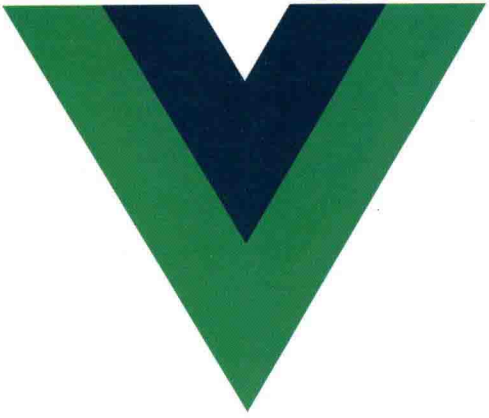
异步图书
www.epubit.com.cn

Vue.js前端开发

快速入门与专业应用

陈陆扬 著



学以致用

以实际项目为立足点，
拒绝纸上谈兵，
帮助你快速上手！

应用广泛

摸透Vue.js，无论是
桌面端还是移动端，
让你从容面对！

求职必备

国内互联网公司已广泛
使用Vue.js开发，
再不学就晚了！

中国工信出版集团
人民邮电出版社
POSTS & TELECOM PRESS

Vue.js前端开发

快速入门与专业应用

陈陆扬 著

人民邮电出版社
北京

图书在版编目（CIP）数据

Vue.js前端开发快速入门与专业应用 / 陈陆扬著
. -- 北京 : 人民邮电出版社, 2017.2
ISBN 978-7-115-44493-6

Ⅰ. ①V… Ⅱ. ①陈… Ⅲ. ①三维动画软件 Ⅳ.
①TP391.414

中国版本图书馆CIP数据核字(2017)第004640号

内 容 提 要

本书主要介绍 Vue.js 的使用方法和在实际项目中的运用，它既可以在一个页面中单独使用，也可以将整站都构建成单页面应用。为了便于理解，本书会从传统的开发角度切入，先从数据渲染、事件绑定等方面介绍在 Vue.js 中的使用方法，然后渐进到 Vue.js 自身的特性，例如数据绑定、过滤器、指令以及最重要的组件部分。除了框架用法外，本书还介绍了和 Vue.js 相关的重要插件和构建工具，这些工具有助于帮助用户构建一个完整的单页面应用，而不仅仅是停留在个人 DEMO 阶段的试验品。而对于复杂项目，Vue.js 也提供了对应的状态管理工具 Vuex，降低项目的开发和维护成本。鉴于完稿前，Vue.js 2.0 已正式发布完毕，本书也在相关用法上对比了 1.0 和 2.0 的区别，并补充了 render 函数和服务端渲染等特性。

本书适用于尚未接触过 MVVM 类前端框架的开发者，或者初步接触 Vue.js 的开发者，以及实际应用 Vue.js 开发项目的开发者。

◆ 著　　陈陆扬
责任编辑　赵　轩
责任印制　焦志炜

◆ 人民邮电出版社出版发行　　北京市丰台区成寿寺路 11 号
邮编　100164　　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
固安县铭成印刷有限公司印刷

◆ 开本：720×960　1/16
印张：12.75
字数：299 千字　　2017 年 2 月第 1 版
印数：1 – 3 000 册　　2017 年 2 月河北第 1 次印刷

定价：45.00 元

读者服务热线：(010)81055410　印装质量热线：(010)81055316
反盗版热线：(010)81055315

前言

PREFACE

近年来，前端框架的发展依旧火热，新的框架和名词依旧不停地出现在开发者眼前，而且开发模式也产生了一定的变化。目前来看，前端 MVVM 框架的出现给开发者带来了不小的便利，其中的代表就有 Angular.js、React.js 以及本书中将要介绍的 Vue.js。这些框架的产生使得开发者能从过去手动维护 DOM 状态的繁琐操作中解脱出来，尽可能地让 DOM 的更新操作是自动的，状态变化的时候就自动更新到正确的状态。不过，新框架的引入不可避免的就是学习成本的增加以及框架普及性的问题，相对于 Angular.js 和 React.js 而言，Vue.js 的学习曲线则比较平稳，上手比较简单，并且配合自身插件能。目前在 GitHub 上已经获得了超过 30000 的 star，成为了时下无论从实用性还是普遍性来说都是可靠的 MVVM 框架选择之一。

