Vue.js 教程

Vue.js（读音 /vju:/, 类似于 view） 是一套构建用户界面的渐进式框架。

Vue 只关注视图层， 采用自底向上增量开发的设计。

Vue 的目标是通过尽可能简单的 API 实现响应的数据绑定和组合的视图组件。

Vue 学习起来非常简单，本教程基于 Vue 2.1.8 版本测试。

阅读本教程前，您需要了解的知识：

- HTML
- CSS
- JavaScript

本教程主要介绍了 Vue2.x 版本的使用。

第一个实例

Vue 2.0 Hello World

```
<div id="app">
<p>{{ message }}</p>
</div>
```

尝试一下 »

点击 "尝试一下" 按钮查看在线实例

参考资料：

Webpack 入门教程：http://www.runoob.com/w3cnote/webpack-tutorial.html

官方文档：http://vuejs.org/v2/guide/syntax.html

中文文档: https://cn.vuejs.org/v2/guide/syntax.html

Vue.js 安装

1、独立版本

我们可以在 Vue.js 的官网上直接下载 vue.min.js 并用 **<script>** 标签引入。

下载 Vue.js

2、使用 CDN 方法

以下推荐国外比较稳定的两个 CDN，国内还没发现哪一家比较好，目前还是建议下载到本地。

BootCDN（国内）：https://cdn.bootcss.com/vue/2.2.2/vue.min.js

unpkg：https://unpkg.com/vue/dist/vue.js, 会保持和 npm 发布的最新的版本一致。

cdnjs：https://cdnjs.cloudflare.com/ajax/libs/vue/2.1.8/vue.min.js

BootCDN（国内）

```
<div id="app">
<p>{{ message }}</p>
</div>
```

尝试一下 »

unpkg（推荐）

```
<div id="app">
<p>{{ message }}</p>
</div>
```

尝试一下 »

cdnjs

```
<div id="app">
<p>{{ message }}</p>
</div>
```

尝试一下 »

3、NPM 方法

由于 npm 安装速度慢，本教程使用了淘宝的镜像及其命令 cnpm，安装使用介绍参照：使用淘宝 NPM 镜像。

npm 版本需要大于 3.0，如果低于此版本需要升级它：

```
# 查看版本

$ npm -v

2.3.0



#升级 npm
```

```
cnpm install npm -g
```

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
```

命令行工具

Vue.js 提供一个官方命令行工具，可用于快速搭建大型单页应用。

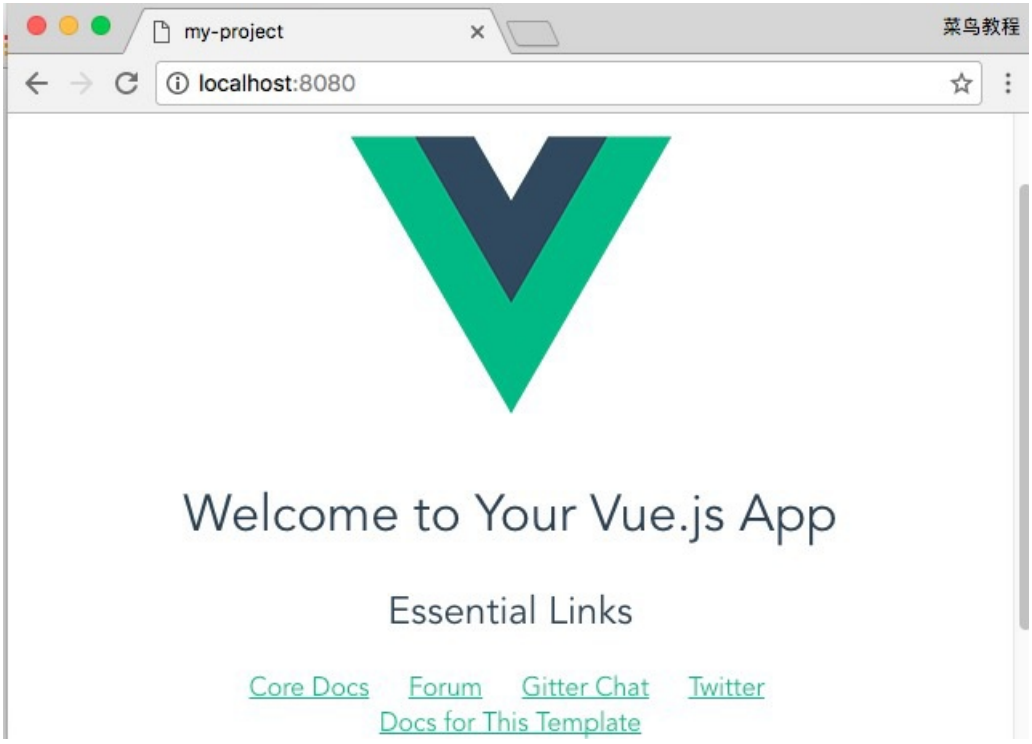
```
# 全局安装 vue-cli

$ cnpm install --global vue-cli

# 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，默认回车即可

This will install Vue 2.x version of the template.



For Vue 1.x use: vue init webpack#1.0 my-project



? Project name my-project

? Project description A Vue.js project

? Author runoob <test@runoob.com>

? Vue build standalone

? Use ESLint to lint your code? Yes

? Pick an ESLint preset Standard

? Setup unit tests with Karma + Mocha? Yes

? Setup e2e tests with Nightwatch? Yes



   vue-cli · Generated "my-project".



   To get started:



     cd my-project

     npm install
```

```
    npm run dev


  Documentation can be found at https://vuejs-templates.github.io/webpack
```

进入项目，安装并运行：

```
$ cd my-project
$ cnpm install
$ cnpm run dev
 DONE  Compiled successfully in 4388ms


> Listening at http://localhost:8080
```

成功执行以上命令后访问 http://localhost:8080/，输出结果如下所示：



> ***注意：****Vue.js 不支持 IE8 及其以下 IE 版本。*

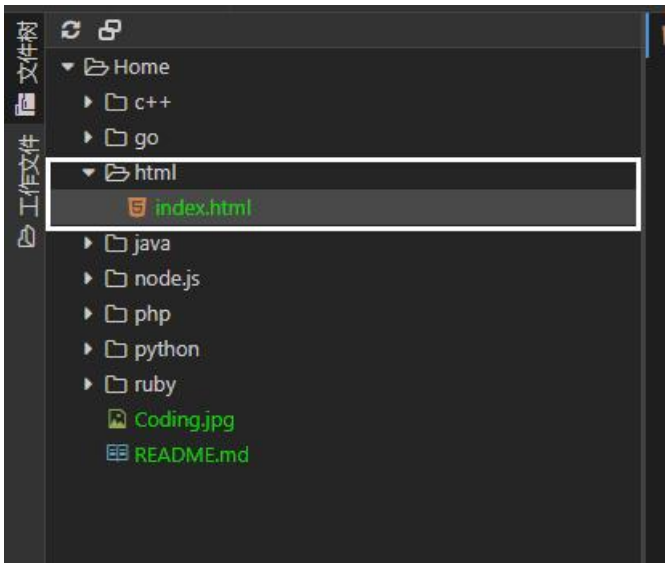
在 Cloud Studio 中运行 Vue.js

下面我们介绍如何在 Cloud Studio 中安装、使用 Vue.js：

step1：访问 Cloud Studio，注册/登录账户。

step2：在右侧的运行环境菜单选择："Node.js"

step3：在左侧代码目录中新建 html 目录，编写你的HTML代码，例如 index.html



step4: 我们推荐链接到一个你可以手动更新的指定版本号：

```
<script src="https://cdn.jsdelivr.net/npm/vue@2.5.17/dist/vue.js"></script>
```

你可以在 cdn.jsdelivr.net/npm/vue 浏览 NPM 包的源代码。 Vue 也可以在 cdnjs 上获取 (cdnjs 的版本更新可能略滞后)。

step5：在用 Vue 构建大型应用时推荐使用 NPM 安装:

```
# 最新稳定版

$ npm install vue
```

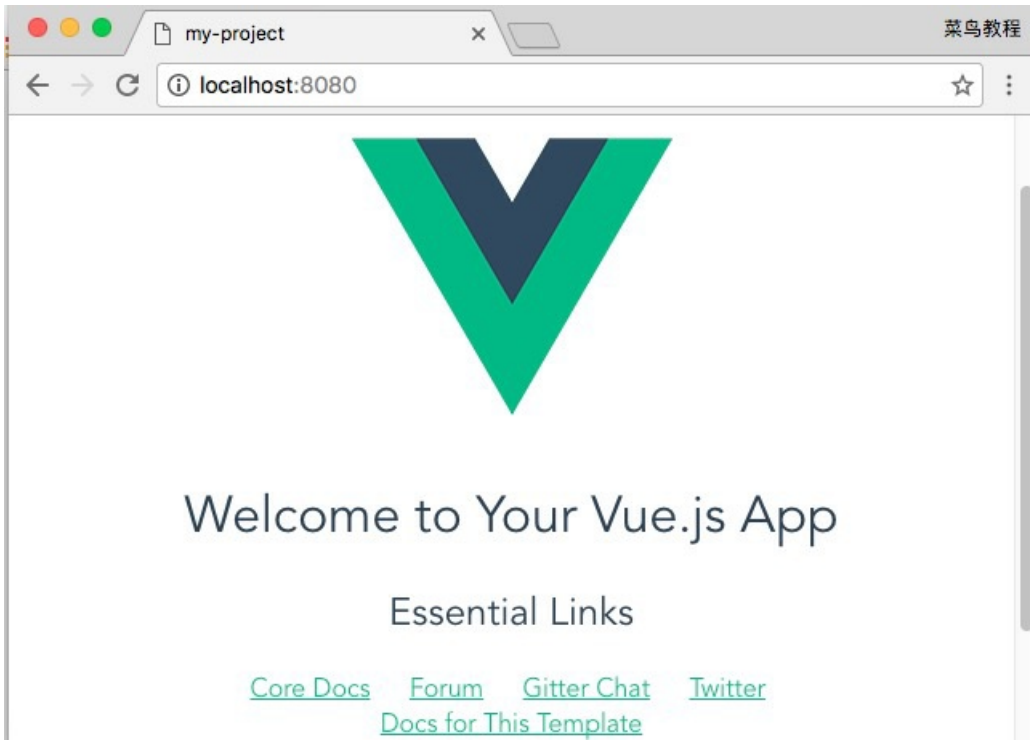
step6：Vue.js 提供一个官方命令行工具，可用于快速搭建大型单页应用。

