Table of Contents

Introduction

基础

兼容性

Vue 不支持 IE8 及以下版本，因为 Vue 使用了 IE8 无法模拟的 ECMAScript 5 特性。但它支持所有兼容 ECMAScript 5 的浏览器。

更新日志

最新稳定版本：

每个版本的更新日志见 GitHub。

Vue Devtools

在使用 Vue 时，我们推荐在你的浏览器上安装 Vue Devtools。它允许你在一个更友好的界面中审查和调试 Vue 应用。

直接用 `<script>` 引入

直接下载并用 `<script>` 标签引入，`Vue` 会被注册为一个全局变量。

在开发环境下不要使用压缩版本，不然你就失去了所有常见错误相关的警告!

开发版本包含完整的警告和调试模式 生产版本删除了警告，KB min+gzip

CDN

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@2.5.13/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 unpkg 和 cdnjs 上获取 (cdnjs 的版本更新可能略滞后)。

请确认了解不同构建版本并在你发布的站点中使用生产环境版本，把 `vue.js` 换成 `vue.min.js` 。这是一个更小的构建，可以带来比开发环境下更快的速度体验。

NPM

在用 Vue 构建大型应用时推荐使用 NPM 安装[1]。NPM 能很好地和诸如 webpack 或 Browserify 模块打包器配合使用。同时 Vue 也提供配套工具来开发单文件组件。

```
# 最新稳定版
$ npm install vue
```

命令行工具 (CLI)

Vue 提供了一个官方的 CLI，为单页面应用快速搭建 (SPA) 繁杂的脚手架。它为现代前端工作流提供了 batteries-included 的构建设置。只需要几分钟的时间就可以运行起来并带有热重载、保存时 lint 校验，以及生产环境可用的构建版本。更多详情可查阅 Vue CLI 的文档

CLI 工具假定用户对 Node.js 和相关构建工具有一定程度的了解。如果你是新手，我们强烈建议先在不用构建工具的情况下通读指南，在熟悉 Vue 本身之后再使用 CLI。

对不同构建版本的解释

在 NPM 包的 `dist/` 目录你将会找到很多不同的 Vue.js 构建版本。这里列出了它们之间的差别：

	UMD	CommonJS	ES Module
完整版	vue.js	vue.common.js	vue.esm.js
只包含运行时版	vue.runtime.js	vue.runtime.common.js	vue.runtime.esm.js
完整版 (生产环境)	vue.min.js	-	-
只包含运行时版 (生产环境)	vue.runtime.min.js	-	-

术语

- 完整版：同时包含编译器和运行时的版本。
- 编译器：用来将模板字符串编译成为 JavaScript 渲染函数的代码。
- 运行时：用来创建 Vue 实例、渲染并处理虚拟 DOM 等的代码。基本上就是除去编译器的其它一切。
- **UMD**：UMD 版本可以通过 `<script>` 标签直接用在浏览器中。jsDelivr CDN 的 https://cdn.jsdelivr.net/npm/vue 默认文件就是运行时 + 编译器的 UMD 版本 (`vue.js`)。
- **CommonJS**：CommonJS 版本用来配合老的打包工具比如 Browserify 或 webpack 1。这些打包工具的默认文件 (`pkg.main`) 是只包含运行时的 CommonJS 版本 (`vue.runtime.common.js`)。
- **ES Module**：ES module 版本用来配合现代打包工具比如 webpack 2 或 Rollup。这些打包工具的默认文件 (`pkg.module`) 是只包含运行时的 ES Module 版本 (`vue.runtime.esm.js`)。

运行时 + 编译器 vs. 只包含运行时

如果你需要在客户端编译模板 (比如传入一个字符串给 `template` 选项，或挂载到一个元素上并以其 DOM 内部的 HTML 作为模板)，就将需要加上编译器，即完整版：

```
// 需要编译器
new Vue({
  template: '<div>{{ hi }}</div>'
})

// 不需要编译器
new Vue({
  render (h) {
    return h('div', this.hi)
  }
})
```

当使用 `vue-loader` 或 `vueify` 的时候，`*.vue` 文件内部的模板会在构建时预编译成 JavaScript。你在最终打好的包里实际上是不需要编译器的，所以只用运行时版本即可。