首次听说 Vue.js 的时候，都是介绍说体积小、适合移动端、使用简单，等等。但一开始对于新框架我一直持观望态度，毕竟前端框架更新太快，而且这又是个个人项目，仅由作者尤雨溪一人维护，不像 Angular.js 和 React.js 那样有公司支持。后来为了解决一个移动端的项目，我才正式接触了 Vue.js。由于项目本身天然存在组件这个概念，并且需要在手机上运行，调研后觉得应该没有比 Vue.js 更适合的工具了。在使用过程中，逐渐体会到了 Vue.js 的便利，数据绑定及组件系统对于提高开发效率和代码复用性来说都有相当大的帮助，并且初期对线上项目使用这种新框架的顾虑也渐渐消除了，即使随着后期复杂度的增加也并没有对项目的开发和维护成本造成影响。

本书主要从我自身的学习和开发经验出发，介绍了 Vue.js 的基础用法和特性，包括 Vue.js 的一些插件用法，用于解决客户端路由和大规模状态管理，以及打包发布等构建工具，便于正式用于线上环境。

最后，感谢 Vue.js 作者尤雨溪先生为前端开发者提供了这款优秀的框架，使得开发者能够更好地应对项目复杂度；也感谢人民邮电出版社的大力支持，写书的过程的确对人是一种折磨和考验；最后感谢每天早上 4 点多就开始叫我起床的两只猫，它们对本书的进度的确起到了很好的推动作用。

目 录

CONTENTS

第1章 Vue.js简介

近几年，互联网前端行业发展得依旧迅猛，涌现出了很多优秀的框架，同时这些框架也正在逐渐改变我们传统的前端开发方式。Google 的 AngularJS、Facebook 的 ReactJS，这些前端 MVC（MVVM）框架的出现和组件化开发的普及和规范化，既改变了原有的开发思维和方式，也使得前端开发者加快脚步，更新自己的知识结构。2014 年 2 月，原 Google 员工尤雨溪公开发布了自己的前端库——Vue.js，时至今日，Vue.js 在 GitHub 上已经收获超过 30000star，而且也有越来越多的开发者在实际的生产环境中运用它。

本书主要以 Vue.js 1.0.26 版本为基准进行说明，Vue.js 2.0 版本与之不同的地方，会在对应章节中说明。

1.1 Vue.js是什么

单独来讲，Vue.js 被定义成一个用来开发 Web 界面的前端库，是个非常轻量级的工具。Vue.js 本身具有响应式编程和组件化的特点。

所谓响应式编程，即为保持状态和视图的同步，这个在大多数前端 MV*（MVC/MVVM/MVW）框架，不管是早期的 backbone.js 还是现在 AngularJS 都对这一特性进行了实现（也称之为数据绑定），但这几者的实现方式和使用方式都不相同。相比而言，Vue.js 使用起来更为简单，也无需引入太多的新概念，声明实例 new Vue({ data : data }) 后自然对 data 里面的数据进行了视图上的绑定。修改 data 的数据，视图中对应数据也会随之更改。

Vue.js 的组件化理念和 ReactJS 异曲同工——“一切都是组件”，可以将任意封装好的代

码注册成标签，例如：Vue.component('example', Example)，可以在模板中以 <example></example> 的形式调用。如果组件抽象得合理，这在很大程度上能减少重复开发，而且配合Vue.js的周边工具vue-loader，我们可以将一个组件的CSS、HTML和js都写在一个文件里，做到模块化的开发。

