第 1 章　例说 Vue.js

本章将通过极具代表性的 Todo 的示例作为引领读者进入 Vue.js 大门的引子。我会以实践为第一出发点，从零开始一步一步地构造一个单页式的 Todo 应用，在这个过程中会将 Vue.js 相关的知识点融入其中，在实际应用中展现这个“小”而“强”的界面框架。

我们先来看看最终希望构造出一个什么样的 App：

Vue.js 与 Angular2 和 React 相比，让我感觉最舒适的是它在一开始就为我们铺平了入门的道路，这就是它的脚手架 vue-cli。因为它的存在，省去了手工配置开发环境、运行环境和测试环境的步骤，开发者可以直接步入 Vue.js 开发的殿堂。然而，现在我并不打算详细地介绍这个脚手架工具，先让我们一起从使用体验来感性地认识它，在后面的章节中我会详细地介绍这个工具。

在开始动手之前，必须先得在机器上安装好 npm，然后输入以下指令将 vue-cli 安装到机器的全局环境中：

```
$ npm i vue-cli -g
```

然后，我们就可以开始建立工程了，键入以下的指令：

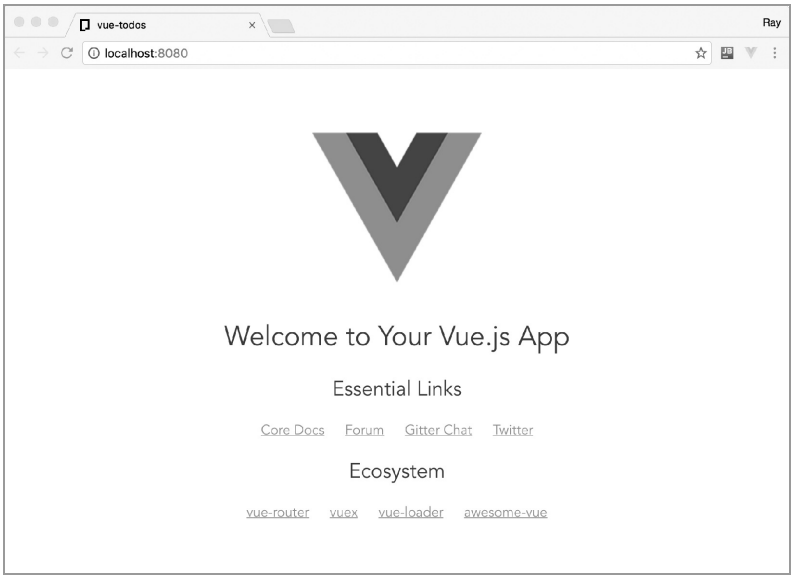
```
$ vue init webpack-simple vue-todos
```

此时控制台会提出一些关于这个新建项目的基本问题，直接“回车”跳过就行了。然后进入 vue-todo 目录，安装脚手架项目的基本支持包：

```
$ npm i
```

安装完支持包后键入以下指令就可以运行一个由脚手架构建的基本 Vue.js 程序了：

```
$ npm run dev
```



是不是很简单？进入代码中看看 vue-cli 到底为我们构造了一个什么样的代码结构：

```
├── README.md
├── index.html            # 默认启动页面
├── package.json          # npm 包配置文件
├── src
│   ├── App.vue           # 启动组件
│   ├── assets
│   │   └── logo.png
│   └── main.js           # Vue 实例启动入口
└── webpack.config.js     # webpack 配置文件
```

Vue2 与 Vue1.x 相比有了很大的区别，从最小化的运行程序开始了解 Vue 是一种绝佳的途径，先从 main.js 文件入手：

```
import Vue from 'vue'
import App from './App.vue'

new Vue({
  el: '#app',
  render: h => h(App)
})
```

这里就运用了 Vue2 新增的特色 Render 方法，如果你曾用过 React，是不是有一种似曾相识之感？确实，Vue2 甚至连渲染机制都与 React 一样了。为了得到更好的运行速度，Vue2 也采用了 Virtual DOM。如果你还没有接触过 Virtual DOM，并不要紧，现在只需要知道它是一种比浏览器原生的 DOM 具有更好性能的虚拟组件模型就行了，我们会在稍后的章节中再来讨论它。

