Web 前端技术丛书

Vue.js 3.x

从入门到精通（视频教学版）

李小威 著

清華大學出版社
北京

内 容 简 介

本书通过应用示例和综合案例的讲解与演练，使读者快速掌握Vue.js 3.x编程知识，提高使用Vue.js开发网站和移动App的实战能力。本书配套示例源码、PPT课件、同步教学视频、教学大纲与执行进度表、习题与答案、其他超值教学资源。

本书共18章，内容包括快速进入Vue.js的世界、搭建开发与调试环境、熟悉ECMAScript 6的语法、熟悉Vue.js的语法、指令、计算属性、精通监听器、事件处理、class与style绑定、表单输入绑定、组件和组合API、过渡和动画效果、精通Vue CLI和Vite、使用Vue Router开发单页面应用、数据请求库——Axios、状态管理——Vuex、网上购物商城开发实战和电影购票App开发实战等。

本书内容丰富、理论结合实践，可以作为工具书和参考手册，适合Web前端开发初学者、网站与移动App设计和开发人员，也适合作为高等院校、中职学校和培训机构计算机相关专业的师生教学参考。

图书在版编目（CIP）数据

Vue.js 3.x从入门到精通：视频教学版/李小威著. —北京：清华大学出版社，2023.2
（Web前端技术丛书）
ISBN 978-7-302-62741-8

Ⅰ. ①V… Ⅱ. ①李… Ⅲ. ①网页制作工具－程序设计 Ⅳ. ①TP393.092.2

中国国家版本馆CIP数据核字（2023）第027207号

责任编辑：夏毓彦
封面设计：王　翔
责任校对：闫秀华
责任印制：沈　露

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦A座　　邮　　编：100084
社 总 机：010-83470000　　邮　　购：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 装 者：三河市龙大印装有限公司
经　　销：全国新华书店
开　　本：190mm×260mm　　**印　　张：**20　　**字　　数：**540千字
版　　次：2023年3月第1版　　**印　　次：**2023年3月第1次印刷
定　　价：79.00元

产品编号：082585-01

前　言

Vue.js是一套构建用户界面的渐进式框架，采用自底向上增量开发的设计。Vue.js的核心库只关注视图层，并且非常容易学习，与其他库或已有项目整合也非常方便，因此Vue.js能够在很大程度上降低Web前端开发的难度，深受广大Web前端开发人员的喜爱。

本书内容

第1章是快速进入Vue.js的世界，内容包括前端开发技术的发展、MV*模式、Vue.js概述、Vue.js的发展历程、Vue.js 3.x的新变化。

第2章介绍搭建开发与调试环境，内容包括安装Vue.js、安装WebStorm、安装vue-devtools、第一个Vue.js程序。

第3章介绍ECMAScript 6的语法，内容包括ECMAScript 6、块作用域构造let和const、模板字面量、默认参数和rest参数、解构赋值、展开运算符、增强的对象文本、箭头函数、Promise实现、Classes、Module。

第4章介绍Vue.js的语法，内容包括创建应用程序实例、插值、方法选项、生命周期钩子函数、指令、缩写、取消构造函数。

第5章介绍指令，内容包括内置指令、自定义指令、通过指令实现下拉菜单效果。

第6章介绍计算属性，内容包括使用计算属性、计算属性的getter和setter方法、计算属性的缓存、计算属性代替v-for和v-if、使用计算属性设计购物车效果。

第7章是精通监听器，内容包括使用监听器、监听方法、监听对象、使用监听器设计购物车效果。

第8章介绍事件处理，内容包括监听事件、事件处理方法、事件修饰符、按键修饰符、系统修饰键、处理用户注册信息。

第9章介绍class与style绑定，内容包括绑定HTML样式（class）、绑定内联样式（style）、设计隔行变色的商品表。

第10章介绍表单输入绑定，内容包括实现双向数据绑定、单行文本输入框、多行文本输入框、复选框、单选按钮、选择框、值绑定、修饰符、设计用户注册页面。

第11章介绍组件和组合API，内容包括组件是什么、组件的注册、使用prop向子组件传递数据、子组件向父组件传递数据、插槽、组合API、setup()函数、响应式API、访问组件的方式、使用组件创建树状项目分类。

第12章介绍设计过渡和动画效果，内容包括单元素/组件的过渡、初始渲染的过渡、多个元素的过渡、列表过渡、商品编号增加器、设计下拉菜单的过渡动画。

第13章介绍Vue CLI和Vite，内容包括脚手架的组件、脚手架环境搭建、安装脚手架、创建项目、分析项目结构、配置Scss、配置Less和Stuly、配置文件gackage.json、构建工具Vite。

第14章介绍使用Vue Router开发单页面应用，内容包括使用Vue Router、命名路由、命名视图、路由传参、编程式导航、组件与Vue Router间解耦。

第15章介绍Axios，内容包括什么是Axios、安装Axios、Axios的基本用法、Axios API、请求配置、创建实例、配置默认选项、拦截器、显示近7日的天气情况实例。

第16章介绍Vuex，内容包括什么是Vuex、安装Vuex、在项目中使用Vuex。

第17章介绍开发网上购物商城的项目实训。

第18章介绍开发电影购票App的项目实训。

本书特色

（1）知识全面：涵盖所有Vue.js 3.x的知识点，知识由浅入深，便于读者循序渐进地掌握移动网站和App开发技术。

（2）注重操作，图文并茂：在介绍案例的过程中，每一个操作均有对应的插图。这种图文结合的方式，使读者在学习过程中能够直观、清晰地看到操作的过程及效果，便于更快地理解和掌握相关知识点。

（3）易学易用：颠覆传统“看”书的观念，把书变成一本能“操作”的图书。

（4）示例丰富：把知识点融汇于众多的示例中，并且结合实战案例进行讲解和拓展，从而达到“知其然，并知其所以然”的目的。

（5）贴心周到：对读者在学习过程中可能会遇到的疑难问题以“提示”的形式进行说明，避免读者在学习过程中走弯路。

（6）资源丰富：本书提供所有示例的源代码、课件和教学视频，方便读者快速掌握网站前端开发的技能，使本书真正体现“自学无忧”，成为一本物超所值的好书。

（7）技术支持：读者可关注本书的技术支持公众号，向作者索要源代码、教学幻灯片和精品教学视频。在学习过程中遇到问题，也可以通过公众号请作者指点。

超值教学资源下载与技术支持

示例源码、PPT课件、同步教学视频、教学大纲与执行进度表、习题与答案、就业面试题、开发技巧与常见错误等教学资源，请用微信扫描以下二维码下载，也可按页面提示把下载链接转发到

