介绍

Vue.js 是什么 [#Vue-js-是什么]

观看本节视频讲解

Vue (读音 /vjuː/，类似于 view) 是一套用于构建用户界面的**渐进式框架**。与其它大型框架不同的是，Vue 被设计为可以自底向上逐层应用。Vue 的核心库只关注视图层，不仅易于上手，还便于与第三方库或既有项目整合。另一方面，当与**现代化的工具链** [single-file-components.html] 以及各种**支持类库** [https://github.com/vuejs/awesome-vue#libraries--plugins] 结合使用时，Vue 也完全能够为复杂的单页应用提供驱动。

如果你想在深入学习 Vue 之前对它有更多了解，我们**制作了一个视频** [#] ，带您了解其核心概念和一个示例工程。

如果你已经是有经验的前端开发者，想知道 Vue 与其它库/框架有哪些区别，请查看**对比其它框架** [comparison.html] 。

起步 [#起步]

观看本节视频讲解

> ! 官方指南假设你已了解关于 HTML、CSS 和 JavaScript 的中级知识。如果你刚开始学习前端开发，将框架作为你的第一步可能不是最好的主意——掌握好基础知识再来吧！之前有其它框架的使用经验会有帮助，但这不是必需的。

安装 [installation.html]

尝试 Vue.js 最简单的方法是使用 **JSFiddle 上的 Hello World 例子** [https://jsfiddle.net/chrisvfritz/50wL7mdz/] 。你可以在浏览器新标签页中打开它，跟着例子学习

一些基础用法。或者你也可以**创建一个 `.html` 文件 [https://gist.githubusercontent.com/chrisvfritz/7f8d7d63000b48493c336e48b3db3e52/raw/ed60c4e5d5c6f**，然后通过如下方式引入 Vue：

HTML

```
<!-- 开发环境版本，包含了有帮助的命令行警告 -->
<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
```

或者：

HTML

```
<!-- 生产环境版本，优化了尺寸和速度 -->
<script src="https://cdn.jsdelivr.net/npm/vue"></script>
```

安装教程 [/guide/installation.html] 给出了更多安装 Vue 的方式。请注意我们**不推荐**新手直接使用 `vue-cli`，尤其是在你还不熟悉基于 Node.js 的构建工具时。

如果你喜欢交互式的东西，你也可以查阅**这个 Scrimba 上的系列教程 [https://scrimba.com/playlist/pXKqta]**，它揉合了录屏和代码试验田，并允许你随时暂停和播放。

声明式渲染 [#声明式渲染]

观看本节视频讲解

Vue.js 的核心是一个允许采用简洁的模板语法来声明式地将数据渲染进 DOM 的系统：

HTML

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

JS

```
var app = new Vue({
  el: '#app',
  data: {
    message: 'Hello Vue!'
  }
})
```

Hello Vue!

我们已经成功创建了第一个 Vue 应用！看起来这跟渲染一个字符串模板非常类似，但是 Vue 在背后做了大量工作。现在数据和 DOM 已经被建立了关联，所有东西都是**响应式的**。我们要怎么确

认呢？打开你的浏览器的 JavaScript 控制台 (就在这个页面打开)，并修改 `app.message` 的值，你将看到上例相应地更新。

除了文本插值，我们还可以像这样来绑定元素特性：

HTML

```
<div id="app-2">
  <span v-bind:title="message">
    鼠标悬停几秒钟查看此处动态绑定的提示信息！
  </span>
</div>
```

JS

```
var app2 = new Vue({
  el: '#app-2',
  data: {
    message: '页面加载于 ' + new Date().toLocaleString()
  }
})
```

鼠标悬停几秒钟查看此处动态绑定的提示信息！

这里我们遇到了一点新东西。你看到的 `v-bind` 特性被称为**指令**。指令带有前缀 `v-` ，以表示它们是 Vue 提供的特殊特性。可能你已经猜到了，它们会在渲染的 DOM 上应用特殊的响应式行为。在这里，该指令的意思是：“将这个元素节点的 `title` 特性和 Vue 实例的 `message` 属性保持一致”。

如果你再次打开浏览器的 JavaScript 控制台，输入 `app2.message = '新消息'` ，就会再一次看到这个绑定了 `title` 特性的 HTML 已经进行了更新。

条件与循环 [#条件与循环]