我们需要知道的是，通过 import 将一个 Vue.js 的组件文件引入，并创建一个 Vue 对象的实例，在 Vue 实例中用 Render 方法来绘制这个 Vue 组件（App）就完成了初始化。

然后，将 Vue 实例绑定到一个页面上，真实存在的元素 App Vue 程序就引导成功了。

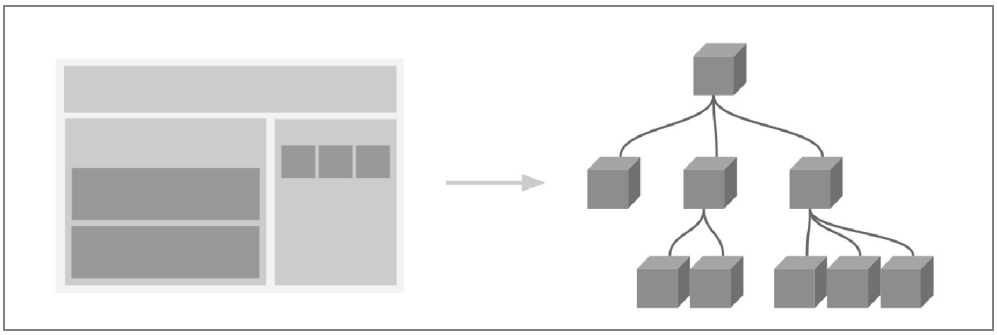
打开 index.html 文件就能看到 Vue 实例与页面的对应关系：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
</head>
<body>
    <!-- Vue 实例所对应的页面元素 -->
    <div id="app"></div>
    <!-- 由 Webpack 编译后的运行文件 -->
    <script src="/dist/build.js"></script>
</body>
</html>
```

也就是说，一个 Vue 实例必须与一个页面元素绑定。Vue 实例一般用作 Vue 的全局配置来使用，例如向实例安装路由、资源插件，配置应用于全局的自定义过滤器、自定义指令等。在本章示例中，我们只需要知道它的作用就可以了。

我们需要了解的是 App.vue 这个文件，*.vue 是 Vue.js 特有的文件格式，表示的就是一个 Vue 组件，它也是 Vue.js 的最大特色，被称为单页式组件。“*.vue”文件可以同时承载“视图模板”、“样式定义”和组件代码，它使得组件的文件组织更加清晰与统一。

Vue.js 的组件系统提供了一种抽象，让我们可以用独立可复用的小组件来构建大型应用。如果我们考虑到这一点，几乎任意类型应用的界面都可以抽象为一个组件树：



> Vue2 具有很高的兼容性，我们也可以用“.js”文件来单纯地定义组件的逻辑，甚至可以使用 React 的 JSX 格式的组件（需要 babel-plugin-transform-vue-jsx 支持）。

脚手架为我们创建的这个 App 组件内加入了不少介绍性的文字，将这个文件“净化”后就可以得到一个最简单的 Vue 组件定义模板：

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

<style></style>

<script>
export default {
  name: 'app'
}
</script>
```

由以上的代码我们可以了解到，单页组件由以下三个部分组成：

- <template>——视图模板；
- <style>——组件样式表；
- <script>——组件定义。

接下来我们就从这个示例开始，一步步学习 Vue 的基本组成部分，在实践中理解它们的作用。

1.1　插值

Vue 的视图模板是基于 DOM 实现的。这意味着所有的 Vue 模板都是可解析的有效的 HTML，而且它对一些特殊的特性做了增强。接下来，我们就在模板上定义一个网页标题，并通过数据绑定语法将 App 组件上定义的数据模型绑定到模板上。

首先，在组件脚本定义中使用 data 定义用于内部访问的数据模型：

```
export default {
  ...
  data () {
    return {
       title: "vue-todos"
    }
  }
}
```

data 可以是一个返回 Object 对象的函数，也可以是一个对象属性，也就是说，可以写成以下的方式：

```
export default {
  ...
  data : {
    title: "vue-todos"
  }
}
```

使用函数返回是为了可以具有更高的灵活性，例如对内部数据进行一些初始化的处理，官方推荐的用法是采用返回 Object 对象的函数。

在模板中引用 data.title 数据时我们并不需要写上 data，这只是 Vue 定义时的一个内部数据容器，通过 Vue 模块的插值方式直接写上 title 即可：

```
<h1>{{ title }}</h1>
```