```
# 全局安装 vue-cli

$ cnpm install --global vue-cli

# 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，默认回车即可
```

进入项目，安装并运行：

```
$ cd my-project

$ cnpm install

$ cnpm run dev

 DONE  Compiled successfully in 4388ms



> Listening at http://localhost:8080
```

step6：点击最右侧的【访问链接】选项卡，在访问链接面板中填写端口号为：8080（和刚才配置文件中的端口号一致），点击创建链接，即可点击生成的链接访问我们刚刚编写的代码，查看 Vue.js 效果。



Vue.js 教程 Vue.js 模板语法

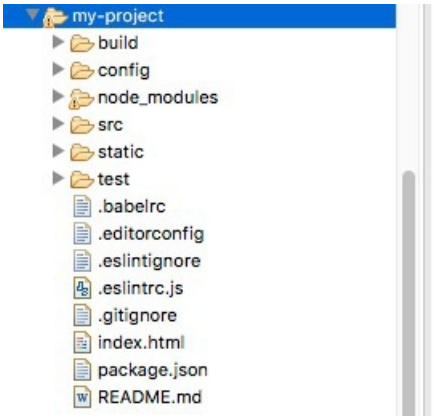
点我分享笔记

反馈/建议

Vue.js 条件语句 Vue.js 计算属性

Vue.js 目录结构

上一章节中我们使用了 npm 安装项目，我们在 IDE（Eclipse、Atom等） 中打开该目录，结构如下所示：



目录解析

目录/文件	说明
build	项目构建(webpack)相关代码
config	配置目录，包括端口号等。我们初学可以使用默认的。
node_modules	npm 加载的项目依赖模块
src	这里是我们要开发的目录，基本上要做的事情都在这个目录里。里面包含了几个目录及文件： assets: 放置一些图片，如logo等。 components: 目录里面放了一个组件文件，可以不用。 App.vue: 项目入口文件，我们也可以直接将组件写这里，而不使用 components 目录。 main.js: 项目的核心文件。
static	静态资源目录，如图片、字体等。
test	初始测试目录，可删除
.xxxx文件	这些是一些配置文件，包括语法配置，git配置等。
index.html	首页入口文件，你可以添加一些 meta 信息或统计代码啥的。
package.json	项目配置文件。
README.md	项目的说明文档，markdown 格式

在前面我们打开 APP.vue 文件，代码如下（解释在注释中）：

src/APP.vue

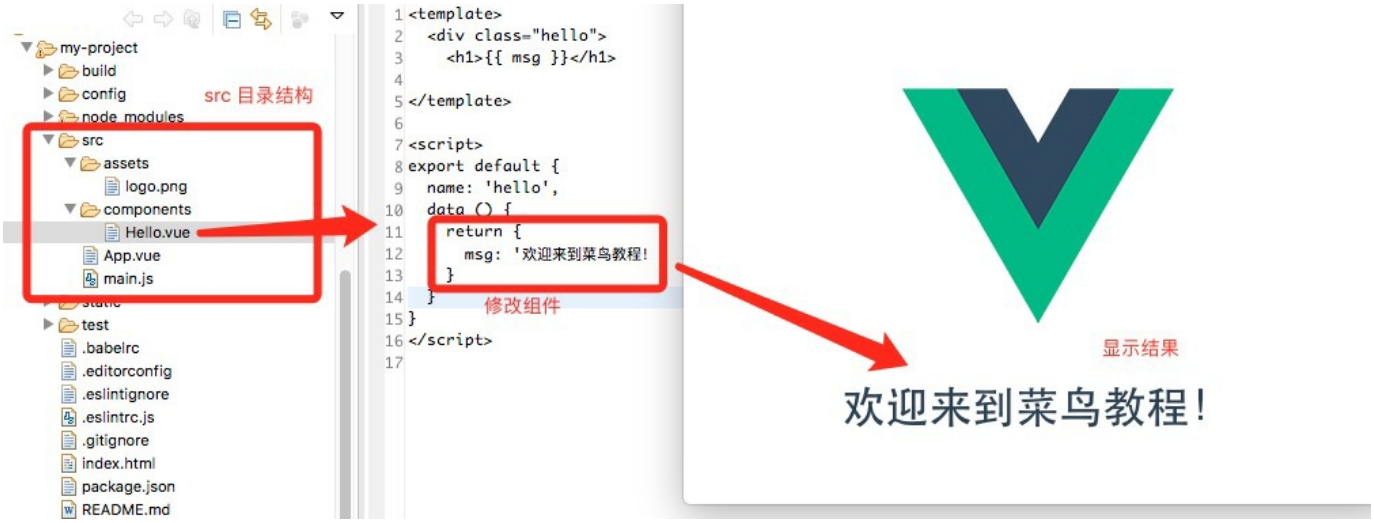
```
<!-- 展示模板 -->
<template>
<div id="app">
<img src="./assets/logo.png">
<hello></hello>
</div>
</template>
<script>
// 导入组件
import Hello from './components/Hello'
export default {
name: 'app',
components: {
Hello
}
}
</script>
<!-- 样式代码 -->
<style>
#app {
font-family: 'Avenir', Helvetica, Arial, sans-serif;
-webkit-font-smoothing: antialiased;
-moz-osx-font-smoothing: grayscale;
text-align: center;
color: #2c3e50;
margin-top: 60px;
}
</style>
```

接下来我们可以尝试修改下初始化的项目，将 Hello.vue 修改为以下代码：

src/components/Hello.vue

```
<template>
<div class="hello">
<h1>{{ msg }}</h1>
</div>
</template>
<script>
export default {
name: 'hello',
data () {
return {
msg: '欢迎来到菜鸟教程！'
}
}
}
</script>
```

重新打开页面 http://localhost:8080/，一般修改后会自动刷新，显示效果如下所示:



Vue.js 条件语句 Vue.js 计算属性

点我分享笔记

反馈/建议

Vue.js 路由 Vue.js 监听属性

Vue.js 起步

每个 Vue 应用都需要通过实例化 Vue 来实现。

语法格式如下：

```
var vm = new Vue({

  // 选项
```

```
})
```

接下来让我们通过实例来看下 Vue 构造器中需要哪些内容：

实例

```
<div id="vue_det">
<h1>site : {{site}}</h1>
<h1>url : {{url}}</h1>
<h1>{{details()}}</h1>
</div>
<script type="text/javascript">
var vm = new Vue({
el: '#vue_det',
data: {
site: "菜鸟教程",
url: "www.runoob.com",
alexa: "10000"
},
methods: {
details: function() {
return this.site + " - 学的不仅是技术，更是梦想！ ";
}
}
})
</script>
```

尝试一下 »

点击 "尝试一下" 按钮查看在线实例

可以看到在 Vue 构造器中有一个el 参数，它是 DOM 元素中的 id。在上面实例中 id 为 vue_det，在 div 元素中：

```
<div id = "vue_det"></div>
```

这意味着我们接下来的改动全部在以上指定的 div 内，div 外部不受影响。

接下来我们看看如何定义数据对象。

data 用于定义属性，实例中有三个属性分别为：site、url、alexa。

`methods` 用于定义的函数，可以通过 return 来返回函数值。

`{{ }}` 用于输出对象属性和函数返回值。

```
<div id="vue_det">

    <h1>site : {{site}}</h1>

    <h1>url : {{url}}</h1>

    <h1>{{details()}}</h1>

</div>
```

当一个 Vue 实例被创建时，它向 Vue 的响应式系统中加入了其 data 对象中能找到的所有的属性。当这些属性的值发生改变时，html 视图将也会产生相应的变化。

实例

```
<div id="vue_det">
<h1>site : {{site}}</h1>
<h1>url : {{url}}</h1>
<h1>Alexa : {{alexa}}</h1>
</div>
<script type="text/javascript">
// 我们的数据对象
var data = { site: "菜鸟教程", url: "www.runoob.com", alexa: 10000}
```

```
var vm = new Vue({
el: '#vue_det',
data: data
})
// 它们引用相同的对象!
document.write(vm.site === data.site) // true
document.write("<br>")
// 设置属性也会影响到原始数据
vm.site = "Runoob"
document.write(data.site + "<br>") // Runoob
// ……反之亦然
data.alexa = 1234
document.write(vm.alexa) // 1234
</script>
```

尝试一下 »

除了数据属性，Vue 实例还提供了一些有用的实例属性与方法。它们都有前缀 $，以便与用户定义的属性区分开来。例如：

实例

```
<div id="vue_det">
<h1>site : {{site}}</h1>
<h1>url : {{url}}</h1>
<h1>Alexa : {{alexa}}</h1>
</div>
<script type="text/javascript">
// 我们的数据对象
var data = { site: "菜鸟教程", url: "www.runoob.com", alexa: 10000}
var vm = new Vue({
el: '#vue_det',
data: data
})
document.write(vm.$data === data) // true
document.write("<br>") // true
document.write(vm.$el === document.getElementById('vue_det')) // true
</script>
```

尝试一下 »

Vue.js 路由　　Vue.js 监听属性

Vue.js 安装　　Vue.js 条件语句

Vue.js 模板语法

Vue.js 使用了基于 HTML 的模版语法，允许开发者声明式地将 DOM 绑定至底层 Vue 实例的数据。

Vue.js 的核心是一个允许你采用简洁的模板语法来声明式的将数据渲染进 DOM 的系统。

结合响应系统，在应用状态改变时， Vue 能够智能地计算出重新渲染组件的最小代价并应用到 DOM 操作上。

插值

文本

数据绑定最常见的形式就是使用 {{...}}（双大括号）的文本插值：

文本插值

```
<div id="app">
<p>{{ message }}</p>
</div>
```

尝试一下 »

Html

使用 v-html 指令用于输出 html 代码：

v-html 指令

```
<div id="app">
<div v-html="message"></div>
</div>
<script>
new Vue({
el: '#app',
data: {
message: '<h1>菜鸟教程</h1>'
}
})
</script>
```

尝试一下 »

属性

HTML 属性中的值应使用 v-bind 指令。

以下实例判断 class1 的值，如果为 true 使用 class1 类的样式，否则不使用该类：

v-bind 指令

```
<div id="app">
<label for="r1">修改颜色</label><input type="checkbox" v-model="class1" id="r1">
<br><br>
<div v-bind:class="{'class1': class1}">
v-bind:class 指令
</div>
</div>
<script>
new Vue({
el: '#app',
data:{
class1: false
}
});
</script>
```

尝试一下 »

表达式

Vue.js 都提供了完全的 JavaScript 表达式支持。

JavaScript 表达式

```
<div id="app">
{{5+5}}<br>
{{ ok ? 'YES' : 'NO' }}<br>
```

```
{{ message.split('').reverse().join('') }}
<div v-bind:id="'list-' + id">菜鸟教程</div>
</div>
<script>
new Vue({
el: '#app',
data: {
ok: true,
message: 'RUNOOB',
id : 1
}
})
</script>
```

尝试一下 »

指令

指令是带有 v- 前缀的特殊属性。

指令用于在表达式的值改变时，将某些行为应用到 DOM 上。如下例子：

实例

```
<div id="app">
<p v-if="seen">现在你看到我了</p>
</div>
<script>
new Vue({
el: '#app',
data: {
seen: true
}
})
</script>
```

尝试一下 »

这里， v-if 指令将根据表达式 seen 的值(true 或 false)来决定是否插入 p 元素。

参数

参数在指令后以冒号指明。例如， v-bind 指令被用来响应地更新 HTML 属性：

实例

```
<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>
```

尝试一下 »

在这里 href 是参数，告知 v-bind 指令将该元素的 href 属性与表达式 url 的值绑定。

另一个例子是 v-on 指令，它用于监听 DOM 事件：

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

在这里参数是监听的事件名。

修饰符

修饰符是以半角句号 `.` 指明的特殊后缀，用于指出一个指令应该以特殊方式绑定。例如，**.prevent** 修饰符告诉 `v-on` 指令对于触发的事件调用 **event.preventDefault()**：

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

用户输入

在 input 输入框中我们可以使用 v-model 指令来实现双向数据绑定：

双向数据绑定

```
<div id="app">
<p>{{ message }}</p>
<input v-model="message">
</div>
<script>
new Vue({
el: '#app',
data: {
message: 'Runoob!'
}
})
</script>
```

尝试一下 »

`v-model` 指令用来在 input、select、text、checkbox、radio 等表单控件元素上创建双向数据绑定，根据表单上的值，自动更新绑定的元素的值。

按钮的事件我们可以使用 v-on 监听事件，并对用户的输入进行响应。

以下实例在用户点击按钮后对字符串进行反转操作：

字符串反转

```
<div id="app">
<p>{{ message }}</p>
<button v-on:click="reverseMessage">反转字符串</button>
</div>
<script>
new Vue({
el: '#app',
data: {
message: 'Runoob!'
},
methods: {
reverseMessage: function () {
this.message = this.message.split('').reverse().join('')
}
}
})
</script>
```

尝试一下 »

过滤器

Vue.js 允许你自定义过滤器，被用作一些常见的文本格式化。由"管道符"指示, 格式如下：

```
<!-- 在两个大括号中 -->
{{ message | capitalize }}

<!-- 在 v-bind 指令中 -->
```

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

过滤器函数接受表达式的值作为第一个参数。

以下实例对输入的字符串第一个字母转为大写：

实例

```
<div id="app">
{{ message | capitalize }}
</div>
<script>
new Vue({
el: '#app',
data: {
message: 'runoob'
},
filters: {
capitalize: function (value) {
if (!value) return ''
value = value.toString()
return value.charAt(0).toUpperCase() + value.slice(1)
}
}
})
</script>
```

尝试一下 »

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

Vue.js 为两个最为常用的指令提供了特别的缩写：

```
<!-- 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.js 安装 Vue.js 条件语句 □

2 篇笔记

#2

□ 写笔记

给元素绑定href时可以也绑一个target，新窗口打开页面。

```
new Vue({
  el: '#app',
  data: {
    url: 'http://www.runoob.com',
    target:'_blank'
  }
})
```