观看本节视频讲解

控制切换一个元素是否显示也相当简单：

HTML

```
<div id="app-3">
  <p v-if="seen">现在你看到我了</p>
</div>
```

JS

```
var app3 = new Vue({
  el: '#app-3',
  data: {
    seen: true
  }
})
```

现在你看到我了

继续在控制台输入 `app3.seen = false` ，你会发现之前显示的消息消失了。

这个例子演示了我们不仅可以把数据绑定到 DOM 文本或特性，还可以绑定到 DOM **结构**。此外，Vue 也提供一个强大的过渡效果系统，可以在 Vue 插入/更新/移除元素时自动应用**过渡效果 [transitions.html]** 。

还有其它很多指令，每个都有特殊的功能。例如， `v-for` 指令可以绑定数组的数据来渲染一个项目列表：

HTML

```
<div id="app-4">
  <ol>
    <li v-for="todo in todos">
      {{ todo.text }}
    </li>
  </ol>
</div>
```

JS

```
var app4 = new Vue({
  el: '#app-4',
  data: {
    todos: [
      { text: '学习 JavaScript' },
      { text: '学习 Vue' },
      { text: '整个牛项目' }
    ]
  }
})
```

1. 学习 JavaScript
2. 学习 Vue
3. 整个牛项目

在控制台里，输入 `app4.todos.push({ text: '新项目' })`，你会发现列表最后添加了一个新项目。

处理用户输入 [#处理用户输入]

> 观看本节视频讲解

为了让用户和你的应用进行交互，我们可以用 `v-on` 指令添加一个事件监听器，通过它调用在 Vue 实例中定义的方法：

HTML

```
<div id="app-5">
  <p>{{ message }}</p>
  <button v-on:click="reverseMessage">反转消息</button>
</div>
```

JS

```
var app5 = new Vue({
  el: '#app-5',
  data: {
    message: 'Hello Vue.js!'
  },
  methods: {
    reverseMessage: function () {
      this.message = this.message.split('').reverse().join('')
    }
  }
})
```

Hello Vue.js!

逆转消息

注意在 `reverseMessage` 方法中，我们更新了应用的状态，但没有触碰 DOM——所有的 DOM 操作都由 Vue 来处理，你编写的代码只需要关注逻辑层面即可。

Vue 还提供了 `v-model` 指令，它能轻松实现表单输入和应用状态之间的双向绑定。

HTML

```
<div id="app-6">
  <p>{{ message }}</p>
  <input v-model="message">
</div>
```

JS

```
var app6 = new Vue({
  el: '#app-6',
  data: {
    message: 'Hello Vue!'
  }
})
```

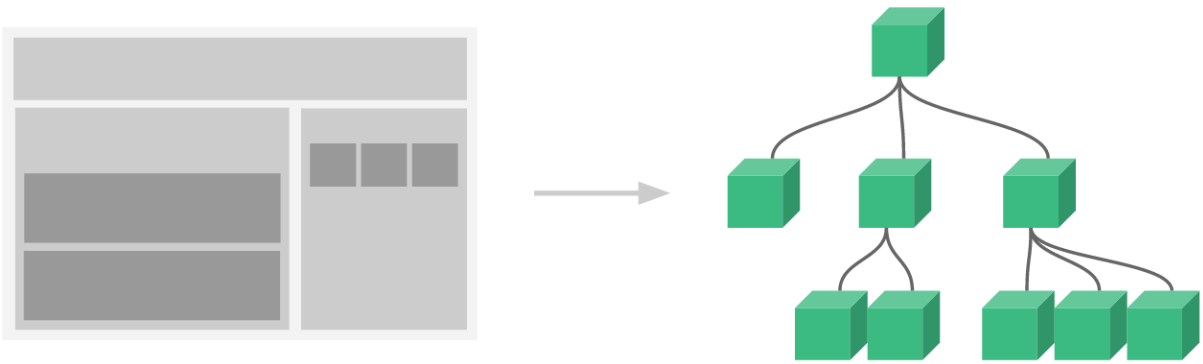
Hello Vue!

Hello Vue!