自己的邮箱下载。如果学习本书的过程中发现问题，请联系booksaga@163.com，邮件主题写“Vue.js 3.x从入门到精通（视频教学版）”。作者微信技术支持信息请查阅下载文档中的相关文件获取。

读者对象

本书是一本完整介绍Vue.js前端开发技术的教程，内容丰富，条理清晰，实用性强，适合以下读者学习使用：

- 没有任何 Vue.js 前端开发基础的初学者。
- 希望快速、全面掌握 Vue.js 框架的开发人员。
- 高等院校、中职学校及培训机构的学生。

鸣谢

本书由李小威创作，参与编写的还有王英英、张工厂、刘增杰、胡同夫、刘玉萍、刘玉红。本书的编写虽然倾注了编者的努力，但由于水平有限、时间仓促，书中难免有疏漏之处，欢迎读者批评指正。如果遇到问题或有好的建议，敬请与我们联系，我们将全力提供帮助。

编　者

2023年1月

目　录

第 1 章

快速进入 Vue.js 的世界

在 Web 前端开发技术的发展过程中，网页变得更加动态化，例如轮播图、图片无缝滚动等效果，这都是借助了 JavaScript 的能力。现在的 Web 开发者已经把很多传统的服务端代码放到了浏览器中，这样就产生了成千上万行的 JavaScript 代码，它们连接了许多 HTML 和 CSS 文件，但由于缺乏正规的组织形式，因此网站变得越发臃肿。Vue.js 框架的出现正是为了解决这个问题。本章将重点学习 MV*模式、Vue.js 概述和 Vue.js 3.x 的新变化。

1.1 前端开发技术的发展

Vue.js 是基于 JavaScript 的一套 MVVC 前端框架。在介绍 Vue.js 之前，先来了解一下 Web 前端技术的发展过程。

Web 刚起步阶段，只有可怜的 HTML，浏览器请求某个 URL 时，Web 服务器就把对应的 HTML 文件返回给浏览器，浏览器做解析后展示给用户。随着时间的推移，为了能给不同用户展示不同的页面信息，慢慢发展出了基于服务器的可动态生成 HTML 的语言，例如 ASP、PHP、JSP 等。

但是，当浏览器接收到一个 HTML 后，如果要更新页面的内容，就只能重新向服务器请求获取一份新的 HTML 文件，即刷新页面。在 2G 的流量年代，这种体验很容易让人崩溃，而且还浪费流量。

1995 年，Web 进入 JavaScript 阶段，在浏览器中引入了 JavaScript。JavaScript 是一种脚本语言，浏览器中带有 JavaScript 引擎，用于解析并执行 JavaScript 代码，然后就可以在客户端操作 HTML 页面中的 DOM，这样就解决了不刷新页面的情况，动态地改变用户 HTML 页面的内容。再后来发现编写原生的 JavaScript 代码太烦琐了，还需要记住各种晦涩难懂的 API，最重要的是还需要考虑各种浏览器的兼容性，因此出现了 jQuery，并很快占领了 JavaScript 世界，几乎成为前端开发的标配。

直到 HTML5 的出现，前端能够实现的交互功能越来越多，代码也越来越复杂，从而出现了各种 MV*框架，使得网站开发进入 SPA（Single Page Application，单页应用程序）时代。SPA 是指只

有一个 Web 页面的应用。单页应用程序是加载单个 HTML 页面，并在用户与程序交互时动态更新该页面的 Web 应用程序。浏览器一开始会加载必需的 HTML、CSS 和 JavaScript，所有的操作都在这个页面上完成，由 JavaScript 来控制交互和页面的局部刷新。

2015 年 6 月，ECMAScript 6 发布，其正式名称为 ECMAScript 2015。该版本增加了很多新的语法，从而拓展了 JavaScript 的开发潜力。在 Vue.js 项目开发中经常会用 ECMAScript 6 语法。

1.2 MV*模式

MVC 是 Web 开发中应用非常广泛的一种架构模式，之后又演变成了 MVVM 模式。

1.2.1 MVC 模式

随着 JavaScript 的发展，渐渐显现出各种不和谐：组织代码混乱，业务与操作 DOM 杂合，所以引入了 MVC 模式。

在 MVC 模式中，M 指模型（Model），是后端传递的数据；V 指视图（View），是用户所看到的页面；C 指控制器（Controller），是页面业务逻辑。MVC 模式示意图如图 1-1 所示。

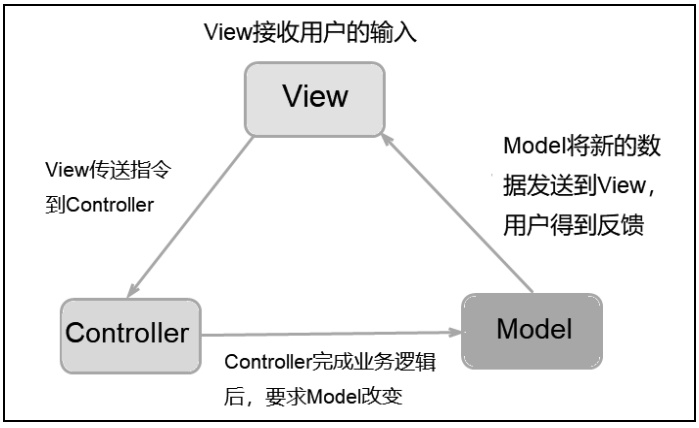

图 1-1 MVC 模式示意图

使用 MVC 模式的目的是将 Model 和 View 的代码分离，实现 Web 应用系统的职能分工。MVC 模式是单向通信的，也就是 View 和 Model 需要通过 Controller 来承上启下。

