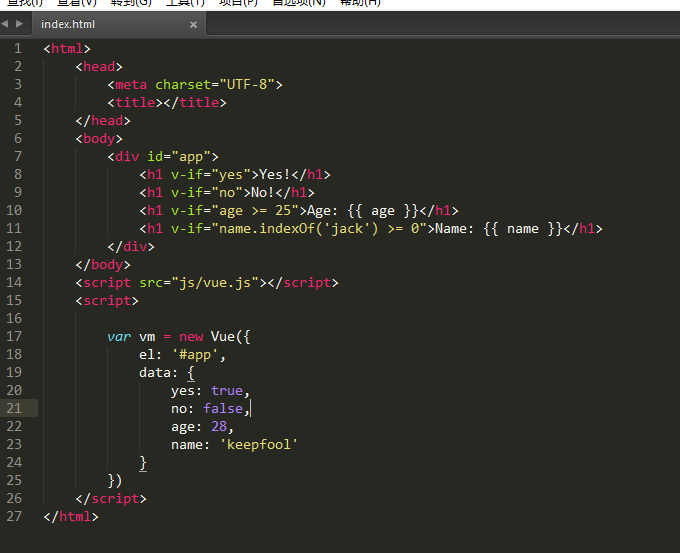


# 二、再次加深学习

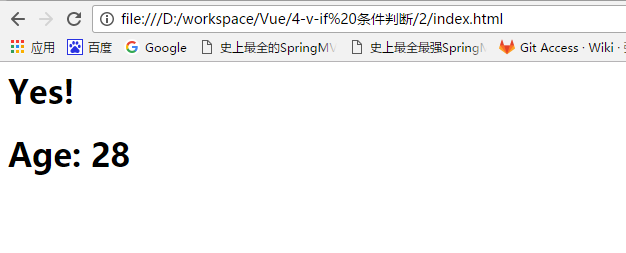
# 1.v-if 里面的值可以作为一个 变量的名字（自己设想的）

data中进行设置里面的值

## 1.源码

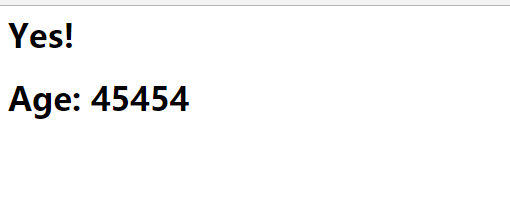


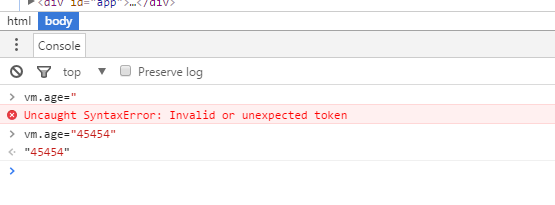
## 2测试



## 3.浏览器控制台进行改变里面的值

vm是 变量，在其内部使用了data添加数据也就可以直接使用age或是那么的其他的属性了





# 2.data中使用var变量的值

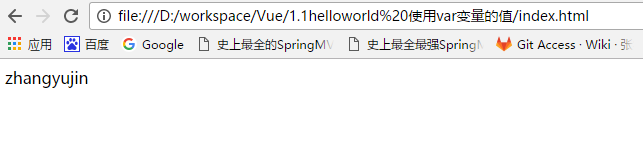
## 1.源码

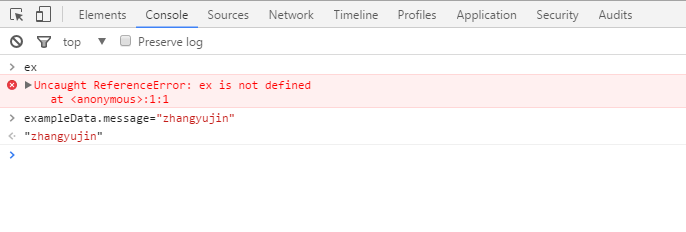


### 1.解释;data中使用变量 exampleData，这个变量其实在上面的var中就已经诶赋值了

## 2.测试使用浏览器控制台改变里面的值

使用变量的名字 exampleData.message="zhangyujin"



