概述

Vue.js（读音 /vju:/, 类似于 **view**）是一个构建数据驱动的 web 界面的库。Vue.js 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.js 自身不是一个全能框架——它只聚焦于视图层。因此它非常容易学习，非常容易与其它库或已有项目整合。另一方面，在与相关工具和支持库一起使用时，Vue.js 也能完美地驱动复杂的单页应用。

如果你是有经验的前端开发者，想知道 Vue.js 与其它库/框架的区别，查看**对比其它框架**；如果你对使用 Vue.js 开发大型应用更感兴趣，查看**构建大型应用**。

响应的数据绑定

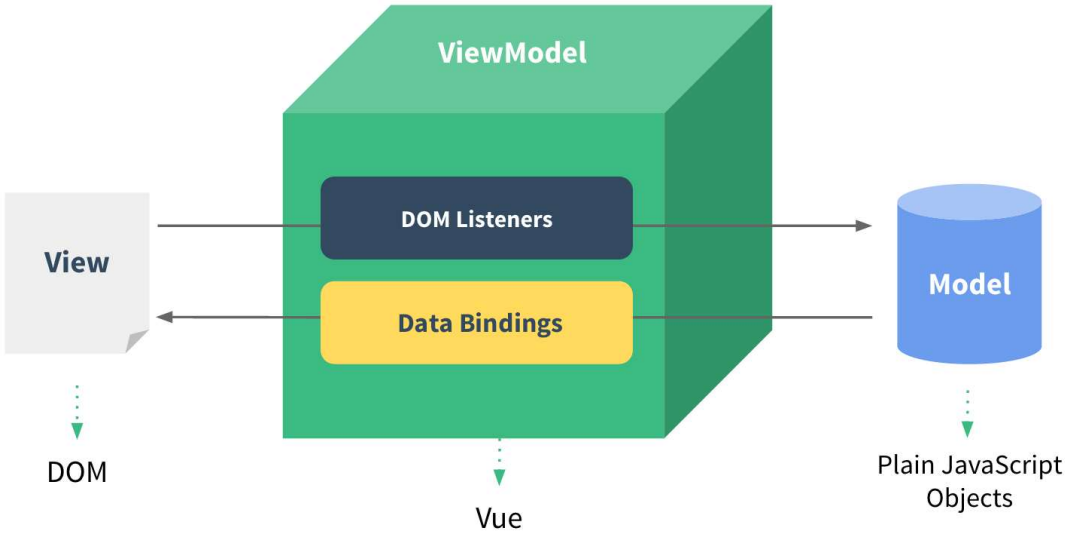
Vue.js 的核心是一个响应的数据绑定系统，它让数据与 DOM 保持同步非常简单。在使用 jQuery 手工操作 DOM 时，我们的代码常常是命令式的、重复的与易错的。Vue.js 拥抱**数据驱动的视图**概念。通俗地讲，它意味着我们在普通 HTML 模板中使用特殊的语法将 DOM “绑定”到底层数据。一旦创建了绑定，DOM 将与数据保持同步。每当修改了数据，DOM 便相应地更新。这样我们应用中的逻辑就几乎都是直接修改数据了，不必与 DOM 更新搅在一起。这让我们的代码更容易撰写、理解与维护。

插件

构建大型应用

对比其它框架

加入 Vue 社区！



可能是最简单的例子：

HTML

```
<!-- 这是我们的 View -->
<div id="example-1">
  Hello {{ name }}!
</div>
```

JS

```
// 这是我们的 Model
var exampleData = {
  name: 'Vue.js'
}
// 创建一个 Vue 实例或 "ViewModel"
// 它连接 View 与 Model
var exampleVM = new Vue({
  el: '#example-1',
  data: exampleData
})
```

结果：

Hello Vue.js!