因为运行时版本相比完整版体积要小大约 30%，所以应该尽可能使用这个版本。如果你仍然希望使用完整版，则需要在打包工具里配置一个别名：

webpack

```
module.exports = {
  // ...
  resolve: {
    alias: {
      'vue$': 'vue/dist/vue.esm.js' // 用 webpack 1 时需用 'vue/dist/vue.common.js'
    }
  }
}
```

Rollup

```
const alias = require('rollup-plugin-alias')

rollup({
  // ...
  plugins: [
    alias({
      'vue': 'vue/dist/vue.esm.js'
    })
  ]
})
```

Browserify

添加到你项目的 `package.json`：

```
{
  // ...
  "browser": {
    "vue": "vue/dist/vue.common.js"
  }
}
```

Parcel

在你项目的 `package.json` 中添加：

```
{
  // ...
  "alias": {
    "vue" : "./node_modules/vue/dist/vue.common.js"
  }
}
```

开发环境 **vs.** 生产环境模式

对于 UMD 版本来说，开发环境/生产环境模式是硬编码好的：开发环境下用未压缩的代码，生产环境下使用压缩后的代码。

CommonJS 和 ES Module 版本是用于打包工具的，因此我们不提供压缩后的版本。你需要自行将最终的包进行压缩。

CommonJS 和 ES Module 版本同时保留原始的 `process.env.NODE_ENV` 检测，以决定它们应该运行在什么模式下。你应该使用适当的打包工具配置来替换这些环境变量以便控制 Vue 所运行的模式。把 `process.env.NODE_ENV` 替换为字符串字面量同时可以让 UglifyJS 之类的压缩工具完全丢掉仅供开发环境的代码块，以减少最终的文件尺寸。

webpack

在 webpack 4+ 中，你可以使用 `mode` 选项：

```
module.exports = {
  mode: 'production'
}
```

但是在 webpack 3 及其更低版本中，你需要使用 DefinePlugin：

```
var webpack = require('webpack')

module.exports = {
  // ...
  plugins: [
    // ...
    new webpack.DefinePlugin({
      'process.env': {
        NODE_ENV: JSON.stringify('production')
      }
    })
  ]
}
```

Rollup

使用 rollup-plugin-replace：

```
const replace = require('rollup-plugin-replace')

rollup({
  // ...
  plugins: [
    replace({
      'process.env.NODE_ENV': JSON.stringify('production')
    })
  ]
}).then(...)
```

Browserify

为你的包应用一次全局的 envify 转换。

```
NODE_ENV=production browserify -g envify -e main.js | uglifyjs -c -m > build.js
```

也可以移步生产环境部署。

CSP 环境

有些环境，如 Google Chrome Apps，会强制应用内容安全策略 (CSP)，不能使用 `new Function()` 对表达式求值。这时可以用 CSP 兼容版本。完整版本依赖于该功能来编译模板，所以无法在这些环境下使用。

另一方面，运行时版本则是完全兼容 CSP 的。当通过 webpack + vue-loader 或者 Browserify + vueify 构建时，模板将被预编译为 `render` 函数，可以在 CSP 环境中完美运行。

开发版本

重要: GitHub 仓库的 `/dist` 文件夹只有在新版本发布时才会提交。如果想要使用 GitHub 上 Vue 最新的源码，你需要自己构建！

```
git clone https://github.com/vuejs/vue.git node_modules/vue
cd node_modules/vue
npm install
npm run build
```

Bower

Bower 只提供 UMD 版本。

```
# 最新稳定版本
$ bower install vue
```

AMD 模块加载器

所有 UMD 版本都可以直接用作 AMD 模块。

译者注 [1] 对于中国大陆用户，建议将 NPM 源设置为国内的镜像，可以大幅提升安装速度。

介绍

深入了解组件

组件注册

概述

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。 包括以下工具：

- 在 CSS 过渡和动画中自动应用 class
- 可以配合使用第三方 CSS 动画库，如 Animate.css
- 在过渡钩子函数中使用 JavaScript 直接操作 DOM
- 可以配合使用第三方 JavaScript 动画库，如 Velocity.js