1.2.2 MVVM 模式

随着网站前端开发技术的发展，又出现了 MVVM 模式。不少前段框架采用了 MVVM 模式，例如当前比较流行的 Angular 和 Vue.js。

MVVM 是 Model-View-ViewModel 的简写。其中 MV 和 MVC 模式中的意思一样，VM 指 ViewModel，是视图模型。

MVVM 模式示意图如图 1-2 所示。

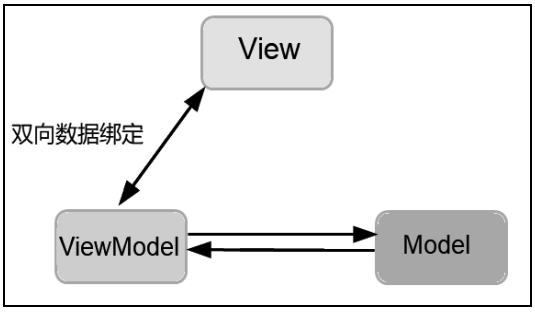

图 1-2　MVVM 模式示意图

ViewModel 是 MVVM 模式的核心，是连接 View 和 Model 的桥梁。它有两个方向：

（1）将模型转化成视图，将后端传递的数据转化成用户所看到的页面。
（2）将视图转化成模型，即将所看到的页面转化成后端的数据。

在 Vue.js 框架中，这两个方向都实现了，就是 Vue.js 中数据的双向绑定。

1.3　Vue.js 概述

Vue.js 是一套构建前端的 MVVM 框架，它集合了众多优秀主流框架设计的思想，轻量、数据驱动（默认单向数据绑定，但也支持双向数据绑定）、学习成本低，且可与 Webpack/Gulp 构建工具结合，以实现 Web 组件化开发、构建和部署等。

Vue.js 本身就拥有一套较为成熟的生态系统：Vue+vue-router+Vuex+Webpack+Sass/Less，不仅可以满足小的前端项目开发，也能完全胜任大型的前端应用开发，包括单页面应用和多页面应用等。Vue.js 可实现前端页面和后端业务分离、快速开发、单元测试、构建优化、部署等。

提到前端框架，当下比较流行的有 Vue.js、React.js 和 Angular.js。Vue.js 以容易上手的 API、不俗的性能、渐进式的特性和活跃的社区从中脱颖而出。截至目前，Vue.js 在 GitHub 上的 star 数已经超过了其他两个框架，成为最热门的框架。

Vue.js 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以及各种支持类库结合使用时，Vue.js 完全能够为复杂的单页应用提供驱动。

Vue.js 的目标就是通过尽可能简单的 API 实现响应、数据绑定和组合的视图组件，核心是一个响应的数据绑定系统。Vue.js 被定义成一个用来开发 Web 界面的前端框架，是一个非常轻量级的工具。使用 Vue.js 可以让 Web 开发变得简单，同时也颠覆了传统前端开发的模式。

Vue.js 是渐进式的 JavaScript 框架，如果已经有一个现成的服务端应用，可以将 Vue.js 作为该应用的一部分嵌入其中，带来更加丰富的交互体验。或者，如果希望将更多的业务逻辑放到前端来实现，那么 Vue.js 的核心库及其生态系统也可以满足用户的各种需求。

和其他前端框架一样，Vue.js 允许将一个网页分割成可复用的组件，每个组件都包含属于自己的 HTML、CSS 和 JavaScript，如图 1-3 所示，以用来渲染网页中相应的地方。

这种把网页分割成可复用组件的方式就是框架“组件化”的思想。

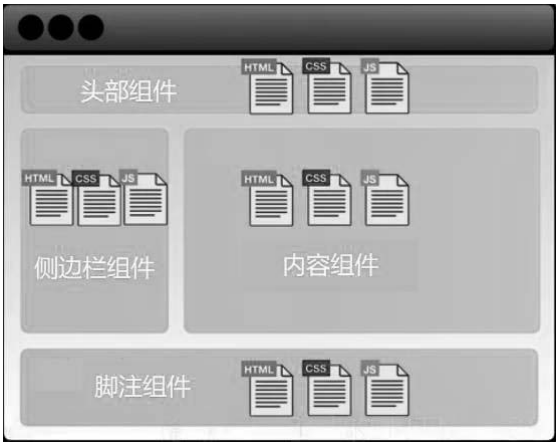

图 1-3　组件化

Vue.js 组件化的理念和 React 异曲同工——一切皆组件。Vue.js 可以将任意封装好的代码注册成组件，例如 Vue.component('example',Example)，可以在模板中以标签的形式调用。

Example 是一个对象，组件的参数配置经常使用到的是 template，它是组件将要渲染的 HTML 内容。

例如，example 组件的调用方式如下：

```
<body>
<hi>我是主页</hi>
<!-- 在模板中调用 example 组件 -->s
<example></example>
<p>欢迎访问我们的网站</p>
</body>
```

如果组件设计合理，在很大程度上可以减少重复开发，而且配合 Vue.js 的单文件组件（vue-loader），可以将一个组件的 CSS、HTML 和 JavaScript 都写在一个文件里，做到模块化的开发。此外，Vue.js 也可以与 vue-router 和 vue-resource 插件配合起来，以支持路由和异步请求，这样就满足了开发 SPA 的基本条件。

在 Vue.js 中，单文件组件是指一个后缀名为.vue 的文件，它可以由各种各样的组件组成，大至一个页面组件，小至一个按钮组件。在后面的章节将详细介绍单文件组件的实现。

1.4　Vue.js 的发展历程

Vue.js 正式发布于 2014 年 2 月，包含 70 多位开发人员的贡献。从脚手架、构建、组件化、插件化，到编辑器工具、浏览器插件等，基本涵盖了从开发到测试等多个环节。

Vue.js 的发展过程如下：

2013 年 12 月 24 日，发布 0.7.0 版本。
2014 年 1 月 27 日，发布 0.8.0 版本。
2014 年 2 月 25 日，发布 0.9.0 版本。

2014 年 3 月 24 日，发布 0.10.0 版本。
2015 年 10 月 27 日，正式发布 1.0.0 版本。
2016 年 4 月 27 日，发布 2.0 的 Preview 版本。
2017 年第一个发布的 Vue.js 版本为 v2.1.9，最后一个发布的 Vue.js 版本为 v2.5.13。
2019 年发布 Vue.js 的 2.6.10 版本，也是比较稳定的版本。
2020 年 09 月 18 日，Vue.js 3.x 正式发布。

1.5　Vue.js 3.x 的新变化

Vue.js 3.x 并没有延用 Vue.js 2.x 版本的代码，而是从头重写了整个框架，代码采用 TypeScript 进行编写，新版本的 API 全部采用普通函数，在编写代码时可以享受完整的性能推断。

与 Vue.js 2.x 版本相比，Vue.js 3.x 具有以下新变化。

1. 重构响应式系统

Vue.js 2.x 利用 Object.defineProperty()方法侦查对象的属性变化，该方法有一定的缺点：

（1）性能较差。
（2）在对象上新增属性是无法被侦测的。
（3）改变数组的 length 属性是无法被侦测的。

Vue.js 3.x 重构了响应式系统，使用 Proxy 替换 Object.defineProperty。Proxy 被称为代理，它的 Proxy 优势如下：

（1）性能更优异。
（2）可直接监听数组类型的数据变化。
（3）监听的目标为对象本身，不需要像 Object.defineProperty 一样遍历每个属性，有一定的性能提升。
（4）Proxy 可拦截 apply、ownKeys、has 等 13 种方法，而 Object.defineProperty 不行。