尝试一下 »

苏安年1年前 (2017-09-27)

#1

当我们给一个比如 props 中，或者 data 中被观测的对象添加一个新的属性的时候，不能直接添加，必须使用 Vue.set 方法。
Vue.set 方法用来新增对象的属性。如果要增加属性的对象是响应式的，那该方法可以确保属性被创建后也是响应式的，同时触发视图更新

```
export default {
  props: {
    food:{
      type:Object
    }
  },
  methods: {
    addCart() {
      if(!this.food.count){
        Vue.set(this.food,'count',1)
      }else{
        this.food.count++;
      }
    }
  }
```


这里本来 food 对象是没有 count 属性的，我们要给其添加 count 属性就必须使用 Vue.set 方法，而不能写成 `this.food.count = 1`

锋12个月前 (10-19)

反馈/建议

Vue.js 条件与循环

条件判断

v-if

条件判断使用 v-if 指令：

v-if 指令

在元素 和 template 中使用 v-if 指令：

```
<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>
```

尝试一下 »

这里， v-if 指令将根据表达式 seen 的值(true 或 false)来决定是否插入 p 元素。

在字符串模板中，如 Handlebars ，我们得像这样写一个条件块：

```
<!-- Handlebars 模板 -->

{{#if ok}}

  <h1>Yes</h1>

{{/if}}
```

v-else

可以用 v-else 指令给 v-if 添加一个 "else" 块：

v-else 指令

随机生成一个数字，判断是否大于0.5，然后输出对应信息：

```
<div id="app">
<div v-if="Math.random() > 0.5">
Sorry
</div>
<div v-else>
Not sorry
</div>
</div>
<script>
new Vue({
el: '#app'
})
</script>
```

尝试一下 »

v-else-if

v-else-if 在 2.1.0 新增，顾名思义，用作 v-if 的 else-if 块。可以链式的多次使用：

v-else 指令

判断 type 变量的值：

```
<div id="app">
<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>
</div>
<script>
```

```
new Vue({
el: '#app',
data: {
type: 'C'
}
})
</script>
```

尝试一下 »

v-else 、v-else-if 必须跟在 v-if 或者 v-else-if之后。

v-show

我们也可以使用 v-show 指令来根据条件展示元素：

v-show 指令

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

尝试一下 »

Vue.js 模板语法 Vue.js 目录结构

Vue.js 样式绑定 Vue.js 事件处理器

循环语句

循环使用 v-for 指令。

v-for 指令需要以 **site in sites** 形式的特殊语法， sites 是源数据数组并且 site 是数组元素迭代的别名。

v-for 可以绑定数据到数组来渲染一个列表：

v-for 指令

```
<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>
```

尝试一下 »

模板中使用 v-for：

v-for

```
<ul>
<template v-for="site in sites">
<li>{{ site.name }}</li>
<li>--------------</li>
</template>
</ul>
```

尝试一下 »

v-for 迭代对象

v-for 可以通过一个对象的属性来迭代数据：

v-for

```
<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>
```

尝试一下 »

你也可以提供第二个的参数为键名：

v-for

```
<div id="app">
<ul>
<li v-for="(value, key) in object">
{{ key }} : {{ value }}
</li>
</ul>
</div>
```

尝试一下 »

第三个参数为索引：

v-for

```
<div id="app">
<ul>
```

```
<li v-for="(value, key, index) in object">
{{ index }}. {{ key }} : {{ value }}
</li>
</ul>
</div>
```

尝试一下 »

v-for 迭代整数

v-for 也可以循环整数

v-for

```
<div id="app">
<ul>
<li v-for="n in 10">
{{ n }}
</li>
</ul>
</div>
```

尝试一下 »

Vue.js 样式绑定 Vue.js 事件处理器

5 篇笔记

写笔记

反馈/建议

Vue.js 目录结构 Vue.js 样式绑定

Vue.js 计算属性

计算属性关键词: computed。

计算属性在处理一些复杂逻辑时是很有用的。

可以看下以下反转字符串的例子：

实例 1

```
<div id="app">
{{ message.split('').reverse().join('') }}
</div>
```

尝试一下 »

实例 1 中模板变的很复杂起来，也不容易看懂理解。

接下来我们看看使用了计算属性的实例：

实例 2

```
<div id="app">
<p>原始字符串: {{ message }}</p>
<p>计算后反转字符串: {{ reversedMessage }}</p>
</div>
<script>
var vm = new Vue({
el: '#app',
data: {
message: 'Runoob!'
},
computed: {
// 计算属性的 getter
reversedMessage: function () {
// `this` 指向 vm 实例
return this.message.split('').reverse().join('')
}
}
})
</script>
```

尝试一下 »

实例 2 中声明了一个计算属性 reversedMessage 。

提供的函数将用作属性 vm.reversedMessage 的 getter 。

vm.reversedMessage 依赖于 vm.message，在 vm.message 发生改变时，vm.reversedMessage 也会更新。

computed vs methods

我们可以使用 methods 来替代 computed，效果上两个都是一样的，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。

实例 3

```
methods: {
reversedMessage2: function () {
return this.message.split('').reverse().join('')
}
}
```

尝试一下 »

可以说使用 computed 性能会更好，但是如果你不希望缓存，你可以使用 methods 属性。

computed setter

computed 属性默认只有 getter ，不过在需要时你也可以提供一个 setter ：

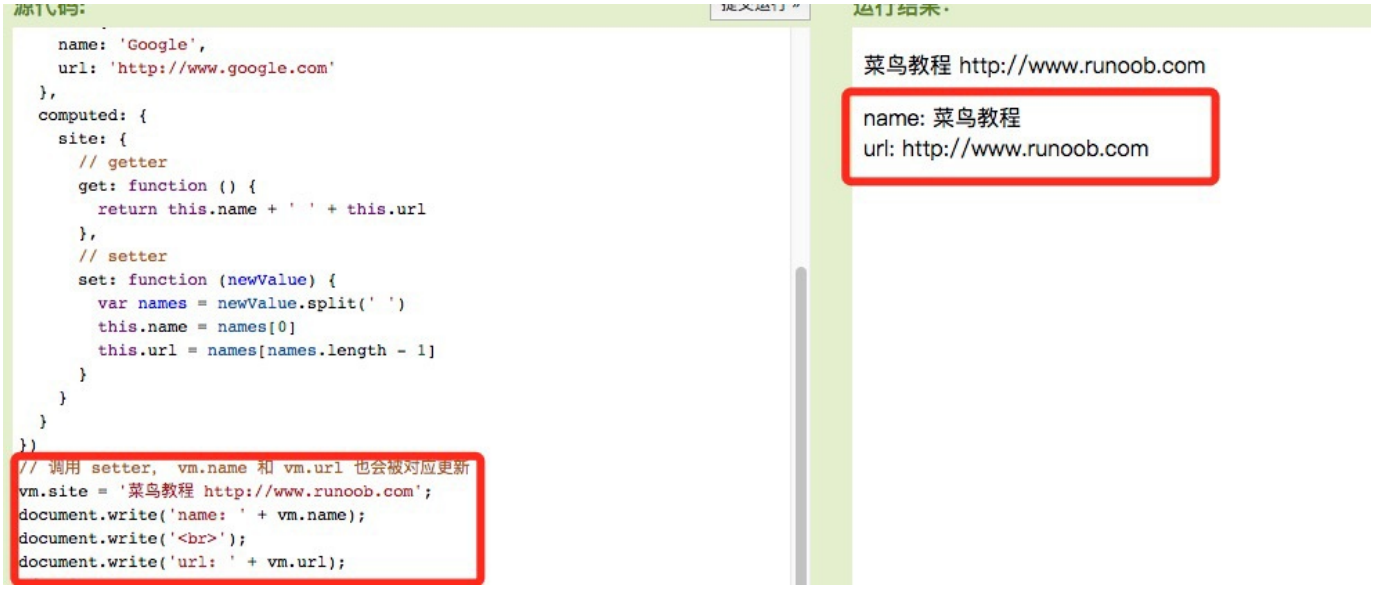
实例 4

```
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) {
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);
```

尝试一下 »

从实例运行结果看在运行 vm.site = '菜鸟教程 http://www.runoob.com'; 时，setter 会被调用， vm.name 和 vm.url 也会被对应更新。



Vue.js 目录结构 Vue.js 样式绑定

点我分享笔记

反馈/建议

Vue.js 起步 Vue.js 过渡 & 动画

Vue.js 监听属性

本章节，我们将为大家介绍 Vue.js 监听属性 watch，我们可以通过 watch 来响应数据的变化。

以下实例通过使用 watch 实现计数器：

实例

```
<div id = "app">
<p style = "font-size:25px;">计数器: {{ counter }}</p>
<button @click = "counter++" style = "font-size:25px;">点我</button>
</div>
<script type = "text/javascript">
var vm = new Vue({
el: '#app',
```

```
data: {
counter: 1
}
});
vm.$watch('counter', function(nval, oval) {
alert('计数器值的变化 :' + oval + ' 变为 ' + nval + '!');
});
</script>
```

尝试一下 »

以下实例进行千米与米之间的换算：

实例

```
<div id = "computed_props">
千米 : <input type = "text" v-model = "kilometers">
米 : <input type = "text" v-model = "meters">
</div>
<p id="info"></p>
<script type = "text/javascript">
var vm = new Vue({
el: '#computed_props',
data: {
kilometers : 0,
meters:0
},
methods: {
},
computed :{
},
watch : {
kilometers:function(val) {
this.kilometers = val;
this.meters = val * 1000;
},
meters : function (val) {
this.kilometers = val/ 1000;
this.meters = val;
}
}
});
// $watch 是一个实例方法
vm.$watch('kilometers', function (newValue, oldValue) {
// 这个回调将在 vm.kilometers 改变后调用
document.getElementById ("info").innerHTML = "修改前值为: " + oldValue + "，修改后值为: " + newValue;
})
</script>
```

尝试一下 »

点击 "尝试一下" 按钮查看在线实例

以上代码中我们创建了两个输入框，data 属性中， kilometers 和 meters 初始值都为 0。watch 对象创建了两个方法 kilometers 和 meters。

当我们再输入框输入数据时，watch 会实时监听数据变化并改变自身的值。可以看下如下视频演示：

您的浏览器不支持 video 标签。

Vue.js 起步 Vue.js 过渡 & 动画

点我分享笔记

反馈/建议

Vue.js 样式绑定

Vue.js class

class 与 style 是 HTML 元素的属性，用于设置元素的样式，我们可以用 v-bind 来设置样式属性。

Vue.js v-bind 在处理 class 和 style 时， 专门增强了它。表达式的结果类型除了字符串之外，还可以是对象或数组。

---

class 属性绑定

我们可以为 v-bind:class 设置一个对象，从而动态的切换 class:

实例 1

实例中将 isActive 设置为 true 显示了一个绿色的 div 块，如果设置为 false 则不显示：

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

尝试一下 »

以上实例 div class 为：

```
<div class="active"></div>
```

我们也可以在对象中传入更多属性用来动态切换多个 class 。

实例 2

text-danger 类背景颜色覆盖了 active 类的背景色：

```
<div class="static"
v-bind:class="{ active: isActive, 'text-danger': hasError }">
</div>
```

尝试一下 »

以上实例 div class 为：

```
<div class="static active text-danger"></div>
```

我们也可以直接绑定数据里的一个对象：

实例 3

text-danger 类背景颜色覆盖了 active 类的背景色：

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

尝试一下 »

实例 3 与 实例 2 的渲染结果是一样的。

此外，我们也可以在这里绑定返回对象的计算属性。这是一个常用且强大的模式：

实例 4

```
new Vue({
el: '#app',
```

```
data: {
isActive: true,
error: null
},
computed: {
classObject: function () {
return {
active: this.isActive && !this.error,
'text-danger': this.error && this.error.type === 'fatal',
}
}
}
})
```

尝试一下 »

数组语法

我们可以把一个数组传给 **v-bind:class**，实例如下：

实例 5

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

尝试一下 »

以上实例 div class 为：

```
<div class="active text-danger"></div>
```

我们还可以使用三元表达式来切换列表中的 class：

实例 6

errorClass 是始终存在的，isActive 为 true 时添加 activeClass 类：

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

尝试一下 »