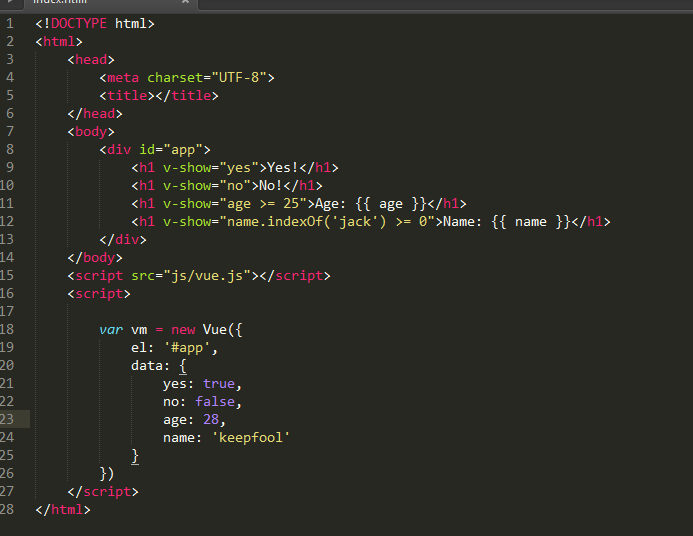


# 3.v-show()和if指令有点像，他们都是条件渲染指令最大不同：

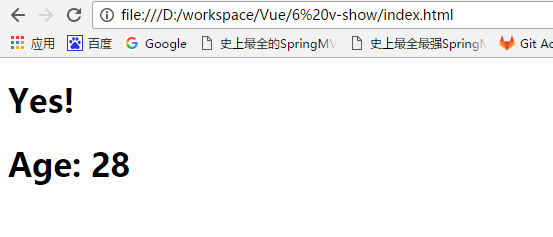
## show 和if 区别;

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。 但是if就是直接清空了。

## 1.源码



## 2.测试



# 4. v-else指令

## 必须搭配使用

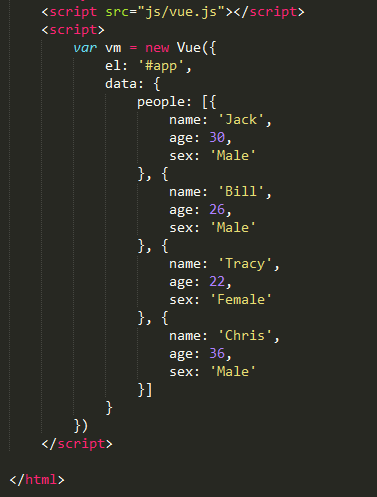
可以用v-else指令为v-if或v-show添加一个“else块”。v-else元素必须立即跟在v-if或v-show元素的后面——否则它不能被识别。