2. 更好的性能

Vue.js 3.x 重写了虚拟 DOM 的实现，并优化了编译模板，提升了组件的初始化速度，更新的性能提升了 1.3~2 倍，服务器端渲染速度提升了 2~3 倍。

3. tree-shaking 支持

Vue.js 3.x 只打包真正需要的模块，删除了无用的模块，从而减小了产品发布版本的大小。而在 Vue.js 2.x 中，很多用不到的模块也会被打包进来。

4. 组合 API

Vue.js 3.x 中引入了基于函数的组合 API。在引入新的 API 之前，Vue 还有其他替代方案，它们

提供了诸如 Mixin、HOC（高阶组件）、作用域插槽之类的组件之间的可复用性，但是所有方法都有自身的缺点，因此它们未被广泛使用。

（1）一旦应用程序包含一定数量的 Mixins，就很难维护。开发人员需要访问每个 Mixin，以查看数据来自哪个 Mixin。

（2）HOC 模式不适用于.vue 单文件组件，因此在 Vue 开发人员中不被广泛推荐或使用。

（3）作用域插槽的内容会封装到组件中，但是开发人员最终拥有许多不可复用的内容，并在组件模板中放置了越来越多的插槽，导致数据来源不明确。

组合 API 的优势如下：

（1）由于 API 是基于函数的，因此可以有效地组织和编写可重用的代码。

（2）将共享逻辑分离为功能来提高代码的可读性。

（3）可以实现代码分离。

（4）在 Vue 应用程序中更好地使用 TypeScript。

5. Teleport（传送）

Teleport 是一种能够将模板移动到 DOM 中 Vue 应用程序之外的其他位置的技术。像 modals 和 toast 等元素，如果嵌套在 Vue 的某个组件内部，那么处理嵌套组件的定位、z-index 样式就会变得很困难。很多情况下，需要将它与 Vue 应用的 DOM 完全剥离，这样管理起来会容易很多，此时就需要用到 Teleport。

6. Fragment（碎片化节点）

在 Vue 2.x 中，每个组件必须有一个唯一的根节点，所以，写每个组件模板时都要套一个父元素。在 Vue 3.x 中，新增了标签元素<Fragment></Fragment>，从而不再限于模板中的单个根节点，组件可以拥有多个节点。这样做可以减少标签层级，减小内存占用。

7. 更好的 TypeScript 支持

Vue.js 3.x 是用 TypeScript 编写的库，可以享受自动的类型定义提示。JavaScript 和 TypeScript 中的 API 相同，从而无须担心兼容性问题。结合使用支持 Vue.js 3.x 的 TypeScript 插件，开发更加高效，还可以拥有类型检查、自动补全等功能。

1.6 疑 难 解 惑

疑问 1：前端开发的技术体系是什么？

目前的前端技术已经形成了一个大的技术体系。

（1）以 GitHub 为代表的代码管理仓库。

（2）以 NPM 和 Yarn 为代表的包管理工具。

（3）ECMAScript 6、TypeScript 及 Babel 构成的脚本体系。

（4）HTML5、CSS 3 及其相应的处理技术。

（5）以 React、Vue、Angular 为代表的前端框架。

疑问 2：在 Vue.js 中怎么理解 MVVM？

MVVM 是 Model-View-ViewModel 的缩写。Model 代表数据模型，也可以在 Model 中定义数据修改和操作的业务逻辑。View 代表 UI 组件，它负责将数据模型转化成 UI 展现出来。

ViewModel 监听模型数据的改变和控制视图行为、处理用户交互，简单理解就是一个同步 View 和 Model 的对象，连接 Model 和 View。

在 MVVM 架构下，View 和 Model 之间并没有直接的联系，而是通过 ViewModel 进行交互，Model 和 ViewModel 之间的交互是双向的，因此 View 数据的变化会同步到 Model 中，而 Model 数据的变化也会立即反映到 View 上。

ViewModel 通过双向数据绑定把 View 层和 Model 层连接起来，而 View 和 Model 之间的同步工作完全是自动的，无须人为干涉，因此开发者只需关注业务逻辑，不需要手动操作 DOM，不需要关注数据状态的同步问题，复杂的数据状态维护完全由 MVVM 来统一管理。

第 2 章

搭建开发与调试环境

在开发 Vue 前端页面之前，首先需要搭建开发和调试环境，主要包括安装 Vue.js 的方法、安装开发工具 WebStorm、安装调试工具 vue-devtools，最后通过一个 Vue.js 程序检验开发和调试环境是否搭建成功。

2.1 安装 Vue.js

Vue.js 的安装有 4 种方式：

（1）使用 CDN 方式。
（2）使用 NPM 方式。
（3）使用命令行工具（Vue CLI）方式。
（4）使用 Vite 方式。

2.1.1 使用 CDN 方式

CDN（Content Delivery Network，内容分发网络）是构建在现有网络基础之上的智能虚拟网络，依靠部署在各地的边缘服务器，通过中心平台的负载均衡、内容分发、调度等功能模块，使用户就近获取所需的内容，降低网络堵塞，提高用户访问响应速度和命中率。CDN 的关键技术主要有内容存储和分发技术。

使用 CDN 方式来安装 Vue 框架，就是选择一个提供稳定 Vue.js 链接的 CDN 服务商。选择 CDN 后，在页面中引入 Vue 的代码如下：

```
<script src="https://unpkg.com/vue@next"></script>
```

2.1.2　NPM

NPM 是一个 Node.js 包管理和分发工具，也是整个 Node.js 社区最流行、支持第三方模块最多的包管理器。在安装 Node.js 环境时，安装包中包含 NPM，如果安装了 Node.js，则不需要再安装 NPM。

用 Vue 构建大型应用时，推荐使用 NPM 安装。NPM 能很好地和诸如 Webpack 或 Browserify 模块打包器配合使用。

使用 NPM 安装 Vue.js 3.x：

```
# 最新稳定版
$ npm install vue@next
```

由于国内访问国外的服务器非常慢，而 NPM 的官方镜像就是国外的服务器，为了节省安装时间，推荐使用淘宝 NPM 镜像 CNPM，在命令提示符窗口中输入下面的命令并执行：

```
npm install -g cnpm --registry=https://registry.npm.taobao.org
```