除此之外，Vue.js也可以和一些周边工具配合起来，例如vue-router和vue-resource，支持了路由和异步请求，这样就满足了开发单页面应用的基本条件。

1.2 为什么要用Vue.js

相比较Angularjs和ReactJS，Vue.js一直以轻量级，易上手被称道。MVVM的开发模式也使前端从原先的DOM操作中解放出来，我们不再需要在维护视图和数据的统一上花大量的时间，只需要关注于data的变化，代码变得更加容易维护。虽然社区和插件并没有一些老牌的开源项目那么丰富，但满足日常的开发是没有问题的。Vue.js 2.0也已经发布了beta版本，渲染层基于一个轻量级的virtual-DOM实现，在大多数场景下初始化渲染速度和内存消耗都提升了2~4倍。而阿里也开源了weex(可以理解成ReactJS-Native和ReacJS的关系)，这也意味着Vue.js在移动端有了更多的可能性。

不过，对于为什么要选择使用一个框架，都需要建立在一定的项目基础上。如果脱离实际项目情况我们来谈某个框架的优劣，以及是否采用这种框架，我觉得是不够严谨的。

作为新兴的前端框架，Vue.js也抛弃了对IE8的支持，在移动端支持到Android 4.2+和iOS 7+。所以如果你在一家比较传统，还需要支持IE6的公司的话，你或许就可以考虑其他的解决方案了(或者说服你的老板)。另外，在传统的前后端混合(通过后端模板引擎渲染)的项目中，Vue.js也会受到一定的限制，Vue实例只能和后端模板文件混合在一起，获取的数据也需要依赖于后端的渲染，这在处理一些JSON对象和数组的时候会有些麻烦。

理想状态下，我们能直接在前后端分离的新项目中使用Vue.js最合适。这能最大程度上发挥Vue.js的优势和特性，熟悉后能极大的提升我们的开发效率以及代码的复用率。尤其是移动浏览器上，Vue.js压缩后只有18KB，而且没有其他的依赖。

1.3 Vue.js的Hello world

现在来看一下我们第一个Vue.js项目，按照传统，我们来写一个Hello World。

首先，引入Vue.js的方式有很多，你可以采用直接使用CDN，例如：

```
<script src='http://cdnjs.cloudflare.com/ajax/libs/vue/1.0.26/vue.min.js'></script>
```

也可以通过NPM进行安装：

```
npm install vue // 最新稳定版本
```

正确引入Vue.js之后，我们在HTML文件中的内容为：

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

```
  <h1>{{message}}</h1>
</div>
```

应用的 js 如下：

```
var vm = new Vue({
  el : '#app',
  data: {
    message : 'Hello world, I am Vue.js'
  }
});
```

输出结果为：

Hello world, I am Vue.js

这种形式类似于前端模板引擎，我们把 js 中 message 值替换了 HTML 模板中 {{message}} 这部分。

不过，如果仅仅是这样的例子，我相信你也不会有什么兴趣去使用 Vue.js。根据上文对 Vue.js 的说明，我们继续写两个有关于它特性的例子。

第一个特性是数据绑定，我们可以在运行上述例子的浏览器控制台（console）环境中输入 vm.message = 'Hello Vue.js'，输出结果就变为了 Hello Vue.js。也就说明 vm.message 和视图中的 {{message}} 是绑定的，我们无需手动去获取 <h1> 标签来修改里面的 innerHTML。

同样，我们也可以绑定用户输入的数据，视图会随着用户的输入而变化，例如：

```
<div id="app">
  <h1>Your input is {{ message }}</h1>
  <input type=" text" v-model=" message" >
</div>
```