Vue.js style(内联样式)

我们可以在 **v-bind:style** 直接设置样式：

实例 7

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

尝试一下 »

以上实例 div style 为：

```
<div style="color: green; font-size: 30px;">菜鸟教程</div>
```

也可以直接绑定到一个样式对象，让模板更清晰：

实例 8

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

尝试一下 »

v-bind:style 可以使用数组将多个样式对象应用到一个元素上：

实例 9

```
<div id="app">
<div v-bind:style="[baseStyles, overridingStyles]">菜鸟教程</div>
</div>
```

尝试一下 »

*注意：当 **v-bind:style** 使用需要特定前缀的 CSS 属性时，如 transform ， Vue.js 会自动侦测并添加相应的前缀。*

Vue.js 计算属性 Vue.js 循环语句

1 篇笔记
#1

写笔记

Mustache (双大括号写法)不能在 HTML 属性中使用，应使用 v-bind 指令：

```
<div v-bind:id="dynamicId"></div>
```

这对布尔值的属性也有效 —— 如果条件被求值为 false 的话该属性会被移除：

```
<button v-bind:disabled="someDynamicCondition">Button</button>
```

1:v-bind动态绑定指令，默认情况下标签自带属性的值是固定的，在为了能够动态的给这些属性添加值，可以使用v-bind:你要动态变化的值="表达式"
2:v-bind用于绑定属性和数据 ， 其缩写为“ : ” 也就是v-bind:id === :id
3:v-model用在表单控件上的，用于实现双向数据绑定，所以如果你用在除了表单控件以外的标签是没有任何效果的。

锋**12**个月前 **(10-19)**

反馈/建议

Vue.js 循环语句 Vue.js 表单

Vue.js 事件处理器

事件监听可以使用 v-on 指令：

v-on

```
<div id="app">
<button v-on:click="counter += 1">增加 1</button>
<p>这个按钮被点击了 {{ counter }} 次。</p>
</div>
<script>
new Vue({
el: '#app',
data: {
counter: 0
}
```

```
})
</script>
```

尝试一下 »

通常情况下，我们需要使用一个方法来调用 JavaScript 方法。

v-on 可以接收一个定义的方法来调用。

v-on

```
<div id="app">
<!-- `greet` 是在下面定义的方法名 -->
<button v-on:click="greet">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 (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
app.greet() // -> 'Hello Vue.js!'
</script>
```

尝试一下 »

除了直接绑定到一个方法，也可以用内联 JavaScript 语句：

v-on

```
<div id="app">
<button v-on:click="say('hi')">Say hi</button>
<button v-on:click="say('what')">Say what</button>
</div>
<script>
new Vue({
el: '#app',
methods: {
say: function (message) {
alert(message)
}
}
})
</script>
```

尝试一下 »

事件修饰符

Vue.js 为 v-on 提供了事件修饰符来处理 DOM 事件细节，如：event.preventDefault() 或 event.stopPropagation()。

Vue.js通过由点(.)表示的指令后缀来调用修饰符。

```
.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（而不是子元素）触发时触发回调 -->

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



<!-- click 事件只能点击一次，2.1.4版本新增 -->

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

按键修饰符

Vue 允许为 v-on 在监听键盘事件时添加按键修饰符：

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

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

记住所有的 keyCode 比较困难，所以 Vue 为最常用的按键提供了别名：

```
<!-- 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

.ctrl

.alt

.shift

.meta

实例

```
<p><!-- Alt + C -->

<input @keyup.alt.67="clear">

<!-- Ctrl + Click -->

<div @click.ctrl="doSomething">Do something</div>
```

Vue.js 循环语句 Vue.js 表单

Vue.js 事件处理器 Vue.js 组件

Vue.js 表单

这节我们为大家介绍 Vue.js 表单上的应用。

你可以用 v-model 指令在表单控件元素上创建双向数据绑定。

v-model 会根据控件类型自动选取正确的方法来更新元素。

输入框

实例中演示了 input 和 textarea 元素中使用 v-model 实现双向数据绑定：

```
<div id="app">
<p>input 元素：</p>
<input v-model="message" placeholder="编辑我……">
<p>消息是：{{ message }}</p>
<p>textarea 元素：</p>
<p style="white-space: pre">{{ message2 }}</p>
<textarea v-model="message2" placeholder="多行文本输入……"></textarea>
</div>
<script>
new Vue({
el: '#app',
data: {
message: 'Runoob',
message2: '菜鸟教程\r\nhttp://www.runoob.com'
}
})
</script>
```

尝试一下 »

复选框

复选框如果是一个为逻辑值，如果是多个则绑定到同一个数组：

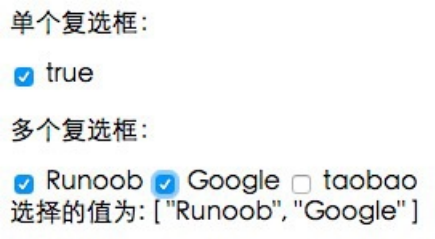
复选框

以下实例中演示了复选框的双向数据绑定：

```
<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>
```

尝试一下 »

实例中勾选复选框效果如下所示：



单选按钮

以下实例中演示了单选按钮的双向数据绑定：

单选按钮

```
<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>
```

尝试一下 »

选中后，效果如下图所示：

select 列表

以下实例中演示了下拉列表的双向数据绑定：

select

```
<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>
```

尝试一下 »

选取 Runoob，输出效果如下所示：

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，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">
```

□ Vue.js 事件处理器　　Vue.js 组件 □


```
new Vue({

    el: '#app',

    data: {

        checked: false,

        checkedNames: [],

        checkedArr: ["Runoob", "Taobao", "Google"]

    },

    methods: {

        changeAllChecked: function() {

            if (this.checked) {

                this.checkedNames = this.checkedArr

            } else {

                this.checkedNames = []

            }

        }

    },

    watch: {

        "checkedNames": function() {

            if (this.checkedNames.length == this.checkedArr.length) {

                this.checked = true
```

```
            } else {
                this.checked = false
            }
        }
    }
})
```

尝试一下 »

修行1年前 (2017-09-28)

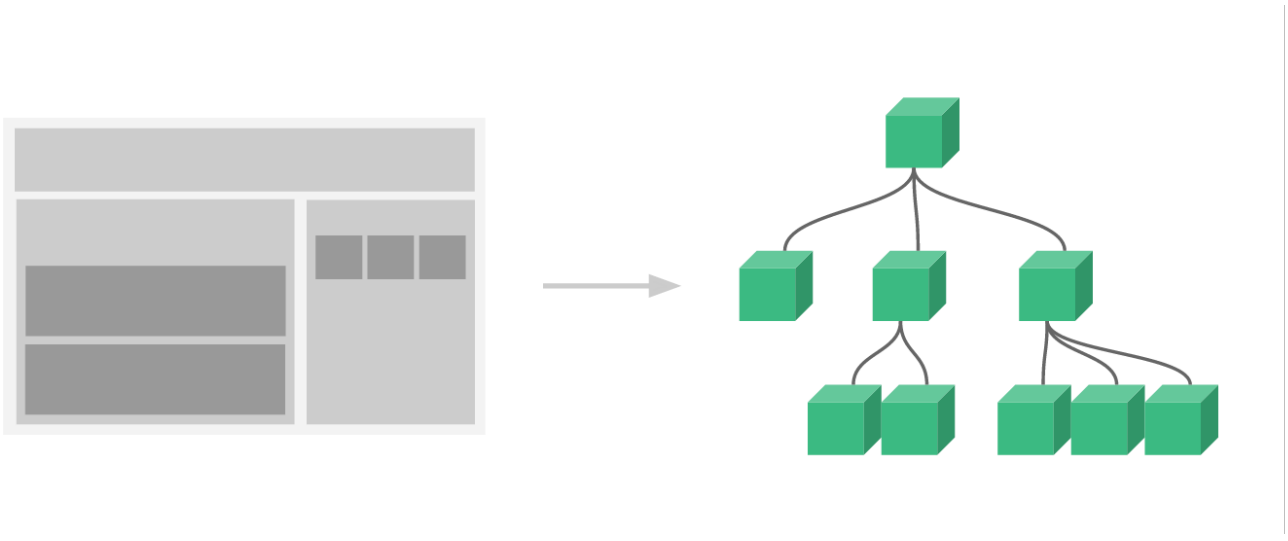
反馈/建议

Vue.js 组件

组件（Component）是 Vue.js 最强大的功能之一。

组件可以扩展 HTML 元素，封装可重用的代码。

组件系统让我们可以用独立可复用的小组件来构建大型应用，几乎任意类型的应用的界面都可以抽象为一个组件树：



注册一个全局组件语法格式如下：

```
Vue.component(tagName, options)
```

tagName 为组件名，options 为配置选项。注册后，我们可以使用以下方式来调用组件：

```
<tagName></tagName>
```

全局组件

所有实例都能用全局组件。

全局组件实例

注册一个简单的全局组件 runoob，并使用它：

```
<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>
```

尝试一下 »

局部组件

我们也可以在实例选项中注册局部组件，这样组件只能在这个实例中使用：

局部组件实例

注册一个简单的局部组件 runoob，并使用它：

```
<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>
```

尝试一下 »

Prop

prop 是父组件用来传递数据的一个自定义属性。

父组件的数据需要通过 props 把数据传给子组件，子组件需要显式地用 props 选项声明 "prop"：

Prop 实例

```
<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>
```

尝试一下 »

动态 Prop

类似于用 v-bind 绑定 HTML 特性到一个表达式，也可以用 v-bind 动态绑定 props 的值到父组件的数据中。每当父组件的数据变化时，该变化也会传导给子组件：

Prop 实例

```
<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>
```

尝试一下 »

以下实例中将 v-bind 指令将 todo 传到每一个重复的组件中：

Prop 实例

```
<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>
```

尝试一下 »

注意: prop 是单向绑定的：当父组件的属性变化时，将传导给子组件，但是不会反过来。

Prop 验证

组件可以为 props 指定验证要求。

prop 是一个对象而不是字符串数组时，它包含验证要求：

```
Vue.component('example', {
  props: {
    // 基础类型检测 （`null` 意思是任何类型都可以）
    propA: Number,
    // 多种类型
    propB: [String, Number],
    // 必传且是字符串
    propC: {
      type: String,
      required: true
    },
    // 数字，有默认值
    propD: {
      type: Number,
      default: 100
    },
    // 数组 / 对象的默认值应当由一个工厂函数返回
    propE: {
      type: Object,
      default: function () {
        return { message: 'hello' }
      }
    },
    // 自定义验证函数
    propF: {
      validator: function (value) {
        return value > 10
      }
    }
```

```
  }
})
```

type 可以是下面原生构造器：

- String
- Number
- Boolean
- Function
- Object
- Array

type 也可以是一个自定义构造器，使用 instanceof 检测。

自定义事件

父组件是使用 props 传递数据给子组件，但如果子组件要把数据传递回去，就需要使用自定义事件！

我们可以使用 v-on 绑定自定义事件, 每个 Vue 实例都实现了事件接口(Events interface)，即：

- 使用 `$on(eventName)` 监听事件
- 使用 `$emit(eventName)` 触发事件

另外，父组件可以在使用子组件的地方直接用 v-on 来监听子组件触发的事件。

以下实例中子组件已经和它外部完全解耦了。它所做的只是触发一个父组件关心的内部事件。

实例

```
<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="incrementHandler">{{ counter }}</button>',
data: function () {
return {
counter: 0
}
},
methods: {
incrementHandler: function () {
this.counter += 1
this.$emit('increment')
}
},
})
new Vue({
el: '#counter-event-example',
data: {
total: 0
},
methods: {
incrementTotal: function () {
this.total += 1
}
}
})
</script>
```

尝试一下 »

如果你想在某个组件的根元素上监听一个原生事件。可以使用 .native 修饰 v-on 。例如：

```
<my-component v-on:click.native="doTheThing"></my-component>
```

Vue.js 表单 Vue.js 自定义指令

Vue.js 组件 Vue.js 路由

Vue.js 自定义指令

除了默认设置的核心指令(v-model 和 v-show), Vue 也允许注册自定义指令。

下面我们注册一个全局指令 v-focus, 该指令的功能是在页面加载时，元素获得焦点：

实例