组件化应用构建 [#组件化应用构建]

观看本节视频讲解

组件系统是 Vue 的另一个重要概念，因为它是一种抽象，允许我们使用小型、独立和通常可复用的组件构建大型应用。仔细想想，几乎任意类型的应用界面都可以抽象为一个组件树：



在 Vue 里，一个组件本质上是一个拥有预定义选项的一个 Vue 实例。在 Vue 中注册组件很简单：

JS

```
// 定义名为 todo-item 的新组件
Vue.component('todo-item', {
  template: '<li>这是个待办项</li>'
})
```

现在你可以用它构建另一个组件模板：

HTML

```
<ol>
  <!-- 创建一个 todo-item 组件的实例 -->
```

```
  <todo-item></todo-item>
</ol>
```

但是这样会为每个待办项渲染同样的文本，这看起来并不炫酷。我们应该能从父作用域将数据传到子组件才对。让我们来修改一下组件的定义，使之能够接受一个 **prop [components.html#通过-Prop-向子组件传递数据]**：

JS

```
Vue.component('todo-item', {
  // todo-item 组件现在接受一个
  // "prop"，类似于一个自定义特性。
  // 这个 prop 名为 todo。
  props: ['todo'],
  template: '<li>{{ todo.text }}</li>'
})
```

现在，我们可以使用 `v-bind` 指令将待办项传到循环输出的每个组件中：

HTML

```
<div id="app-7">
  <ol>
    <!--
      现在我们为每个 todo-item 提供 todo 对象
      todo 对象是变量，即其内容可以是动态的。
      我们也需要为每个组件提供一个“key”，稍后再
      作详细解释。
    -->
    <todo-item
      v-for="item in groceryList"
      v-bind:todo="item"
      v-bind:key="item.id"
    ></todo-item>
  </ol>
</div>
```

JS

```
Vue.component('todo-item', {
  props: ['todo'],
  template: '<li>{{ todo.text }}</li>'
})

var app7 = new Vue({
  el: '#app-7',
  data: {
    groceryList: [
      { id: 0, text: '蔬菜' },
```

```
      { id: 1, text: '奶酪' },
      { id: 2, text: '随便其它什么人吃的东西' }
    ]
  }
})
```

1. 蔬菜
2. 奶酪
3. 随便其它什么人吃的东西

尽管这只是一个刻意设计的例子，但是我们已经设法将应用分割成了两个更小的单元。子单元通过 prop 接口与父单元进行了良好的解耦。我们现在可以进一步改进 `<todo-item>` 组件，提供更为复杂的模板和逻辑，而不会影响到父单元。

在一个大型应用中，有必要将整个应用程序划分为组件，以使开发更易管理。在**后续教程 [components.html]** 中我们将详述组件，不过这里有一个 (假想的) 例子，以展示使用了组件的应用模板是什么样的：

HTML

```
<div id="app">
  <app-nav></app-nav>
  <app-view>
    <app-sidebar></app-sidebar>
    <app-content></app-content>
  </app-view>
</div>
```

与自定义元素的关系 [#与自定义元素的关系]

你可能已经注意到 Vue 组件非常类似于**自定义元素**——它是 **Web 组件规范 [https://www.w3.org/wiki/WebComponents/]** 的一部分，这是因为 Vue 的组件语法部分参考了该规范。例如 Vue 组件实现了 **Slot API [https://github.com/w3c/webcomponents/blob/gh-pages/proposals/Slots-Proposal.md]** 与 `is` 特性。但是，还是有几个关键差别：

1. Web Components 规范已经完成并通过，但未被所有浏览器原生实现。目前 Safari 10.1+、Chrome 54+ 和 Firefox 63+ 原生支持 Web Components。相比之下，Vue 组件不需要任何 polyfill，并且在所有支持的浏览器 (IE9 及更高版本) 之下表现一致。必要时，Vue 组件也可以包装于原生自定义元素之内。

2. Vue 组件提供了纯自定义元素所不具备的一些重要功能，最突出的是跨组件数据流、自定义事件通信以及构建工具集成。

虽然 Vue 内部没有使用自定义元素，不过在应用使用自定义元素、或以自定义元素形式发布时，**依然有很好的互操作性 [https://custom-elements-everywhere.com/#vue]**。Vue CLI 也支持将 Vue 组件构建成为原生的自定义元素。

准备好了吗？ [#准备好了吗？]

我们刚才简单介绍了 Vue 核心最基本的功能——本教程的其余部分将更加详细地涵盖这些功能以及其它高级功能，所以请务必读完整个教程！

← 安装 [/v2/guide/installation.html]

Vue 实例 [/v2/guide/instance.html] →