用双大括号{{ }}引住的内容被称为“Mustache”语法，Mustache 标签会被相应数据对象的 title 属性的值替换。每当这个属性变化时它也会更新。

插值是 Vue 模板语言的最基础用法，很多的变量输出都会采用插值的方式，而且插值还可以支持 JavaScript 表达式运算和过滤器（下文将会提及）。{{}}引用的内容都会被编码，如果要输出未被编码的文本，可以使用{{{}}}对变量进行引用。

完整代码如下所示。

```
<template>
  <div id="app">
    <h1>{{ title }}</h1>
  </div>
</template>

<style></style>

<script>
export default {
  name: 'app',
  data () {
      return {
          title: "vue-todos"
      }
  }
}
</script>
```

> 从 Vue2 开始，组件模板必须且只能有一个顶层元素，如果在组件模块内设置多个顶层元素将会引发编译异常。

请注意，在上述代码中 template 属性是 V，也就是视图，title 属性是 M，也就是模型，这个概念是必须要了解的。

1.2 数据绑定

我们需要一个稍微复杂一点的数据模型来表述 Todo，它的结构应该是这样的：

```
{
    value: '事项 1', // 待办事项的文字内容
    done: false    // 标记该事项是否已完成
}
```

由于是多个事项，那么这个数据模型应该是一个数组，为了能先显示这些待办事项，我们需要先设定一些样本数据。在 Vue 实例定义中的 data 属性中加入以下代码：

```
export default {
    data () {
        return {
            title: 'vue-todos',
            todos: [
```

```
                { value: "阅读一本关于前端开发的书", done: false },
                { value: "补充范例代码", done: true },
                { value: "写心得", done: false }
            ]
        }
    }
}
```

初学者可能会问 data 有什么作用？我们可以将 Vue 实例定义看作一个类的定义，data 相当于这个类的内部字段属性的定义区域。在 Vue 实例内的其他地方可以直接用 this 引用 data 内定义的任何属性，比如 this.title 就是引用了 data.title。

我们要显示 todos 的数据就需要使用 Vue 模板的一个最常用的 v-for 指令标记，它可以用于枚举一个数组并将对象渲染成一个列表。这个指令使用与 JS 类似的语法对 items 进行枚举，形式为 item in items，items 是数据数组，item 是当前数组元素的别名：

```
<ul>
    <li v-for="todo in todos">
        <label>{{ todo.value }}</label>
    </li>
</ul>
```

它的输出结果如下所示。

```
<ul>
  <li>
    <label>阅读一本关于前端开发的书</label>
  </li>
  <li>
    <label>补充范例代码</label>
  </li>
  <li>
    <label>写心得</label>
  </li>
</ul>
```

如果我们要输出待办事项的序号，可以用 v-for 中隐藏的一个 index 值来进行输出，具体用法如下：

```
<ul>
    <li v-for="(todo,index) in todos"
        :id="index">
        <label>{{ index + 1 }}.{{ todo.value }}</label>
    </li>
</ul>
```

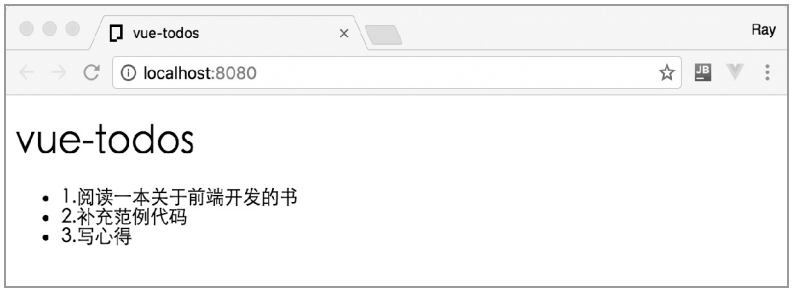
这个用法有点像 Python 的元组引用方式，只要用括号括住引用参数，最后一个值就是循环的索引。索引是由 0 开始计数的，而我们要输出的序号应该从 1 开始，正好我们使用了一个 JavaScript 的表达式插值来输出一个 index + 1 的从 1 开始计数的序号。