```
<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>
```

尝试一下 »

我们也可以在实例使用 directives 选项来注册局部指令，这样指令只能在这个实例中使用：

实例

```
<div id="app">
<p>页面载入时，input 元素自动获取焦点：</p>
<input v-focus>
```

```
</div>
<script>
// 创建根实例
new Vue({
el: '#app',
directives: {
// 注册一个局部的自定义指令 v-focus
focus: {
// 指令的定义
inserted: function (el) {
// 聚焦元素
el.focus()
}
}
}
})
</script>
```

尝试一下 »

钩子

钩子函数

指令定义函数提供了几个钩子函数（可选）：

`bind`: 只调用一次，指令第一次绑定到元素时调用，用这个钩子函数可以定义一个在绑定时执行一次的初始化动作。

`inserted`: 被绑定元素插入父节点时调用（父节点存在即可调用，不必存在于 document 中）。

`update`: 被绑定元素所在的模板更新时调用，而不论绑定值是否变化。通过比较更新前后的绑定值，可以忽略不必要的模板更新（详细的钩子函数参数见下）。

`componentUpdated`: 被绑定元素所在模板完成一次更新周期时调用。

`unbind`: 只调用一次， 指令与元素解绑时调用。

钩子函数参数

钩子函数的参数有：

el: 指令所绑定的元素，可以用来直接操作 DOM 。

binding: 一个对象，包含以下属性：

name: 指令名，不包括 `v-` 前缀。

value: 指令的绑定值， 例如： `v-my-directive="1 + 1"`, value 的值是 `2`。

oldValue: 指令绑定的前一个值，仅在 `update` 和 `componentUpdated` 钩子中可用。无论值是否改变都可用。

expression: 绑定值的表达式或变量名。 例如 `v-my-directive="1 + 1"` ， expression 的值是 `"1 + 1"`。

arg: 传给指令的参数。例如 `v-my-directive:foo`， arg 的值是 `"foo"`。

modifiers: 一个包含修饰符的对象。 例如： `v-my-directive.foo.bar`, 修饰符对象 modifiers 的值是 `{ foo: true, bar: true }`。

vnode: Vue 编译生成的虚拟节点。

oldVnode: 上一个虚拟节点，仅在 `update` 和 `componentUpdated` 钩子中可用。

以下实例演示了这些参数的使用：

实例

```
<div id="app" v-runoob:hello.a.b="message">
</div>
<script>
Vue.directive('runoob', {
bind: function (el, binding, vnode) {
```

```
var s = JSON.stringify
el.innerHTML =
'name: ' + s(binding.name) + '<br>' +
'value: ' + s(binding.value) + '<br>' +
'expression: ' + s(binding.expression) + '<br>' +
'argument: ' + s(binding.arg) + '<br>' +
'modifiers: ' + s(binding.modifiers) + '<br>' +
'vnode keys: ' + Object.keys(vnode).join(', ')
}
})
new Vue({
el: '#app',
data: {
message: '菜鸟教程!'
}
})
</script>
```

尝试一下 »

有时候我们不需要其他钩子函数，我们可以简写函数，如下格式：

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

  // 设置指令的背景颜色

  el.style.backgroundColor = binding.value.color

})
```

指令函数可接受所有合法的 JavaScript 表达式，以下实例传入了 JavaScript 对象：

实例

```
<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>
```

尝试一下 »

Vue.js 组件 Vue.js 路由

点我分享笔记

反馈/建议

Vue.js 路由

本章节我们将为大家介绍 Vue.js 路由。

Vue.js 路由允许我们通过不同的 URL 访问不同的内容。

通过 Vue.js 可以实现多视图的单页Web应用（single page web application，SPA）。

Vue.js 路由需要载入 vue-router 库

中文文档地址：vue-router文档。

安装

1、直接下载 / CDN

```
https://unpkg.com/vue-router/dist/vue-router.js
```

NPM

推荐使用淘宝镜像：

```
cnpm install vue-router
```

简单实例

Vue.js + vue-router 可以很简单的实现单页应用。

<router-link> 是一个组件，该组件用于设置一个导航链接，切换不同 HTML 内容。 **to** 属性为目标地址， 即要显示的内容。

以下实例中我们将 vue-router 加进来，然后配置组件和路由映射，再告诉 vue-router 在哪里渲染它们。代码如下所示：

HTML 代码

```
<script src="https://unpkg.com/vue/dist/vue.js"></script>
<script src="https://unpkg.com/vue-router/dist/vue-router.js"></script>
<div id="app">
<h1>Hello App!</h1>
<p>
<!-- 使用 router-link 组件来导航. -->
<!-- 通过传入 `to` 属性指定链接. -->
<!-- <router-link> 默认会被渲染成一个 `<a>` 标签 -->
<router-link to="/foo">Go to Foo</router-link>
<router-link to="/bar">Go to Bar</router-link>
</p>
<!-- 路由出口 -->
<!-- 路由匹配到的组件将渲染在这里 -->
<router-view></router-view>
</div>
```

JavaScript 代码

```
// 0. 如果使用模块化机制编程，导入 Vue 和 VueRouter，要调用 Vue.use(VueRouter)
// 1. 定义（路由）组件。
// 可以从其他文件 import 进来
const Foo = { template: '<div>foo</div>' }
const Bar = { template: '<div>bar</div>' }
// 2. 定义路由
```

```
// 每个路由应该映射一个组件。 其中"component" 可以是
// 通过 Vue.extend() 创建的组件构造器，
// 或者，只是一个组件配置对象。
// 我们晚点再讨论嵌套路由。
const routes = [
{ path: '/foo', component: Foo },
{ path: '/bar', component: Bar }
]
// 3. 创建 router 实例，然后传 `routes` 配置
// 你还可以传别的配置参数，不过先这么简单着吧。
const router = new VueRouter({
routes // （缩写）相当于 routes: routes
})
// 4. 创建和挂载根实例。
// 记得要通过 router 配置参数注入路由，
// 从而让整个应用都有路由功能
const app = new Vue({
router
}).$mount('#app')
// 现在，应用已经启动了！
```

尝试一下 »

点击过的导航链接都会加上样式 `class ="router-link-exact-active router-link-active"`。

<router-link> 相关属性

接下来我们可以了解下更多关于 <router-link> 的属性。

to

表示目标路由的链接。 当被点击后，内部会立刻把 to 的值传到 router.push()，所以这个值可以是一个字符串或者是描述目标位置的对象。

```
<!-- 字符串 -->

<router-link to="home">Home</router-link>

<!-- 渲染结果 -->

<a href="home">Home</a>



<!-- 使用 v-bind 的 JS 表达式 -->

<router-link v-bind:to="'home'">Home</router-link>



<!-- 不写 v-bind 也可以，就像绑定别的属性一样 -->

<router-link :to="'home'">Home</router-link>



<!-- 同上 -->

<router-link :to="{ path: 'home' }">Home</router-link>



<!-- 命名的路由 -->

<router-link :to="{ name: 'user', params: { userId: 123 }}">User</router-link>
```

```
<!-- 带查询参数，下面的结果为 /register?plan=private -->

<router-link :to="{ path: 'register', query: { plan: 'private' }}">Register</router-link>
```

replace

设置 replace 属性的话，当点击时，会调用 router.replace() 而不是 router.push()，导航后不会留下 history 记录。

```
<router-link :to="{ path: '/abc'}" replace></router-link>
```

append

设置 append 属性后，则在当前 (相对) 路径前添加基路径。例如，我们从 /a 导航到一个相对路径 b，如果没有配置 append，则路径为 /b，如果配了，则为 /a/b

```
<router-link :to="{ path: 'relative/path'}" append></router-link>
```

tag

有时候想要 <router-link> 渲染成某种标签，例如 。 于是我们使用 tag prop 类指定何种标签，同样它还是会监听点击，触发导航。

```
<router-link to="/foo" tag="li">foo</router-link>

<!-- 渲染结果 -->

<li>foo</li>
```

active-class

设置 链接激活时使用的 CSS 类名。可以通过以下代码来替代。

```
<style>

   ._active{

      background-color : red;

   }

</style>

<p>

   <router-link v-bind:to = "{ path: '/route1'}" active-class = "_active">Router Link 1</router-link>

   <router-link v-bind:to = "{ path: '/route2'}" tag = "span">Router Link 2</router-link>

</p>
```

注意这里 class 使用 `active_class="_active"`。

exact-active-class

配置当链接被精确匹配的时候应该激活的 class。可以通过以下代码来替代。

```
<p>

   <router-link v-bind:to = "{ path: '/route1'}" exact-active-class = "_active">Router Link 1</router-link>

   <router-link v-bind:to = "{ path: '/route2'}" tag = "span">Router Link 2</router-link>

</p>
```

event

声明可以用来触发导航的事件。可以是一个字符串或是一个包含字符串的数组。

```
<router-link v-bind:to = "{ path: '/route1'}" event = "mouseover">Router Link 1</router-link>
```

以上代码设置了 event 为 mouseover，及在鼠标移动到 Router Link 1 上时导航的 HTML 内容会发生改变。

NPM 路由实例

接下来我们演示了一个使用 npm 简单的路由实例，开始前，请先下载该实例源代码：

路由实例

你也可以在 Github 上下载：https://github.com/chrisvfritz/vue-2.0-simple-routing-example

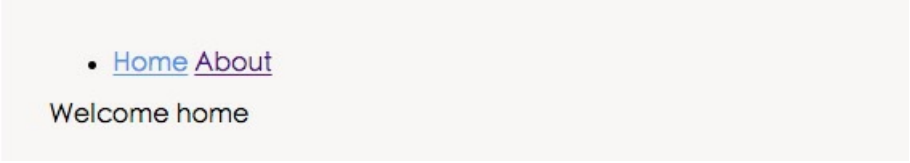
下载完后，解压该目录，重命名目录为 vue-demo，vu 并进入该目录，执行以下命令：

```
# 安装依赖，使用淘宝资源命令 cnpm

cnpm install



# 启动应用，地址为 localhost:8080

cnpm run dev
```

如果你需要发布到正式环境可以执行以下命令：

```
cnpm run build
```

执行成功后，访问 http://localhost:8080 即可看到如下界面：



Vue.js 自定义指令 Vue.js 起步

点我分享笔记

反馈/建议

Vue.js 过渡 & 动画

本章节我们主要讨论 Vue.js 的过渡效果与动画效果。

过渡

Vue 在插入、更新或者移除 DOM 时，提供多种不同方式的应用过渡效果。

Vue 提供了内置的过渡封装组件，该组件用于包裹要实现过渡效果的组件。

语法格式

```
<transition name = "nameoftransition">

   <div></div>

</transition>
```

我们可以通过以下实例来理解 Vue 的过渡是如何实现的：

实例

```
<div id = "databinding">
<button v-on:click = "show = !show">点我</button>
<transition name = "fade">
<p v-show = "show" v-bind:style = "styleobj">动画实例</p>
</transition>
</div>
<script type = "text/javascript">
var vm = new Vue({
el: '#databinding',
data: {
show:true,
styleobj :{
fontSize:'30px',
color:'red'
}
},
methods : {
}
});
</script>
```

尝试一下 »

实例中通过点击 "点我" 按钮将变量 show 的值从 true 变为 false。如果为 true 显示子元素 p 标签的内容。

下面这段代码展示了 transition 标签包裹了 p 标签：

```
<transition name = "fade">

    <p v-show = "show" v-bind:style = "styleobj">动画实例</p>

</transition>
```

过渡其实就是一个淡入淡出的效果。Vue在元素显示与隐藏的过渡中，提供了 6 个 class 来切换：

v-enter：定义进入过渡的开始状态。在元素被插入之前生效，在元素被插入之后的下一帧移除。

v-enter-active：定义进入过渡生效时的状态。在整个进入过渡的阶段中应用，在元素被插入之前生效，在过渡/动画完成之后移除。这个类可以被用来定义进入过渡的过程时间，延迟和曲线函数。

v-enter-to: **2.1.8版及以上** 定义进入过渡的结束状态。在元素被插入之后下一帧生效 (与此同时 v-enter 被移除)，在过渡/动画完成之后移除。

v-leave: 定义离开过渡的开始状态。在离开过渡被触发时立刻生效，下一帧被移除。

v-leave-active：定义离开过渡生效时的状态。在整个离开过渡的阶段中应用，在离开过渡被触发时立刻生效，在过渡/动画完成之后移除。这个类可以被用来定义离开过渡的过程时间，延迟和曲线函数。

v-leave-to: **2.1.8版及以上** 定义离开过渡的结束状态。在离开过渡被触发之后下一帧生效 (与此同时 v-leave 被删除)，在过渡/动画完成之后移除。