以后可以直接使用 cnpm 命令安装模块，代码如下：

```
cnpm install 模块名称
```

注意：通常在开发 Vue.js 3.x 的前端项目时，多数情况下会使用 Vue CLI 先搭建脚手架项目，此时会自动安装 Vue 的各个模块，不需要使用 NPM 单独安装 Vue。

2.1.3　命令行工具（CLI）

Vue 提供了一个官方的脚手架（Vue CLI），使用它可以快速搭建一个应用。搭建的应用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。

例如想构建一个大型的应用，可能需要将应用分割成各自的组件和文件，如图 2-1 所示，此时便可以使用 Vue CLI 快速初始化工程。

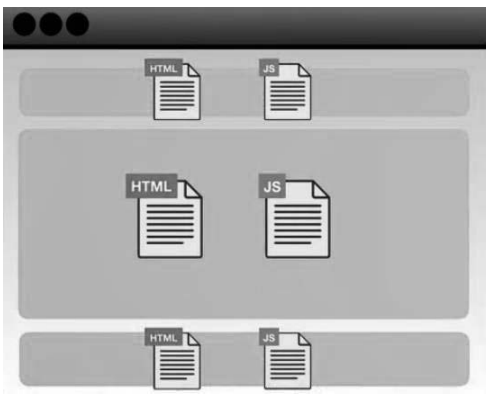

图 2-1　各自的组件和文件

因为初始化的工程可以使用 Vue 的单文件组件，它包含各自的 HTML、JavaScript 以及带作用域的 CSS 或者 SCSS，格式如下：

```
<template>
    HTML
</template>
<script>
    JavaScript
</script>
<style scoped>
    CSS 或者 SCSS
</style>
```

Vue 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。如果是初学者，建议熟悉

Vue 本身之后再使用 Vue CLI 工具。本书后面的章节将具体介绍脚手架的安装以及如何快速创建一个项目。

2.1.4 使用 Vite 方式

Vite 是 Vue 的作者尤雨溪开发的 Web 开发构建工具，它是一个基于浏览器原生 ES 模块导入的开发服务器。在开发环境下，利用浏览器去解析 import，在服务器端按需编译返回，完全跳过了打包这个概念，服务器随启随用。本书后面的章节将具体介绍 Vite 的使用方法。

2.2 安装 WebStorm

WebStorm 是一款前端页面开发工具。该工具的主要优势是有智能提示、智能补齐代码、代码格式化显示、联想查询和代码调试等。对于初学者而言，WebStorm 不仅功能强大，而且非常容易上手操作，被广大前端开发者誉为 Web 前端开发神器。

下面以 WebStorm 英文版为例进行讲解。首先打开浏览器，进入 WebStorm 官网下载页面，如图 2-2 所示。单击 Download 按钮，即可开始下载 WebStorm 安装程序。