这里除了用插值绑定，还使用了属性绑定语法，就是上面的 id="index"，这样的写法是一种缩写，下文中会有解释，意思是将 index 的值输出到 DOM 的 id 属性上。如果 index=1，那么输出结果就是 id="1"，如果没有在 id 前面加上“:”，那么 Vue 就会认为我们正在为 id 属性赋予一个字符串。

完成这一步，我们打开终端输入：

```
$ npm run dev
```

npm 将自动打开流浏览器并显示以下的结果：



v-for 不单单可以循环渲染数组，还可以渲染对象属性，例如：

```
<ul>
  <li v-for="value in object">
    {{ value }}
  </li>
</ul>
data () {
    return {
        object {
            first_name : "Ray",
            last_name : "Liang"
        }
    }
}
```

输出

- "Ray"
- "Liang"

小结

对于从来没有接触过 Angular 和 Vue 的初学者，可能对上述的代码感到疑惑，为什么我们的代码内没有任何一个地方操作 DOM 并且将 data 内的变量设置到 DOM 上面呢？

首先，在 Vue 的代码中直接操作 DOM 是不被推荐的，如果你之前是 jQuery 的开发者，这一点一定要牢记；其次，DOM 是被 Vue 直接托管的，所有“绑定”到 DOM 上的变量一旦发生变化，DOM 所对应的属性就会被 Vue 自动重绘而不需要像 jQuery 那样通过编码来显式地操作，这才是绑定的意义所在。

1.3　样式绑定

没有样式的输出结果样子很丑，此时我们就需要用 CSS 来美化我们的 App。我个人并不推荐直接使用 CSS 语法来编写样式表，因为纯 CSS 的代码量很大，而且需要不断地重复，我很讨厌重复而且对 DRY（Don't Repeat Yourself）有一种偏执。由于 CSS 总是充满各种不得不重复的写法，所以我更愿意使用 less，以下是安装 webpack 支持 less 编译的包的方法：

```
$ npm i less style-loader css-loader less-loader -D
```

安装完成后在 webpack.config.js 的 modules 设置内加入以下的配置：

```
module : {
    rules: [
    // ... 省略
    {
        test: /\.less$/,
        loader: "style!css!less"
    }
  ]
}
```

在/assets/中添加一个 todos.less 文件，并在 App.vue 的组件定义内引入 less 样式表：

```
import './assets/todos.less'

export default {
    // ... 省略
}
```

使用 import 将样式表直接导入到代码的效果是：webpack 的 less-loader 会生成一些代码，在页面运行的时候将编译后的 less 代码生成到<style>标签内并自动插入到页面的

<head>中。有一点要注意的是，这种做法是全局的，在后面介绍路由部分时会有多个组件页面加载到同一个页上，如果使用 import 导入样式的话，样式就会长期驻留页面直至 Vue 的根（root）实例被销毁。

关于这个 less 样式表的定义属于 HTML 的基础，由于篇幅问题就不在此罗列出来了，读者可以到本书的 github 地址 http://www.github.com/dotnetage/vue-in-action 上下载。

运行效果如下：

现在终于舒服多了。这里所有的待办事项都没有显示任何的状态，此时就需要使用 Vue 的**样式绑定**功能了。

> 通过 import 将样式文件导入是一种全局性的做法，也就是说，在每一个页面内的<head>中都会有这一个样式表，这样做的缺点是很容易导致样式冲突。如果希望样式表仅应用于当前组件，可以使用<style scoped>，然后用 CSS 的@import 导入样式表：

```
<style scoped>
 @import './assets/todos.less'
</style>
```

前文我们只提到如何将 data 内定义的值以文本插值的方式输出到页面，并没有介绍如何将值“绑定”到属性内。样式的绑定和属性的绑定方式是一样的，我们这里就将 done==true 的待办事项<li>绑定一个 checked 的样式类：

```
<li v-for="(todo,index) in todos"
    :class="{'checked': todo.done}" >
  <!-- 省略 ... -->
</li>
```

Vue 的属性绑定语法是通过 v-bind 实现的，完整的写法是这样：

```
<li v-for="(todo,index) in todos"
   v-bind:class="{'checked': todo.done}">
```