在这里，我们只会讲到进入、离开和列表的过渡，你也可以看下一节的 管理过渡状态。

单元素/组件的过渡

Vue 提供了 `transition` 的封装组件，在下列情形中，可以给任何元素和组件添加进入/离开过渡

- 条件渲染 (使用 `v-if`)
- 条件展示 (使用 `v-show`)
- 动态组件
- 组件根节点

这里是一个典型的例子：

```
<div id="demo">
  <button v-on:click="show = !show">
    Toggle
  </button>
  <transition name="fade">
    <p v-if="show">hello</p>
  </transition>
</div>
```

```
new Vue({
  el: '#demo',
  data: {
    show: true
  }
})
```

```
.fade-enter-active, .fade-leave-active {
  transition: opacity .5s;
}
.fade-enter, .fade-leave-to /* .fade-leave-active below version 2.1.8 */ {
  opacity: 0;
}
```

Toggle
hello

`，则 `v-` 是这些类名的默认前缀。如果你使用了 ``，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 常用的过渡都是使用 CSS 过渡。 下面是一个简单例子： ``` html
Toggle render

hello

``` ``` js new Vue({ el: '#example-1', data: { show: true } }) ``` ``` css /* 可以设置不同的进入和离开动画 */ /* 设置持续时间和动画函数 */ .slide-fade-enter-active { transition: all .3s ease; } .slide-fade-leave-active { transition: all .8s cubic-bezier(1.0, 0.5, 0.8, 1.0); } .slide-fade-enter, .slide-fade-leave-to /* .slide-fade-leave-active for below version 2.1.8 */ { transform: translateX(10px); opacity: 0; } ```
Toggle render
hello

Toggle show
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Mauris facilisis enim libero, at lacinia diam fermentum id. Pellentesque habitant morbi tristique senectus et netus.

``` ``` js new Vue({ el: '#example-2', data: { show: true } }) ``` ``` css .bounce-enter-active { animation: bounce-in .5s; } .bounce-leave-active { animation: bounce-in .5s reverse; } @keyframes bounce-in { 0% { transform: scale(0); } 50% { transform: scale(1.5); } 100% { transform: scale(1); } } ```
Toggle show
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Mauris facilisis enim libero, at lacinia diam fermentum id. Pellentesque habitant morbi tristique senectus et netus.

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+)

他们的优先级高于普通的类名，这对于 Vue 的过渡系统和其他第三方 CSS 动画库，如 Animate.css 结合使用十分有用。

示例：

```
<link href="https://cdn.jsdelivr.net/npm/animate.css@3.5.1" rel="stylesheet" type="text/css">

<div id="example-3">
  <button @click="show = !show">
    Toggle render
  </button>
  <transition
    name="custom-classes-transition"
    enter-active-class="animated tada"
    leave-active-class="animated bounceOutRight"
  >
    <p v-if="show">hello</p>
  </transition>
</div>
```

```
new Vue({
  el: '#example-3',
  data: {
    show: true
  }
})
```

Toggle render
hello

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

> 2.2.0 新增

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 `animationend` 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

```
<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>
```

```
// ...
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，回调函数 `done` 是必须的 **。否则，它们会被同步调用，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：

```
<!--
Velocity 和 jQuery.animate 的工作方式类似，也是用来实现 JavaScript 动画的一个很棒的选择
-->
<script src="https://cdnjs.cloudflare.com/ajax/libs/velocity/1.2.3/velocity.min.js"></script>

<div id="example-4">
  <button @click="show = !show">
    Toggle
  </button>
  <transition
    v-on:before-enter="beforeEnter"
    v-on:enter="enter"
    v-on:leave="leave"
    v-bind:css="false"
  >
    <p v-if="show">
      Demo
    </p>
```

```
  </transition>
</div>
```

```
new Vue({
  el: '#example-4',
  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 })
    }
  }
})
```

Toggle
Demo

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

我们之后讨论多个组件的过渡，对于原生标签可以使用 `v-if` / `v-else` 。最常见的多标签过渡是一个列表和描述这个列表为空消息的元素：

```
<transition>
  <table v-if="items.length > 0">
    <!-- ... -->
  </table>
  <p v-else>Sorry, no items found.</p>
</transition>
```

可以这样使用，但是有一点需要注意：

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。即使在技术上没有必要，**给在 `` 组件中的多个元素设置 key 是一个更好的实践。**

示例：

```
<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>
```

工具