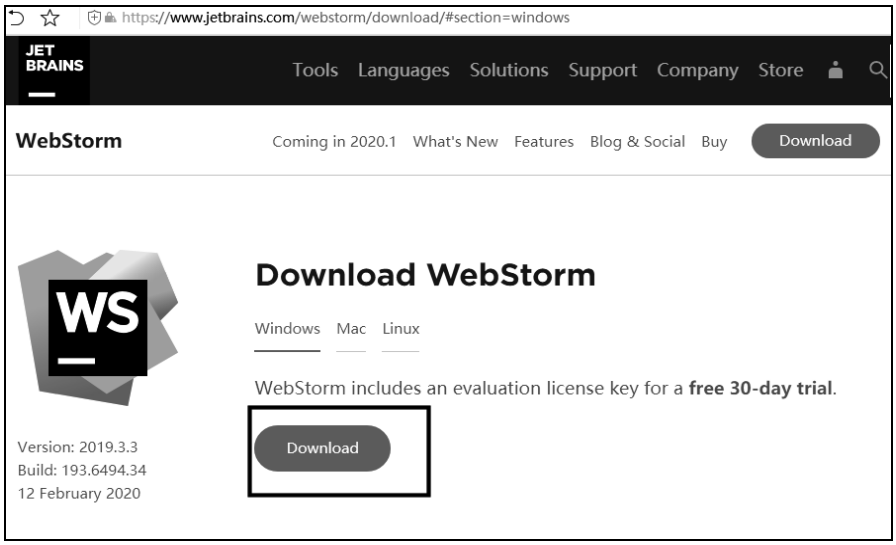

图 2-2 WebStorm 官网下载页面

1. 安装 WebStorm 2019

下载完成后，即可进行安装，具体操作步骤如下：

（1）双击下载的安装文件，进入安装 WebStorm 的欢迎界面，如图 2-3 所示。

（2）单击 Next 按钮，进入选择安装路径窗口，单击 Browse...按钮，即可选择新的安装路径，这里采用默认的安装路径，如图 2-4 所示。

（3）单击 Next 按钮，进入选择安装选项窗口，勾选所有复选框，如图 2-5 所示。

图 2-3　欢迎界面

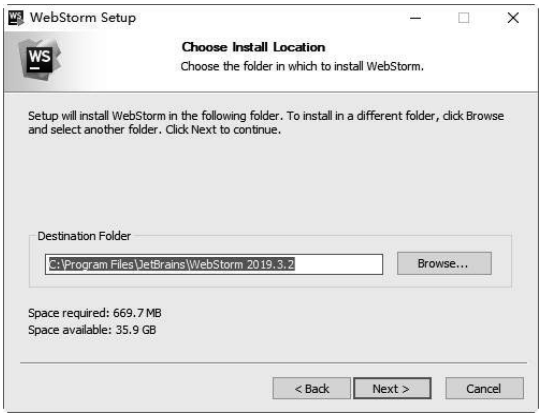

图 2-4　选择安装路径窗口

（4）单击 Next 按钮，进入选择开始菜单文件夹窗口，默认为 JetBrains，如图 2-6 所示。

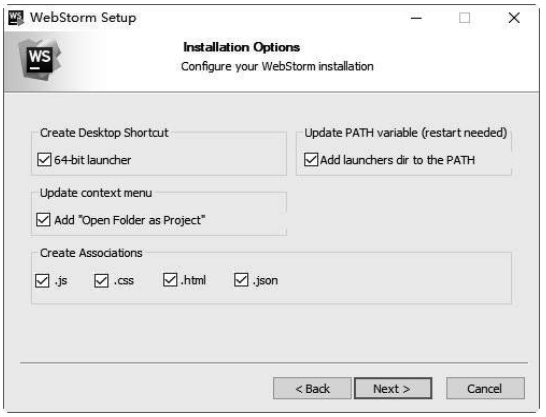

图 2-5　选择安装选项窗口

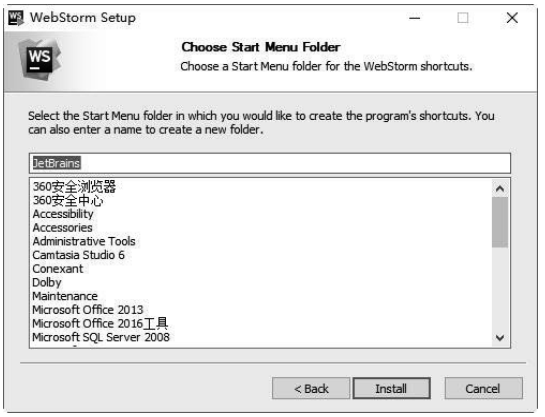

图 2-6　选择开始菜单文件夹窗口

（5）单击 Install 按钮，开始安装软件并显示安装进度，如图 2-7 所示。

（6）安装完成后，单击 Finish 按钮，如图 2-8 所示。

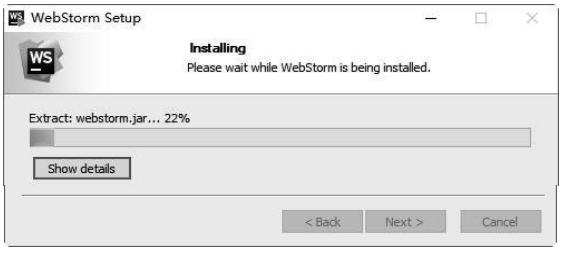

图 2-7　开始安装 WebStorm

2. 创建和运行 HTML 文件

（1）单击 Windows 桌面上的 WebStorm 2019.3.2 x64 快捷键，打开 WebStorm 欢迎界面，如图 2-9 所示。

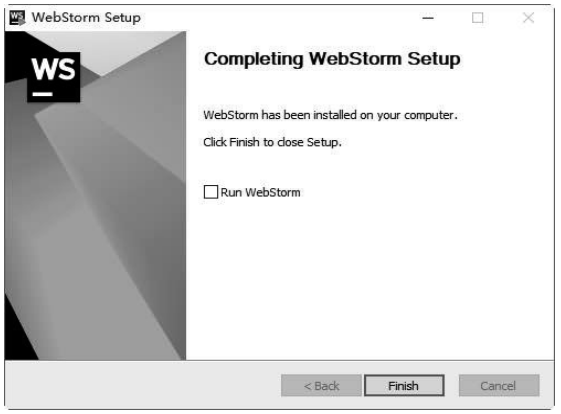

图 2-8　开始安装 WebStorm

图 2-9　WebStorm 欢迎界面

（2）单击 Open 按钮，在弹出的对话框中选择创建好的文件夹 codeHome，如图 2-10 所示。

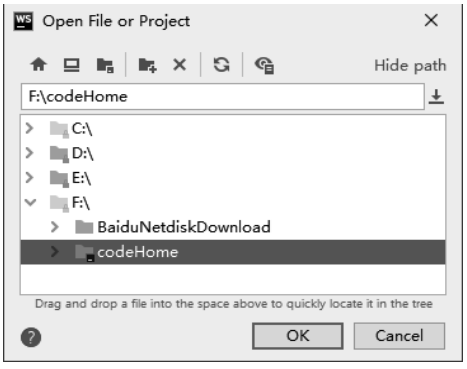

图 2-10　设置工程存放的路径

（3）单击 OK 按钮，进入 WebStorm 主界面，选择 File→New→HTML File 命令，如图 2-11 所示。

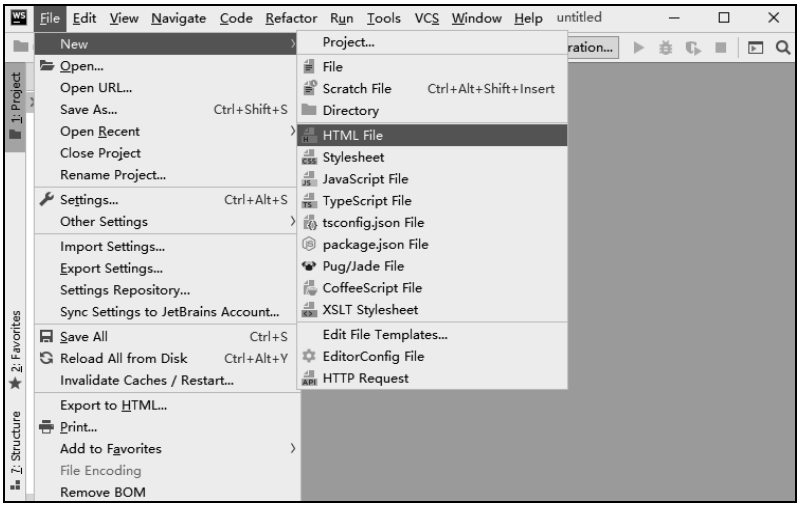

图 2-11　创建一个 HTML 文件

（4）打开 New HTML File 对话框，输入文件名 index.html，选择文件类型为 HTML 5 file，如图 2-12 所示。

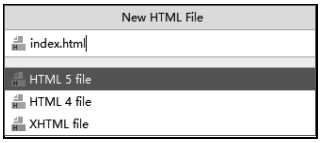

图 2-12　输入文件的名称

（5）按 Enter 键即可查看新建的 HTML 5 文件，接着就可以编辑 HTML 5 文件了。例如，在<body>标记中输入文字“大家一起学习 Vue.js”，如图 2-13 所示。