Your input is Hello, World

Hello, World

vm.message 的值会随着用户在 input 中输入的值的变化而变化，而无需我们手动去获取 DOM 元素的值再同步到 js 中。

第二个特性是组件化，简单来说我们可以自己定义 HTML 标签，并在模板中使用它，例如：

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

```
  <message content='Hello World'></message>
</div>
<script type="text/javascript">
  var Message = Vue.extend({
    props : ['content'],
    template : '<h1>{{content}}</h1>'
  })
  Vue.component('message', Message);
  var vm = new Vue({
    el : '#app',
  });
</script>
```

我们在浏览器里最终看到的 HTML 结果为：

```
▼<div id="app">
    <h1>Hello World</h1>
  </div>
```

可以看到自定义的标签 <message> 被替换成了 <h1>Hello World</h1>，当然，实际中的组件化远比示例复杂，我们会给组件添加参数及方法，使之能更好地被复用。

如果说这几个例子引起了你对 Vue.js 的兴趣的话，那接下来我们就会详细地说明它的基础用法和应用场景，以及最终我们如何将它真实地运用到生产环境中。

第2章 基础特性

其实，无论前端框架如何变化，它需要处理的事情依旧是模板渲染、事件绑定、处理用户交互（输入信息或鼠标操作），只不过提供了不同的写法和理念。Vue.js 则会通过声明一个实例 new Vue({…}) 标记当前页面的 HTML 结构、数据的展示及相关事件的绑定。本章主要介绍 Vue.js 的构造函数的选项对象及用法，以及如何通过 Vue.js 来实现上述的常规前端功能。

2.1 实例及选项

从以前的例子可以看出，Vue.js 的使用都是通过构造函数 Vue({option}) 创建一个 Vue 的实例：var vm = new Vue({})。一个 Vue 实例相当于一个 MVVM 模式中的 ViewModel，如图 2-1 所示。

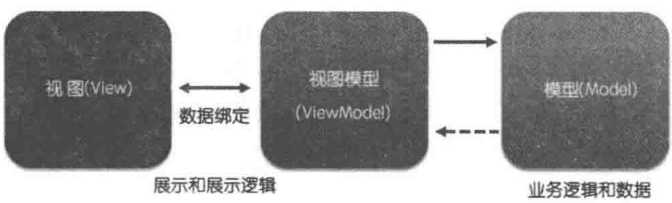


图2-1

在实例化的时候，我们可以传入一个选项对象，包含数据、模板、挂载元素、方法、生命周期钩子等选项。下面就对一些常用的选项对象属性进行具体的说明。

2.1.1 模板

选项中主要影响模板或 DOM 的选项有 el 和 template，属性 replace 和 template 需要一起使用。

el：类型为字符串，DOM 元素或函数。其作用是为实例提供挂载元素。一般来说我们会使用 css 选择符，或者原生的 DOM 元素。例如 el:'#app'。在初始项中指定了 el，实例将立即进入编译过程。

template：类型为字符串。默认会将 template 值替换挂载元素（即 el 值对应的元素），并合并挂载元素和模板根节点的属性（如果属性具有唯一性，类似 id，则以模板根节点为准）。如果 replace 为 false，模板 template 的值将插入挂载元素内。通过 template 插入模板的时候，挂载元素的内容都将被互联，除非使用 slot 进行分发（有关 slot 内容将在第 6 章组件中介绍）。在使用 template 时，我们往往不会把所有的 HTML 字符串直接写在 js 里面，这样影响可读性而且也不利于维护。所以经常用 '#tpl' 的方式赋值，并且在 body 内容添加 <scrip id="tpl" type="x-template"> 为标签包含的 HTML 内容，这样就能将 HTML 从 js 中分离开来，示例如下：

```
<div id="app">
  <p>123</p>
</div>
<script id="tpl" type="x-template">
  <div class='tpl'>
    <p>This is a tpl from script tag</p>
  </div>
</script>
<script type="text/javascript">
  var vm = new Vue({
    el : '#app',
    template : '#tpl'
  });
</script>
```

最终输出 HTML 结构为：

```
▼<div class="tpl" id="app">
    <p>This is a tpl from script tag</p>
  </div>
```