对于这些在过渡中切换的类名来说，如果你使用一个没有名字的 <transition>，则 v- 是这些类名的默认前缀。如果你使用了 <transition name="my-transition">，那么 v-enter 会替换为 my-transition-enter。

v-enter-active 和 v-leave-active 可以控制进入/离开过渡的不同的缓和曲线，在下面章节会有个示例说明。

CSS 过渡

通常我们都使用 CSS 过渡来实现效果。

如下实例：

实例

```
<div id = "databinding">
<button v-on:click = "show = !show">点我</button>
<transition name="slide-fade">
<p v-if="show">hello</p>
</transition>
</div>
<script type = "text/javascript">
new Vue({
el: '#databinding',
data: {
show: true
}
})
</script>
```

尝试一下 »

CSS 动画

CSS 动画用法类似 CSS 过渡，但是在动画中 v-enter 类名在节点插入 DOM 后不会立即删除，而是在 animationend 事件触发时删除。

实例

```
<div id = "databinding">
<button v-on:click = "show = !show">点我</button>
<transition name="bounce">
<p v-if="show">菜鸟教程 -- 学的不仅是技术，更是梦想！！！</p>
</transition>
</div>
<script type = "text/javascript">
new Vue({
el: '#databinding',
data: {
show: true
}
})
</script>
```

尝试一下 »

自定义过渡的类名

我们可以通过以下特性来自定义过渡类名：

enter-class

enter-active-class

enter-to-class (2.1.8+)

leave-class

leave-active-class

leave-to-class (2.1.8+)

自定义过渡的类名优先级高于普通的类名，这样就能很好的与第三方（如：animate.css）的动画库结合使用。

实例

```
<div id = "databinding">
<button v-on:click = "show = !show">点我</button>
<transition
name="custom-classes-transition"
enter-active-class="animated tada"
leave-active-class="animated bounceOutRight"
>
<p v-if="show">菜鸟教程 -- 学的不仅是技术，更是梦想！！！</p>
</transition>
</div>
<script type = "text/javascript">
new Vue({
el: '#databinding',
data: {
show: true
}
})
</script>
```

尝试一下 »

同时使用过渡和动画

Vue 为了知道过渡的完成，必须设置相应的事件监听器。它可以是 transitionend 或 animationend ，这取决于给元素应用的 CSS 规则。如果你使用其中任何一种，Vue 能自动识别类型并设置监听。

但是，在一些场景中，你需要给同一个元素同时设置两种过渡动效，比如 animation 很快的被触发并完成了，而 transition 效果还没结束。在这种情况中，你就需要使用 type 特性并设置 animation 或 transition 来明确声明你需要 Vue 监听的类型。

显性的过渡持续时间

在很多情况下，Vue 可以自动得出过渡效果的完成时机。默认情况下，Vue 会等待其在过渡效果的根元素的第一个 transitionend 或 animation end 事件。然而也可以不这样设定——比如，我们可以拥有一个精心编排的一系列过渡效果，其中一些嵌套的内部元素相比于过渡效果的根元素有延迟的或更长的过渡效果。

在这种情况下你可以用 <transition> 组件上的 duration 属性定制一个显性的过渡持续时间 (以毫秒计)：

```
<transition :duration="1000">...</transition>
```

你也可以定制进入和移出的持续时间：

```
<transition :duration="{ enter: 500, leave: 800 }">...</transition>
```

JavaScript 钩子

可以在属性中声明 JavaScript 钩子:

HTML 代码：

```
<transition
v-on:before-enter="beforeEnter"
v-on:enter="enter"
v-on:after-enter="afterEnter"
v-on:enter-cancelled="enterCancelled"
```

```
v-on:before-leave="beforeLeave"
v-on:leave="leave"
v-on:after-leave="afterLeave"
v-on:leave-cancelled="leaveCancelled"
>
<!-- ... -->
</transition>
```

JavaScript 代码：

```
// ...
methods: {
// --------
// 进入中
// --------
beforeEnter: function (el) {
// ...
},
// 此回调函数是可选项的设置
// 与 CSS 结合时使用
enter: function (el, done) {
// ...
done()
},
afterEnter: function (el) {
// ...
},
enterCancelled: function (el) {
// ...
},
// --------
// 离开时
// --------
beforeLeave: function (el) {
// ...
},
// 此回调函数是可选项的设置
// 与 CSS 结合时使用
leave: function (el, done) {
// ...
done()
},
afterLeave: function (el) {
// ...
},
// leaveCancelled 只用于 v-show 中
leaveCancelled: function (el) {
// ...
}
}
```

这些钩子函数可以结合 CSS transitions/animations 使用，也可以单独使用。

当只用 JavaScript 过渡的时候，**在 `enter` 和 `leave` 中必须使用 `done` 进行回调**。否则，它们将被同步调用，过渡会立即完成。

推荐对于仅使用 JavaScript 过渡的元素添加 `v-bind:css="false"`，Vue 会跳过 CSS 的检测。这也可以避免过渡过程中 CSS 的影响。

一个使用 Velocity.js 的简单例子：

实例

```
<div id = "databinding">
<button v-on:click = "show = !show">点我</button>
<transition
v-on:before-enter="beforeEnter"
v-on:enter="enter"
v-on:leave="leave"
v-bind:css="false"
>
<p v-if="show">菜鸟教程 -- 学的不仅是技术，更是梦想！！！</p>
</transition>
</div>
<script type = "text/javascript">
```

```
new Vue({
el: '#databinding',
data: {
show: false
},
methods: {
beforeEnter: function (el) {
el.style.opacity = 0
el.style.transformOrigin = 'left'
},
enter: function (el, done) {
Velocity(el, { opacity: 1, fontSize: '1.4em' }, { duration: 300 })
Velocity(el, { fontSize: '1em' }, { complete: done })
},
leave: function (el, done) {
Velocity(el, { translateX: '15px', rotateZ: '50deg' }, { duration: 600 })
Velocity(el, { rotateZ: '100deg' }, { loop: 2 })
Velocity(el, {
rotateZ: '45deg',
translateY: '30px',
translateX: '30px',
opacity: 0
}, { complete: done })
}
}
})
</script>
```

尝试一下 »

初始渲染的过渡

可以通过 appear 特性设置节点在初始渲染的过渡

```
<transition appear>
  <!-- ... -->
</transition>
```

这里默认和进入/离开过渡一样，同样也可以自定义 CSS 类名。

```
<transition
  appear
  appear-class="custom-appear-class"
  appear-to-class="custom-appear-to-class" (2.1.8+)
  appear-active-class="custom-appear-active-class"
>
  <!-- ... -->
</transition>
```

自定义 JavaScript 钩子：

```
<transition
  appear
  v-on:before-appear="customBeforeAppearHook"
  v-on:appear="customAppearHook"
  v-on:after-appear="customAfterAppearHook"
  v-on:appear-cancelled="customAppearCancelledHook"
>
  <!-- ... -->
</transition>
```

多个元素的过渡

我们可以设置多个元素的过渡，一般列表与描述：
需要注意的是当有相同标签名的元素切换时，需要通过 key 特性设置唯一的值来标记以让 Vue 区分它们，否则 Vue 为了效率只会替换相同标签内部的内容。

```
<transition>
  <table v-if="items.length > 0">
    <!-- ... -->
  </table>
  <p v-else>抱歉，没有找到您查找的内容。</p>
</transition>
```

如下实例：

```
<transition>
  <button v-if="isEditing" key="save">
    Save
  </button>
  <button v-else key="edit">
    Edit
  </button>
</transition>
```

在一些场景中，也可以通过给同一个元素的 key 特性设置不同的状态来代替 v-if 和 v-else，上面的例子可以重写为：

```
<transition>
  <button v-bind:key="isEditing">
    {{ isEditing ? 'Save' : 'Edit' }}
```

```
  </button>
</transition>
```

使用多个 `v-if` 的多个元素的过渡可以重写为绑定了动态属性的单个元素过渡。例如：

```
<transition>
  <button v-if="docState === 'saved'" key="saved">
    Edit
  </button>
  <button v-if="docState === 'edited'" key="edited">
    Save
  </button>
  <button v-if="docState === 'editing'" key="editing">
    Cancel
  </button>
</transition>
```

可以重写为：

```
<transition>
  <button v-bind:key="docState">
    {{ buttonMessage }}
  </button>
</transition>



// ...
computed: {
  buttonMessage: function () {
    switch (this.docState) {
      case 'saved': return 'Edit'
      case 'edited': return 'Save'
      case 'editing': return 'Cancel'
    }
  }
}
```

Vue.js 混入

混入 (mixins)定义了一部分可复用的方法或者计算属性。混入对象可以包含任意组件选项。当组件使用混入对象时，所有混入对象的选项将被混入该组件本身的选项。

来看一个简单的实例：

实例

```
var vm = new Vue({
el: '#databinding',
data: {
},
methods : {
},
});
// 定义一个混入对象
var myMixin = {
created: function () {
this.startmixin()
},
methods: {
startmixin: function () {
document.write("欢迎来到混入实例");
}
}
};
var Component = Vue.extend({
mixins: [myMixin]
})
var component = new Component();
```

尝试一下 »

选项合并

当组件和混入对象含有同名选项时，这些选项将以恰当的方式混合。

比如，数据对象在内部会进行浅合并 (一层属性深度)，在和组件的数据发生冲突时以组件数据优先。

以下实例中，Vue 实例与混入对象包含了相同的方法。从输出结果可以看出两个选项合并了。

实例

```
var mixin = {
created: function () {
```

```
document.write('混入调用' + '<br>')
}
}
new Vue({
mixins: [mixin],
created: function () {
document.write('组件调用' + '<br>')
}
});
```

输出结果为：

```
混入调用

组件调用
```

尝试一下 »

如果 methods 选项中有相同的函数名，则 Vue 实例优先级会较高。如下实例，Vue 实例与混入对象的 methods 选项都包含了相同的函数：

实例

```
var mixin = {
methods: {
hellworld: function () {
document.write('HelloWorld 方法' + '<br>');
},
samemethod: function () {
document.write('Mixin: 相同方法名' + '<br>');
}
}
};
var vm = new Vue({
mixins: [mixin],
methods: {
start: function () {
document.write('start 方法' + '<br>');
},
samemethod: function () {
document.write('Main: 相同方法名' + '<br>');
}
}
});
vm.hellworld();
vm.start();
vm.samemethod();
```

输出结果为：

```
HelloWorld 方法

start 方法

Main: 相同方法名
```

尝试一下 »

以上实例，我们调用了以下三个方法：

```
vm.hellworld();
```

```
vm.start();

vm.samemethod();
```

从输出结果 methods 选项中如果碰到相同的函数名则 Vue 实例有更高的优先级会执行输出。

全局混入

也可以全局注册混入对象。注意使用！ 一旦使用全局混入对象，将会影响到 所有 之后创建的 Vue 实例。使用恰当时，可以为自定义对象注入处理逻辑。

实例

```
// 为自定义的选项 'myOption' 注入一个处理器。
Vue.mixin({
created: function () {
var myOption = this.$options.myOption
if (myOption) {
console.log(myOption)
}
}
})
new Vue({
myOption: 'hello!'
})
// => "hello!"
```

尝试一下 »

谨慎使用全局混入对象，因为会影响到每个单独创建的 Vue 实例 (包括第三方模板)。

Vue.js 过渡 & 动画 Vue.js Ajax(vue-resource)

Vue.js 混入 Vue.js 响应接口

Vue.js Ajax(vue-resource)

Vue 要实现异步加载需要使用到 vue-resource 库。

```
<script src="https://cdn.bootcss.com/vue-resource/1.5.1/vue-resource.min.js"></script>
```