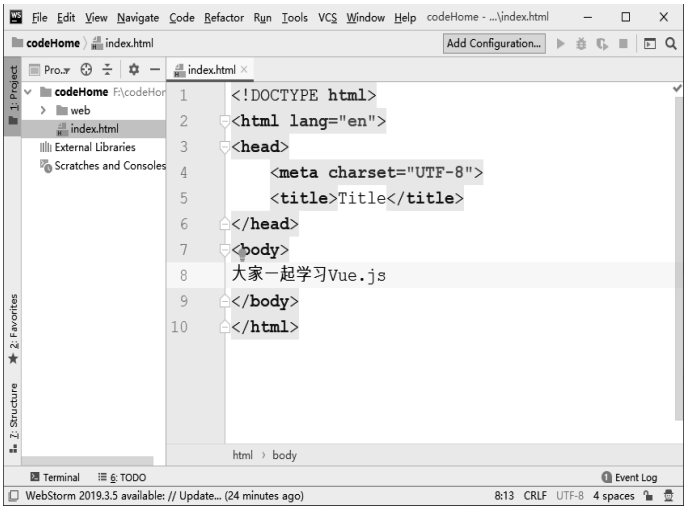

图 2-13　输入文本内容

（6）在谷歌浏览器（下文的浏览器都表示谷歌浏览器）中打开文件的路径，或者打开软件右上角的浏览器工具栏，如图 2-14 所示，选择指定的浏览器，单击即可打开。

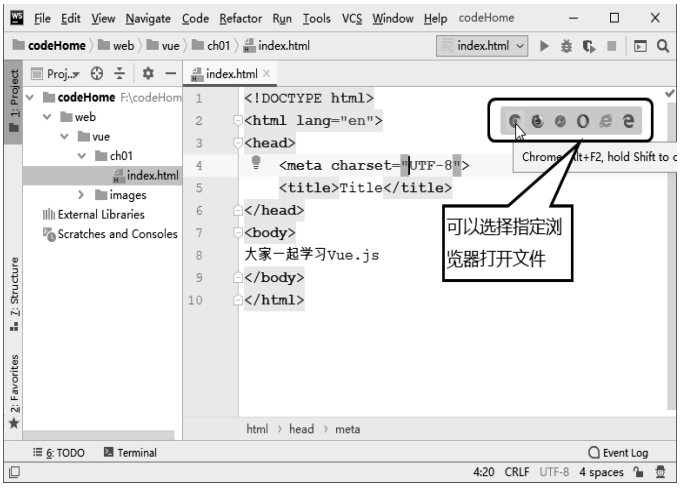

图 2-14　浏览器工具栏

在浏览器中显示的效果如图 2-15 所示。

图 2-15　index.html 文件显示效果

2.3　安装 vue-devtools

vue-devtools 是一款调试 Vue.js 应用的开发者浏览器扩展，可以在浏览器开发者工具下调试代码。不同的浏览器有不同的安装方法，以浏览器为例，其具体安装步骤如下：

（1）打开浏览器，单击“自定义和控制”按钮，在打开的下拉菜单中选择“更多工具”菜单项，然后在弹出的子菜单中选择“扩展程序”菜单项，如图 2-16 所示。

图 2-16　选择“扩展程序”菜单项

（2）在“扩展程序”页面中单击“Chrome 网上应用店”链接，如图 2-17 所示。

图 2-17　“扩展程序”页面

（3）在“chrome 网上应用店”页面搜索 vue-devtools，如图 2-18 所示。

图 2-18　chrome 网上应用店

（4）添加搜索到的扩展程序 Vue.js devtools，如图 2-19 所示。

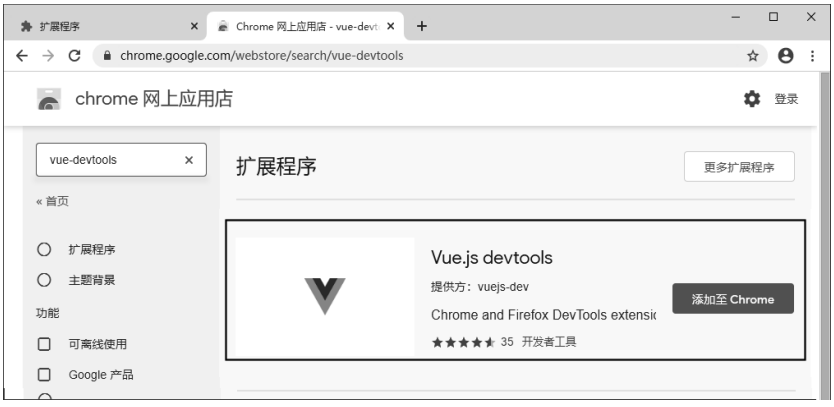

图 2-19　添加扩展程序

（5）在弹出的窗口中单击“添加扩展程序”按钮，如图 2-20 所示。

图 2-20　单击“添加扩展程序”按钮

（6）添加完成后，回到扩展程序页面，可以发现已经显示了 Vue devtools 调试程序，如图 2-21 所示。

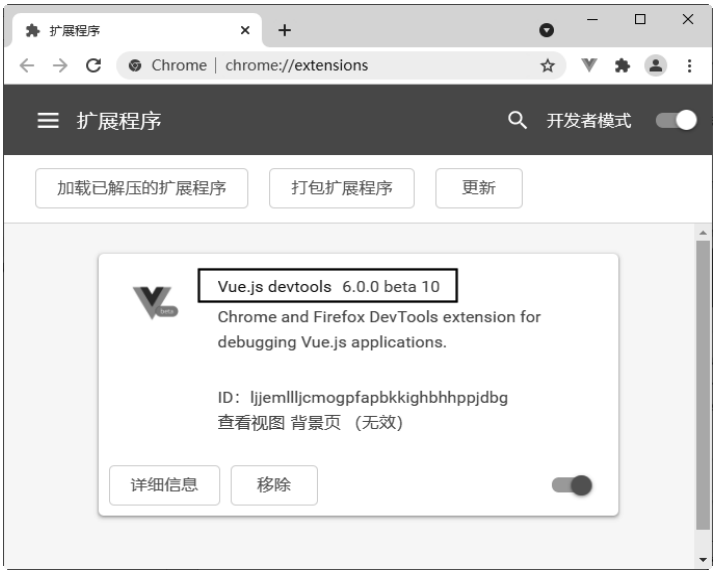

图 2-21　扩展程序页面

（7）单击“详细信息”按钮，在展开的页面中选择“允许访问文件网址”选项，如图 2-22 所示。

图 2-22　详细信息页面

2.4　第一个 Vue.js 程序

接下来让我们动手感受一下 Vue.js，构建一个“水果介绍”的简单页面。和许多 JavaScript 应用一样，首先从网页中需要展示的数据开始。使用 Vue 的起步非常简单，安装 Vue 库，使用 Vue.createApp 创建一个应用程序实例。Vue 在创建组件实例时会调用 data()函数，该函数将返回数据对象，最后通过 mount()方法在指定的 DOM 元素上装载应用程序实例的根组件，从而实现数据的双向绑定。