看起来这跟单单渲染一个模板非常类似，但是 Vue.js 在背后做了大量工作。并且 DOM 会自动响应数据的变化。我们如何知道？打开你的浏览器的控制台，修改 `exampleData.name` ，你将看到上例相应地更新。

注意我们不需要撰写任何 DOM 操作代码：被绑定增强的 HTML 模板是底层数据状态的声明式的映射，数据不过是普通 JavaScript 对象。我们的视图完全由数据驱动。

让我们来看第二个例子：

HTML

```
<div id="example-2">
  <p v-if="greeting">Hello!</p>
</div>
```

JS

```
var exampleVM2 = new Vue({
  el: '#example-2',
  data: {
    greeting: true
  }
})
```

Hello!

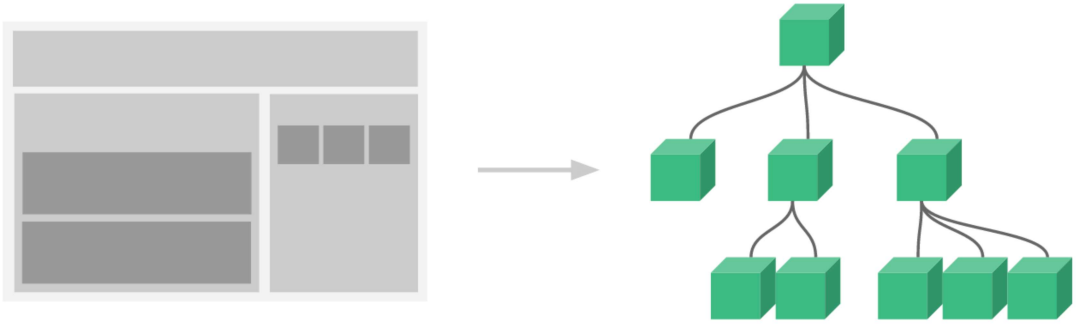
这里我们遇到新东西。你看到的 `v-if` 特性被称为**指令**。指令带有前缀 `v-` ，以指示它们是 Vue.js 提供的特殊特性。并且如你所想象的，它们会对绑定的目标元素添加响应式的特殊行为。继续在控制台设置 `exampleVM2.greeting` 为 `false` ，你会发现 “Hello!” 消失了。

第二个例子演示了我们不仅可以绑定 DOM 文本到数据，也可以绑定 DOM **结构** 到数据。而且，Vue.js 也提供一个强大的过渡效果系统，可以在 Vue 插入/删除元素时自动应用过渡效果。

也有一些其它指令，每个都有特殊的功能。例如 `v-for` 指令用于显示数组元素， `v-bind` 指令用于绑定 HTML 特性。我们将在后面详细讨论全部的数据绑定语法。

组件系统

组件系统是 Vue.js 另一个重要概念，因为它提供了一种抽象，让我们可以用独立可复用的小组件来构建大型应用。如果我们考虑到这点，几乎任意类型的应用的界面都可以抽象为一个组件树：



实际上，一个典型的用 Vue.js 构建的大型应用将形成一个组件树。在后面的教程中我们将详述组件，不过这里有一个假想的例子，看看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>
```

你可能已经注意到 Vue.js 组件非常类似于**自定义元素**——它是 **Web 组件规范**的一部分。实际上 Vue.js 的组件语法参考了该规范。例如 Vue 组件实现了 **Slot API** 与 `is` 特性。但是，有几个关键的不同：

1. Web 组件规范仍然远未完成，并且没有浏览器实现。相比之下，Vue.js 组件不需要任何补丁，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.js 组件也可以放在原生自定义元素之内。

2. Vue.js 组件提供了原生自定义元素所不具备的一些重要功能，比如组件间的数据流，自定义事件系统，以及动态的、带特效的组件替换。

组件系统是用 Vue.js 构建大型应用的基础。另外，Vue.js 生态系统也提供了高级工具与多种支持库，它们和 Vue.js 一起构成了一个更加“框架”性的系统。

← **起步**

Vue 实例 →