# 5. v-for指令

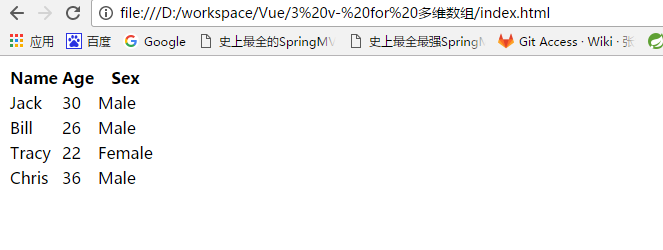
## 1.源码

### 1.1解释 同上文中的解释一样下面的people是一个总体，person是在里面的存放的，然后person中有数据，这里的数据通过model进行设置，也就是赋值操作





## 2.测试



# 6. v-bind指令（不会）

## 1.源码

### 1.1解释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。



## 2.测试



# 7. v-on指令

## 1.源码

### 1.1解释

#### 1.1.1-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：

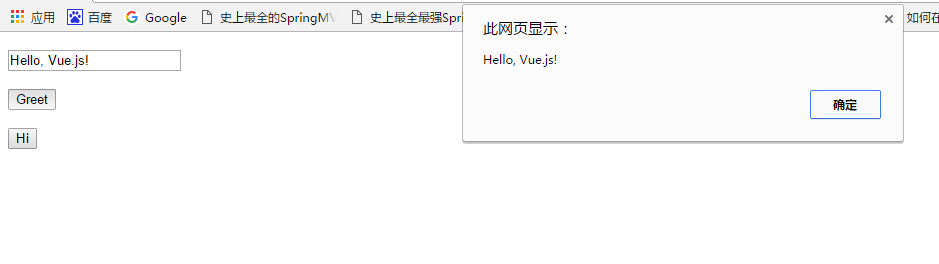
<a v-on:click="doSomething">

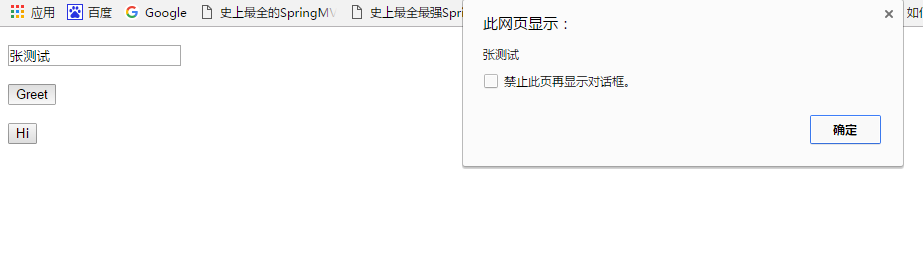
有两种形式调用方法：**绑定一个方法（让事件指向方法的引用），或者使用内联语句。**Greet按钮将它的单击事件直接绑定到greet()方法，而Hi按钮则是调用say()方法。

#### 1.1.2method中书写相关的函数



## 2.测试





# 8v-bind和v-on的缩写

## 1.如何缩写

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-->

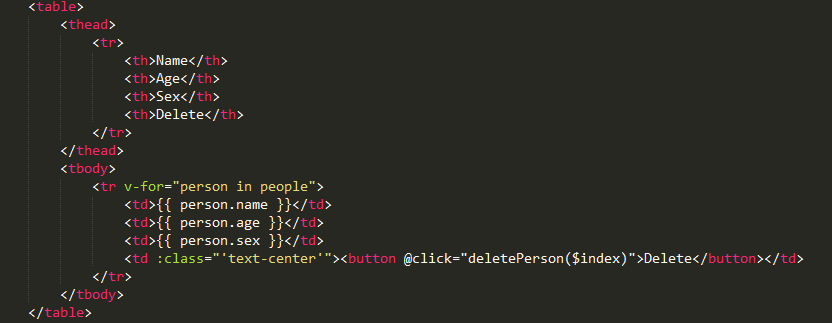
<button v-on:click="greet">Greet</button>

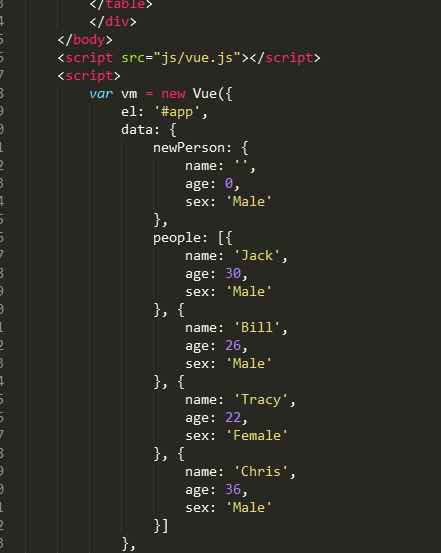
<!--缩写语法-->

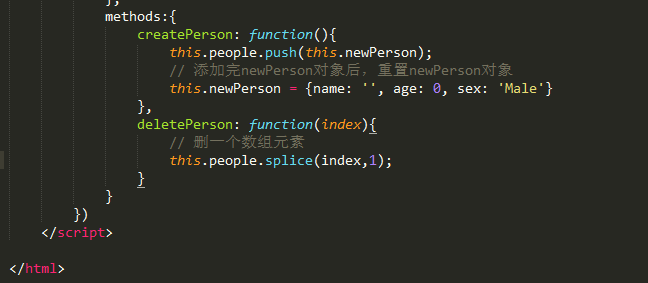
<button @click="greet">Greet</button>

## 2.源码

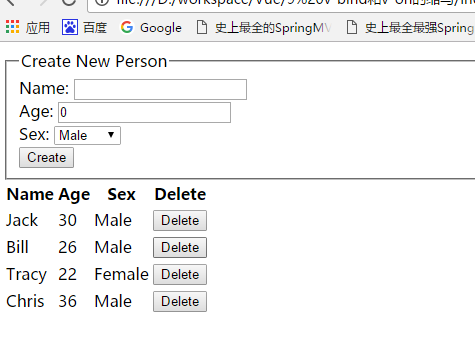








## 3.测试



# 9.总结，命令

v-for 列表显示

v-if v-show 条件循环 v-else

v-bind v-on html控件 和dom点击事件