Vue.js 2.0 中废除了 replace 这个参数，并且强制要求每一个 Vue.js 实例需要有一个根元素，即不允许组件模板为：

```
<script id="tpl" type="x-template">
  <div class='tpl'>
    ...
  </div>
```

```
  <div class='tpl'>
    ...
  </div>
</script>
```

这样的兄弟节点为根节点的模板形式，需要改写成：

```
<script id="tpl" type="x-template">
  <div class='wrapper'>
    <div class='tpl'>
      ...
    </div>
    <div class='tpl'>
      ...
    </div>
  </div>
</script>
```

2.1.2 数据

Vue.js 实例中可以通过 data 属性定义数据，这些数据可以在实例对应的模板中进行绑定并使用。需要注意的是，如果传入 data 的是一个对象，Vue 实例会代理起 data 对象里的所有属性，而不会对传入的对象进行深拷贝。另外，我们也可以引用 Vue 实例 vm 中的 $data 来获取声明的数据，例如：

```
var data = { a: 1 }
var vm = new Vue({
  data: data
})
vm.$data === data // -> true
vm.a === data.a // -> true
// 设置属性也会影响到原始数据
vm.a = 2
data.a // -> 2
// 反之亦然
data.a = 3
vm.a // -> 3
```

然后在模板中使用 {{a}} 就会输出 vm.a 的值，并且修改 vm.a 的值，模板中的值会随之改变，我们也会称这个数据为响应式（responsive）数据（具体的用法和特性会在第 2.2 节的数据绑定中说明）。

需要注意的是，只有初始化时传入的对象才是响应式的，即在声明完实例后，再加上一句 vm.$data.b = '2'，并在模板中使用 {{b}}，这时是不会输出字符串 '2' 的。例如：

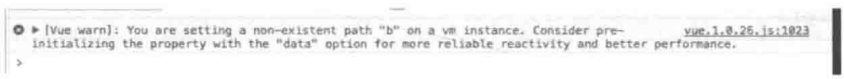
```
<div id="app">
```

```
  <p>{{a}}</p>
  <p>{{b}}</p>
</div>
var vm = new Vue({
   el : '#app',
   data : {
      a : 1
   }
});
vm.$data.b = 2;
```

如果需要在实例化之后加入响应式变量，需要调用实例方法 $set，例如：

```
vm.$set('b', 2);
```

不过 Vue.js 并不推荐这么做，这样会抛出一个异常：

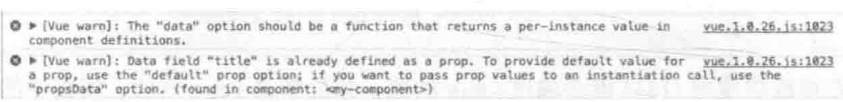


所以，我们应尽量在初始化的时候，把所有的变量都设定好，如果没有值，也可以用 undefined 或 null 占位。

另外，组件类型的实例可以通过 props 获取数据，同 data 一样，也需要在初始化时预设好。示例：

```
<my-component title='myTitle' content='myContent'></my-component>
var myComponent = Vue.component('my-component', {
  props : ['title', 'content'],
  template : '<h1>{{title}}</h1><p>{{content}}</p>'
})
```

我们也可以在上述组件类型实例中同时使用 data，但有两个地方需要注意：① data 的值必须是一个函数，并且返回值是原始对象。如果传给组件的 data 是一个原始对象的话，则在建立多个组件实例时它们就会共用这个 data 对象，修改其中一个组件实例的数据就会影响到其他组件实例的数据，这显然不是我们所期望的。② data 中的属性和 props 中的不能重名。这两者均会抛出异常：



所以正确的使用方法如下：

```
var MyComponent = Vue.component('my-component', {
  props : ['title', 'content'],
  data : function() {
    return {
```

```
        desc : '123'
      }
    },
    template : '<div> \
        <h1>{{title}}</h1> \
        <p>{{content}}</p> \
        <p>{{desc}}</p> \
      </div>'
})
```