但 v-bind 可以采用缩写方式“:”表示，采用完整写法又将出现各种重复，所以建议还是直接使用缩写方式，这样会更直观。

由此可见，Vue 的属性绑定语法是 attribute="expression"，attribute 就是元素接收的属性值（既可以是原生的也可以是自定义的），expression 则是在 Vue 组件内由 data 或 props 内定义的对象属性，又或是一个合法的表达式。

> **要谨记一点**：如果在元素属性中不加上“:”，Vue 认为是向这个属性赋上字符串值而不是 Vue 组件上定义的属性引用！

上例中:class="{'checked': todo.done}"的意思是：当 todo.done 为 true 时，向<li>元素的 class 添加 checked 样式类。这是 Vue 样式绑定与普通属性绑定最大的不同点，凡是样式绑定必然是绑定到判断对象上的，不能直接写 CSS 类名，即使要绑定一个固定的 CSS 类也都要这样写，即:class="{'btn':true}"，除非不使用样式绑定。

以下是应用样式绑定后的输出效果：

小结

这里推荐一个简单的记忆方法来学习 Vue 的样式绑定，无论绑定的是样式类还是样式属性，:class 和:style 表达式内一定是一个 JSON 对象。

- :class 的 JSON 对象的值一定是布尔型的，true 表示加上样式，false 表示移除样式类。
- :style 的 JSON 对象则像是一个样式配置项，key 声明属性名，value 则是样式属性的具体值。

1.4 过滤器

我们在待办事项的右侧增加一个时间字段 created，并用<time>元素表示，修改后完整的代码如下所示。

```
<template>
  <div id="app">
    <h1>{{ title }}</h1>
    <ul class="todos">
      <li v-for="(todo,index) in todos"
          :class="{'checked': todo.done}">
        <label>{{ index + 1 }}.{{ todo.value }}</label>
        <time>{{ todo.created }}</time>
      </li>
    </ul>
  </div>
</template>

<script>
import './assets/todos.less'

export default {
  name: 'app',
  data () {
    return {
       title: 'vue-todos',
       todos: [
         {
           value: "阅读一本关于前端开发的书",
           done: false ,
           created : Date.now()
         },
         {
           value: "补充范例代码",
           done: true ,
           created: Date.now() + 300000
         },
```

```
            {
              value: "写心得",
              done: false ,
              created: Date.now() - 30000000
            }
          ]
      }
    }
}
</script>
```

查看输出结果：

很明显，时间的输出并不是我们想要的结果，这里输出的是一个整数，因为将 Date 对象直接输出的话，JavaScript 引擎会将其时间戳作为值输出，所以我们需要对这个时间戳来一个漂亮的格式化。

此时我们可以用一个很出名的时间格式化专用的包——moment.js，先安装 moment.js：

```
$ npm i moment -S
```

Vue.js 用“过滤器”进行模板格式化，过滤器实质上是一个只带单一输入参数的函数，在 Vue2 中已经将原有的内置过滤器移除了，甚至将一些相关的特色功能也移除了，例如双向过滤器。官方的说法是“计算方法”会比使用“过滤器”更明确，代码更容易读。我认为这有点矫枉过正，过滤器并不是 Vue 和 Angular 这类前端框架所独有的，在很多的服务端视图框架中也是一种很常见的用法。过滤器有用的地方是可以以管道方式进行传递调用。在此，对日期的格式化我还是倾向于使用过滤器的方式，在 Vue 组件中加入自定义过滤器非常简单，只要在 filters 属性内加入方法定义就可以在模块上使用了。

首先，我们要引入 moment，并设定 moment 的区域为中国：

```
import moment from 'moment'
import 'moment/locale/zh-cn'
moment.locale('zh-cn')
```

然后加入一个 date 的过滤器：

```
export default {
    // 省略 ...
    filters: {
        date(val) {
            return moment(val).calendar()
        }
    }
}
```

最后在模板上应用这个过滤器：

```
<time>{{ todo.created | date }}</time>
```

我们可以看到浏览器的显示结果将变为下图的方式：

> 在所有的过滤器中是没有 this 引用的，过滤器内的 this 是一个 undefined 的值，所以不要在过滤器内尝试引用组件实例内的变量或方法，否则会引发空值引用的异常。