Get 请求

以下是一个简单的 Get 请求实例，请求地址是一个简单的 txt 文本：

实例

```
window.onload = function(){
var vm = new Vue({
el:'#box',
data:{
msg:'Hello World!',
},
methods:{
get:function(){
//发送get请求
this.$http.get('/try/ajax/ajax_info.txt').then(function(res){
document.write(res.body);
},function(){
console.log('请求失败处理');
});
}
}
});
}
```

尝试一下 »

如果需要传递数据，可以使用 `this.$http.get('get.php',jsonData)` 格式，第二个参数 **jsonData** 就是传到后端的数据。

```
this.$http.get('get.php',{a:1,b:2}).then(function(res){

    document.write(res.body);

},function(res){

    console.log(res.status);

});
```

post 请求

post 发送数据到后端，需要第三个参数 `{emulateJSON:true}`。

emulateJSON 的作用： 如果Web服务器无法处理编码为 application/json 的请求，你可以启用 emulateJSON 选项。

实例

```
window.onload = function(){
var vm = new Vue({
el:'#box',
data:{
msg:'Hello World!',
},
methods:{
post:function(){
//发送 post 请求
this.$http.post('/try/ajax/demo_test_post.php',{name:"菜鸟教程",url:"http://www.runoob.com"},{emulateJSON:true}).th
en(function(res){
document.write(res.body);
},function(res){
console.log(res.status);
});
}
}
});
}
```

尝试一下 »

demo_test_post.php 代码如下：

```
<?php

$name = isset($_POST['name']) ? htmlspecialchars($_POST['name']) : '';
```

```
$city = isset($_POST['url']) ? htmlspecialchars($_POST['url']) : '';

echo '网站名: ' . $name;

echo "\n";

echo 'URL 地址: ' .$city;

?>
```

语法 & API

你可以使用全局对象方式 Vue.http 或者在一个 Vue 实例的内部使用 this.$http来发起 HTTP 请求。

```
// 基于全局Vue对象使用http

Vue.http.get('/someUrl', [options]).then(successCallback, errorCallback);

Vue.http.post('/someUrl', [body], [options]).then(successCallback, errorCallback);



// 在一个Vue实例内使用$http

this.$http.get('/someUrl', [options]).then(successCallback, errorCallback);

this.$http.post('/someUrl', [body], [options]).then(successCallback, errorCallback);
```

vue-resource 提供了 7 种请求 API(REST 风格):

```
get(url, [options])

head(url, [options])

delete(url, [options])

jsonp(url, [options])

post(url, [body], [options])

put(url, [body], [options])

patch(url, [body], [options])
```

除了 jsonp 以外，另外 6 种的 API 名称是标准的 HTTP 方法。

options 参数说明:

参数	类型	描述
url	string	请求的目标URL
body	Object, FormData, string	作为请求体发送的数据
headers	Object	作为请求头部发送的头部对象
params	Object	作为URL参数的参数对象
method	string	HTTP方法 (例如GET，POST，...)
timeout	number	请求超时（单位：毫秒） (0表示永不超时)

参数	类型	描述
before	function(request)	在请求发送之前修改请求的回调函数
progress	function(event)	用于处理上传进度的回调函数 ProgressEvent
credentials	boolean	是否需要出示用于跨站点请求的凭据
emulateHTTP	boolean	是否需要通过设置X-HTTP-Method-Override头部并且以传统POST方式发送PUT，PATCH和DELETE请求。
emulateJSON	boolean	设置请求体的类型为application/x-www-form-urlencoded

通过如下属性和方法处理一个请求获取到的响应对象：

属性	类型	描述
url	string	响应的URL源
body	Object, Blob, string	响应体数据
headers	Header	请求头部对象
ok	boolean	当HTTP响应码为200到299之间的数值时该值为true
status	number	HTTP响应吗
statusText	string	HTTP响应状态
方法	**类型**	**描述**
text()	约定值	以字符串方式返回响应体
json()	约定值	以格式化后的json对象方式返回响应体
blob()	约定值	以二进制Blob对象方式返回响应体

□ Vue.js 混入 Vue.js 响应接口 □

□ Vue.js Ajax(vue-resource) Vue.js 实例 □

Vue.js 响应接口

Vue 可以添加数据动态响应接口。

例如以下实例，我们通过使用 $watch 属性来实现数据的监听，$watch 必须添加在 Vue 实例之外才能实现正确的响应。

实例中通过点击按钮自动加 1。setTimeout 设置两秒后计算器的值加上 20 。

实例

```
<div id = "app">
<p style = "font-size:25px;">计数器: {{ counter }}</p>
<button @click = "counter++" style = "font-size:25px;">点我</button>
</div>
<script type = "text/javascript">
var vm = new Vue({
el: '#app',
data: {
counter: 1
}
});
vm.$watch('counter', function(nval, oval) {
alert('计数器值的变化 :' + oval + ' 变为 ' + nval + '!');
});
setTimeout(
function(){
vm.counter = 20;
},2000
);
</script>
```

尝试一下 »

Vue 不允许在已经创建的实例上动态添加新的根级响应式属性。

Vue 不能检测到对象属性的添加或删除，最好的方式就是在初始化实例前声明根级响应式属性，哪怕只是一个空值。

如果我们需要在运行过程中实现属性的添加或删除，则可以使用全局 Vue，Vue.set 和 Vue.delete 方法。

Vue.set

Vue.set 方法用于设置对象的属性，它可以解决 Vue 无法检测添加属性的限制，语法格式如下：

```
Vue.set( target, key, value )
```

参数说明：

target: 可以是对象或数组

key : 可以是字符串或数字

value: 可以是任何类型

实例

```
<div id = "app">
<p style = "font-size:25px;">计数器: {{ products.id }}</p>
<button @click = "products.id++" style = "font-size:25px;">点我</button>
</div>
<script type = "text/javascript">
var myproduct = {"id":1, name:"book", "price":"20.00"};
var vm = new Vue({
el: '#app',
data: {
counter: 1,
products: myproduct
}
});
vm.products.qty = "1";
console.log(vm);
vm.$watch('counter', function(nval, oval) {
alert('计数器值的变化 :' + oval + ' 变为 ' + nval + '!');
});
</script>
```

尝试一下 »

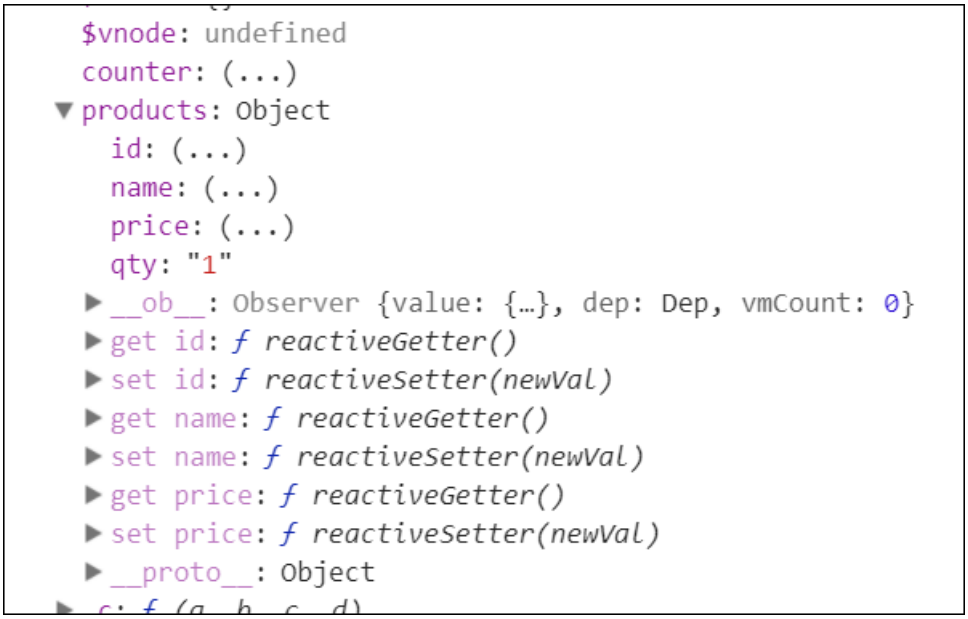
在以上实例中，使用以下代码在开始时创建了一个变量 myproduct：

```
var myproduct = {"id":1, name:"book", "price":"20.00"};
```

该变量在赋值给了 Vue 实例的 data 对象：
var vm = new Vue({ el: '#app', data: { counter: 1, products: myproduct } });
如果我们想给 myproduct 数组添加一个或多个属性，我们可以在 Vue 实例创建后使用以下代码：

```
vm.products.qty = "1";
```

查看控制台输出：



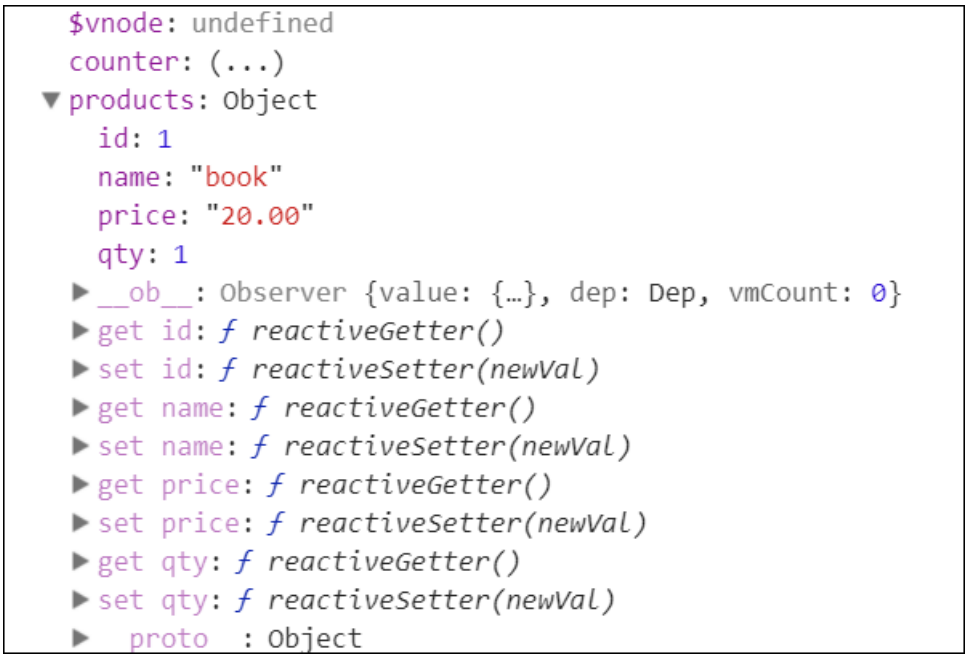
如上图看到的，在产品中添加了数量属性 qty，但是 get/set 方法只可用于 id，name 和 price 属性，却不能在 qty 属性中使用。

我们不能通过添加 Vue 对象来实现响应。 Vue 主要在开始时创建所有属性。 如果我们要实现这个功能，可以通过 Vue.set 来实现：

实例

```
<div id = "app">
<p style = "font-size:25px;">计数器: {{ products.id }}</p>
<button @click = "products.id++" style = "font-size:25px;">点我</button>
</div>
<script type = "text/javascript">
var myproduct = {"id":1, name:"book", "price":"20.00"};
var vm = new Vue({
el: '#app',
data: {
counter: 1,
products: myproduct
}
});
Vue.set(myproduct, 'qty', 1);
console.log(vm);
vm.$watch('counter', function(nval, oval) {
alert('计数器值的变化 :' + oval + ' 变为 ' + nval + '!');
});
</script>
```

尝试一下 »