2.1.3 方法

我们可以通过选项属性methods对象来定义方法，并且使用v-on指令来监听DOM事件，例如：

```
<button v-on:click="alert"/>alert<button>
new Vue({
  el : '#app',
  data : { a : 1},
  methods : {
    alert : function() {
       alert(this.a);
    }
  }
});
```

另外，Vue.js实例也支持自定义事件，可以在初始化时传入events对象，通过实例的$emit方法进行触发。这套通信机制常用在组件间相互通信的情况中，例如子组件冒泡触发父组件事件方法，或者父组件广播某个事件，子组件对其进行监听等。这里先简单说明下用法，详细的情况将会在第6章组件中进行说明。

```
var vm = new Vue({
  el : '#app',
  data : data,
  events : {
    'event.alert' : function() {
      alert('this is event alert :' + this.a);
    }
  }
});
vm.$emit('event.alert');
```

而Vue.js 2.0中废弃了events选项属性，不再支持事件广播这类特性，推荐直接使用Vue实例的全局方法$on()/$emit()，或者使用插件Vuex来处理。

2.1.4 生命周期

Vue.js 实例在创建时有一系列的初始化步骤，例如建立数据观察，编译模板，创建数据绑定等。在此过程中，我们可以通过一些定义好的生命周期钩子函数来运行业务逻辑。例如：

```
var vm = new Vue({
  data: {
    a: 1
  },
  created: function () {
    console.log('created')
  }
})
```

运行上述例子时，浏览器 console 中就会打印出 created。

下图是实例的生命周期，可以根据提供的生命周期钩子说明 Vue.js 实例各个阶段的情况，Vue.js 2.0 对不少钩子进行了修改，以下一并说明。

Vue.js 实例生命周期（原图出自于 Vue.js 官网），如图 2-2 所示。

init：在实例开始初始化时同步调用。此时数据观测、事件等都尚未初始化。2.0 中更名为 beforeCreate。

created：在实例创建之后调用。此时已完成数据绑定、事件方法，但尚未开始 DOM 编译，即未挂载到 document 中。

beforeCompile：在 DOM 编译前调用。2.0 废弃了该方法，推荐使用 created。

beforeMount：2.0 新增的生命周期钩子，在 mounted 之前运行。

compiled：在编译结束时调用。此时所有指令已生效，数据变化已能触发 DOM 更新，但不保证 $el 已插入文档。2.0 中更名为 mounted。

ready：在编译结束和 $el 第一次插入文档之后调用。2.0 废弃了该方法，推荐使用 mounted。这个变化其实已经改变了 ready 这个生命周期状态，相当于取消了在 $el 首次插入文档后的钩子函数。

attached：在 vm.$el 插入 DOM 时调用，ready 会在第一次 attached 后调用。操作 $el 必须使用指令或实例方法（例如 $appendTo()），直接操作 vm.$el 不会触发这个钩子。2.0 废弃了该方法，推荐在其他钩子中自定义方法检查是否已挂载。

detached：同 attached 类似，该钩子在 vm.$el 从 DOM 删除时调用，而且必须是指令或实例方法。2.0 中同样废弃了该方法。

beforeDestroy：在开始销毁实例时调用，此刻实例仍然有效。

destroyed：在实例被销毁之后调用。此时所有绑定和实例指令都已经解绑，子实例也被销毁。

beforeUpdate：2.0 新增的生命周期钩子，在实例挂载之后，再次更新实例（例如更新 data）时会调用该方法，此时尚未更新 DOM 结构。

updated：2.0 新增的生命周期钩子，在实例挂载之后，再次更新实例并更新完 DOM 结构后调用。

activated：2.0 新增的生命周期钩子，需要配合动态组件 keep-live 属性使用。在动态组件初始化渲染的过程中调用该方法。