从控制台输出的结果可以看出 get/set 方法可用于qty 属性。

Vue.delete

Vue.delete 用于删除动态添加的属性 语法格式：

```
Vue.delete( target, key )
```

参数说明：

target: 可以是对象或数组

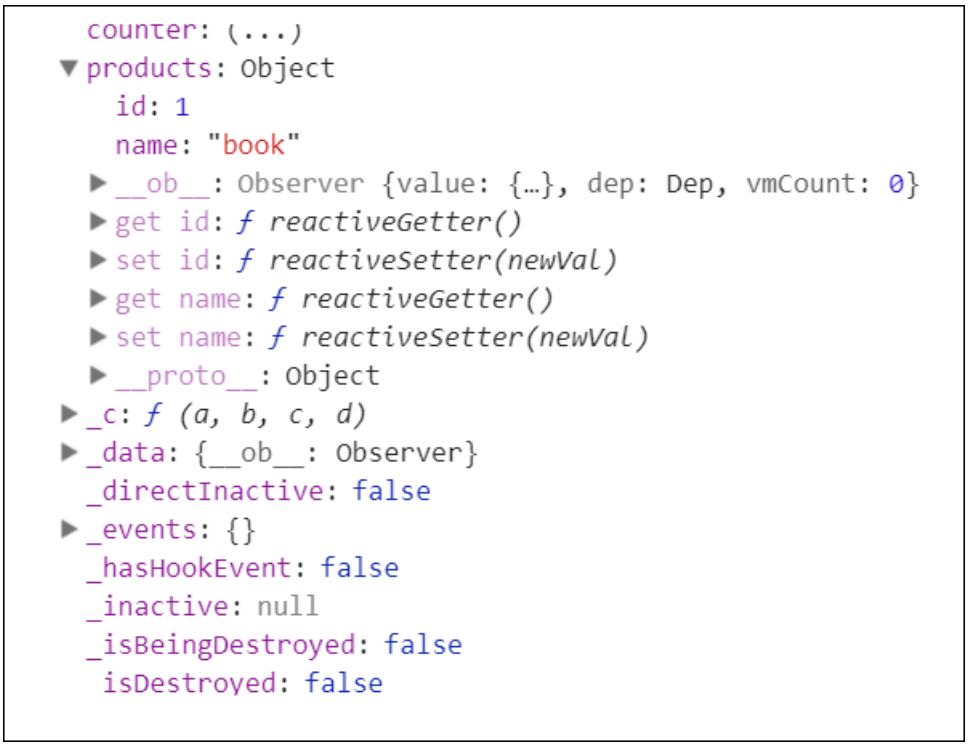
key : 可以是字符串或数字

实例

```
<div id = "app">
<p style = "font-size:25px;">计数器: {{ products.id }}</p>
<button @click = "products.id++" style = "font-size:25px;">点我</button>
</div>
<script type = "text/javascript">
var myproduct = {"id":1, name:"book", "price":"20.00"};
var vm = new Vue({
el: '#app',
data: {
counter: 1,
products: myproduct
}
});
Vue.delete(myproduct, 'price');
console.log(vm);
vm.$watch('counter', function(nval, oval) {
alert('计数器值的变化 :' + oval + ' 变为 ' + nval + '!');
});
</script>
```

尝试一下 »

以上实例中我们使用 Vue.delete 来删除 price 属性。以下是控制台输出结果：



从上图输出结果中，我们可以看到 price 属性已删除，只剩下了 id 和 name 属性，price 属性的 get/set 方法也已删除。

Vue.js Ajax(vue-resource) Vue.js 实例

点我分享笔记

反馈/建议

Vue.js 响应接口

Vue.js 实例

本章节为大家介绍几个 Vue.js 实例，通过实例练习来巩固学到的知识点。

导航菜单实例

导航菜单

创建一个简单的导航菜单：

```
<div id="main">
<!-- 激活的菜单样式为 active 类 -->
<!-- 为了阻止链接在点击时跳转，我们使用了 "prevent" 修饰符 (preventDefault 的简称)。 -->
<nav v-bind:class="active" v-on:click.prevent>
<!-- 当菜单上的链接被点击时，我们调用了 makeActive 方法，该方法在 Vue 实例中创建。 -->
<a href="#" class="home" v-on:click="makeActive('home')">Home</a>
<a href="#" class="projects" v-on:click="makeActive('projects')">Projects</a>
<a href="#" class="services" v-on:click="makeActive('services')">Services</a>
<a href="#" class="contact" v-on:click="makeActive('contact')">Contact</a>
```

```
</nav>
<!-- 以下 "active" 变量会根据当前选中的值来自动变换 -->
<p>您选择了 <b>{{active}} 菜单</b></p>
</div>
<script>
// 创建一个新的 Vue 实例
var demo = new Vue({
// DOM 元素，挂载视图模型
el: '#main',
// 定义属性，并设置初始值
data: {
active: 'home'
},
// 点击菜单使用的函数
methods: {
makeActive: function(item){
// 模型改变，视图会自动更新
this.active = item;
}
}
});
</script>
```

尝试一下 »

编辑文本实例

文本编辑

点击指定文本编辑内容：

```
<!-- v-cloak 隐藏未编译的变量，直到 Vue 实例准备就绪。 -->
<!-- 元素点击后 hideTooltp() 方法被调用 -->
<div id="main" v-cloak v-on:click="hideTooltip">
<!-- 这是一个提示框
v-on:clock.stop 是一个点击事件处理器，stop 修饰符用于阻止事件传递
v-if 用来判断 show_tooltip 为 true 时才显示 -->
<div class="tooltip" v-on:click.stop v-if="show_tooltip">
<!-- v-model 绑定了文本域的内容
在文本域内容改变时，对应的变量也会实时改变 -->
<input type="text" v-model="text_content" />
</div>
<!-- 点击后调用 "toggleTooltip" 方法并阻止事件传递 -->
<!-- "text_content" 变量根据文本域内容的变化而变化 -->
<p v-on:click.stop="toggleTooltip">{{text_content}}</p>
</div>
<script>
var demo = new Vue({
el: '#main',
data: {
show_tooltip: false,
text_content: '点我，并编辑内容。'
},
methods: {
hideTooltip: function(){
// 在模型改变时，视图也会自动更新
this.show_tooltip = false;
},
toggleTooltip: function(){
this.show_tooltip = !this.show_tooltip;
}
}
})
</script>
```

尝试一下 »

订单列表实例

订单列表

创建一个订单列表，并计算总价：

```
<form id="main" v-cloak>
<h1>Services</h1>
<ul>
<!-- 循环输出 services 数组，设置选项点击后的样式 -->
<li v-for="service in services" v-on:click="toggleActive(service)" v-bind:class="{ 'active': service.active}">
<!-- 显示订单中的服务名，价格
Vue.js 定义了货币过滤器，用于格式化价格 -->
{{service.name}} <span>{{service.price | currency}}</span>
</li>
</ul>
<div class="total">
<!-- 计算所有服务的价格，并格式化货币 -->
Total: <span>{{total() | currency}}</span>
</div>
</form>
<script>
// 自定义过滤器 "currency".
Vue.filter('currency', function (value) {
return '$' + value.toFixed(2);
});
var demo = new Vue({
el: '#main',
data: {
// 定义模型属性 the model properties. The view will loop
// 视图将循环输出数组的数据
services: [
{
name: 'Web Development',
price: 300,
active:true
},{
name: 'Design',
price: 400,
active:false
},{
name: 'Integration',
price: 250,
active:false
},{
name: 'Training',
price: 220,
active:false
}
]
},
methods: {
toggleActive: function(s){
s.active = !s.active;
},
total: function(){
var total = 0;
this.services.forEach(function(s){
if (s.active){
total+= s.price;
}
});
return total;
}
}
});
</script>
```

尝试一下 »

搜索页面实例

搜索页面

在输入框输入搜索内容，列表显示配置的列表：

```
<form id="main" v-cloak>
<div class="bar">
<!-- searchString 模型与文本域创建绑定 -->
<input type="text" v-model="searchString" placeholder="输入搜索内容" />
</div>
<ul>
<!-- 循环输出数据 -->
<li v-for="article in filteredArticles">
<a v-bind:href="article.url"><img v-bind:src="article.image" /></a>
<p>{{article.title}}</p>
</li>
</ul>
</form>
<script>
var demo = new Vue({
el: '#main',
data: {
searchString: "",
// 数据模型，实际环境你可以根据 Ajax 来获取
articles: [
{
"title": "What You Need To Know About CSS Variables",
"url": "https://www.runoob.com/css/css-tutorial.html",
"image": "https://static.runoob.com/images/icon/css.png"
},
{
"title": "Freebie: 4 Great Looking Pricing Tables",
"url": "https://www.runoob.com/html/html-tutorial.html",
"image": "https://static.runoob.com/images/icon/html.png"
},
{
"title": "20 Interesting JavaScript and CSS Libraries for February 2016",
"url": "https://www.runoob.com/css3/css3-tutorial.html",
"image": "https://static.runoob.com/images/icon/css3.png"
},
{
"title": "Quick Tip: The Easiest Way To Make Responsive Headers",
"url": "https://www.runoob.com/css3/css3-tutorial.html",
"image": "https://static.runoob.com/images/icon/css3.png"
},
{
"title": "Learn SQL In 20 Minutes",
"url": "https://www.runoob.com/sql/sql-tutorial.html",
"image": "https://static.runoob.com/images/icon/sql.png"
},
{
"title": "Creating Your First Desktop App With HTML, JS and Electron",
"url": "https://www.runoob.com/js/js-tutorial.html",
"image": "https://static.runoob.com/images/icon/html.png"
}
]
},
computed: {
// 计算数学，匹配搜索
filteredArticles: function () {
var articles_array = this.articles,
searchString = this.searchString;
if(!searchString){
return articles_array;
}
searchString = searchString.trim().toLowerCase();
articles_array = articles_array.filter(function(item){
if(item.title.toLowerCase().indexOf(searchString) !== -1){
return item;
}
})
// 返回过来后的数组
return articles_array;;
}
}
});
</script>
```

尝试一下 »

切换不同布局实例

切换不同布局

点击右上角的按钮来切换不同页面布局：

```
<form id="main" v-cloak>
<div class="bar">
<!-- 两个按钮用于切换不同的列表布局 -->
<a class="list-icon" v-bind:class="{ 'active': layout == 'list'}" v-on:click="layout = 'list'"></a>
<a class="grid-icon" v-bind:class="{ 'active': layout == 'grid'}" v-on:click="layout = 'grid'"></a>
</div>
<!-- 我们设置了两套布局页面。使用哪套依赖于 "layout" 绑定 -->
<ul v-if="layout == 'grid'" class="grid">
<!-- 使用大图，没有文本 -->
<li v-for="a in articles">
<a v-bind:href="a.url" target="_blank"><img v-bind:src="a.image.large" /></a>
</li>
</ul>
<ul v-if="layout == 'list'" class="list">
<!-- 使用小图及标题 -->
<li v-for="a in articles">
<a v-bind:href="a.url" target="_blank"><img v-bind:src="a.image.small" /></a>
<p>{{a.title}}</p>
</li>
</ul>
</form>
<script>
var demo = new Vue({
el: '#main',
data: {
// 视图模型，可能的值是 "grid" 或 "list"。
layout: 'grid',
articles: [{
"title": "HTML 教程",
"url": "https://www.runoob.com/html/html-tutorial.html",
"image": {
"large": "https://static.runoob.com/images/mix/htmlbig.png",
"small": "https://static.runoob.com/images/icon/html.png"
}
},
{
"title": "CSS 教程",
"url": "https://www.runoob.com/css/css-tutorial.html",
"image": {
"large": "https://static.runoob.com/images/mix/cssbig.png",
"small": "https://static.runoob.com/images/icon/css.png"
}
},
{
"title": "JS 教程",
"url": "https://www.runoob.com/js/js-tutorial.html",
"image": {
"large": "https://static.runoob.com/images/mix/jsbig.jpeg",
"small": "https://static.runoob.com/images/icon/js.png"
}
},
{
"title": "SQL 教程",
"url": "https://www.runoob.com/sql/sql-tutorial.html",
"image": {
"large": "https://static.runoob.com/images/mix/sqlbig.png",
"small": "https://static.runoob.com/images/icon/sql.png"
}
},
{
"title": "Ajax 教程",
"url": "https://www.runoob.com/ajax/ajax-tutorial.html",
"image": {
```

```
"large": "https://static.runoob.com/images/mix/ajaxbig.png",
"small": "https://static.runoob.com/images/icon/ajax.png"
}
},
{
"title": "Python 教程",
"url": "https://www.runoob.com/pyhton/pyhton-tutorial.html",
"image": {
"large": "https://static.runoob.com/images/mix/pythonbig.png",
"small": "https://static.runoob.com/images/icon/python.png"
}
}]
}
});
</script>
```

尝试一下 »

Vue.js 响应接口

点我分享笔记

反馈/建议