【例 2.1】　编写“水果介绍”页面（源代码\ch02\2.1.html）。

这里使用 v-bind 指令绑定 IMG 的 src 属性，使用{{}}语法（插值语法）显示标题<h2>的内容。

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    <div id="app">
        <div><img v-bind:src="url" width="450"></div>
        <h2>{{ explain }}</h2>
    </div>
    <!--引入 Vue 文件-->
    <script src="https://unpkg.com/vue@next"></script>
    <script>
    //创建一个应用程序实例
    const vm= Vue.createApp({
    //该函数返回数据对象
    data(){
       return{
          url:'1.jpg',
          explain:'苹果是蔷薇科苹果亚科苹果属植物，其营养价值很高。',
       }
    }
    //在指定的 DOM 元素上装载应用程序实例的根组件
    }).mount('#app');
    </script>
</body>
</html>
```

程序运行效果如图 2-23 所示。

图 2-23　“水果介绍”页面效果

至此，就成功创建了第一个 Vue 应用，这看起来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。可以通过浏览器的 JavaScript 控制台来验证，也可以使用 vue-devtools 调试工具来验证。

例如，在浏览器上按 F12 键，打开控制台并切换到 Console 选项，修改 vm.explain="我最爱吃的就是苹果！"，按回车键后，可以发现页面的内容也发生了改变，效果如图 2-24 所示。

图 2-24　在控制台上修改后的效果

使用 vue-devtools 工具调试，打开浏览器的控制台，选择 Vue 选项，单击左侧的<Root>，同样修改 vm.explain="苹果中营养成分可溶性大，容易被人体吸收！"，单击“保存”按钮，可以发现页面的内容同样也发生了改变，效果如图 2-25 所示。

图 2-25　vue-devtools 调试效果

出现上面这样的效果，是因为 Vue 是响应式的。也就是说当数据变更时，Vue 会自动更新所有网页中用到它的地方。除了小程序中使用的字符串类型外，Vue 对其他类型的数据也是响应的。

特别说明：在之后的章节中，示例不再提供完整的代码，而是根据上下文，将 HTML 部分与 JavaScript 部分单独展示，省略了<head>、<body>等标签以及 Vue.js 的加载等，读者可根据上面示例的代码结构来组织代码。

2.5 疑 难 解 惑

疑问 1：如何查看 Vue.js 3.x 的源码？

研究 Vue.js 3.x 的源码不仅可以理解 Vue.js 3.x 框架，还可以扩展 Vue.js 3.x 的功能，甚至开发基于 Vue.js 3.x 的第三方库。Vue.js 3.x 的源码是托管在 GitHub 上的，通过网址 https://github.com/vuejs/vue 可以下载 Vue.js 3.x 的源码，如图 2-26 所示。单击 Code 按钮，然后选择 Download ZIP 进行下载。

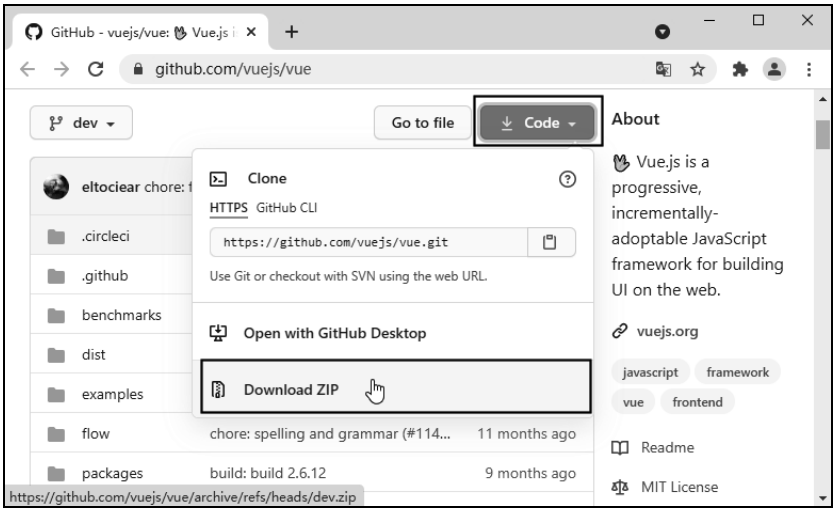

图 2-26　下载 Vue.js 3.x 的源码

疑问 2：安装 vue-devtools 工具时需要注意什么？

安装 vue-devtools 工具的方法有很多种，目前 vue-devtools 6.0 beta 版本支持 Vue.js 3.x，其他的版本有可能还不支持 Vue.js 3.x，所以会出现在控制台中找不到 Vue 选项的问题。因此，在下载 vue-devtools 的时候，一定要多看一下页面下方 vue-devtools 文件的说明。

第 3 章

熟悉 ECMAScript 6 的语法

ECMAScript 6（简称 ES 6）目前基本已经成为业界标准，它的普及速度比 ES 5 要快很多，主要原因是现代浏览器对 ES 6 的支持相当迅速，尤其是 Chrome 和 Firefox 浏览器，已经支持 ES 6 中绝大多数的特性。本章将重点学习 ES 6 中常用的语法规则。

3.1 ECMAScript 6 介绍

1995 年 12 月，Sun 公司与网景公司一起研发了 JavaScript。1996 年 3 月，网景公司发表了支持 JavaScript 的网景导航者（浏览器）2.0 说明。由于 JavaScript 作为网页的客户端脚本语言非常成功，微软于 1996 年 8 月将其引入 Internet Explorer 3.0 中，该软件支持与 JavaScript 兼容的 JScript。1996 年 11 月，网景公司将 JavaScript 提交给欧洲计算机制造商协会（ECMA）进行标准化。ECMA-262 的第一个版本于 1997 年 6 月被 ECMA 组织采纳，这也是 ECMAScript（简称 ES）的由来。

3.1.1 ES 6 的前世今生

ECMAScript 是一种由 ECMA 国际（前身为欧洲计算机制造商协会）通过 ECMA-262 标准化的脚本程序设计语言，该语言在互联网上应用广泛，往往被称为 JavaScript 或 JScript，但实际上后两者是 ECMA-262 标准的实现和扩展。

迄今为止有 7 个 ECMA-262 版本发布，代表着一次次的 JavaScript 更新，具体的版本和详细更新内容如表 3-1 所示。

表3-1 ECMAScript版本更新

版　本	发表日期	与之前版本的差异
1	1997 年 6 月	首版
2	1998 年 6 月	格式修正，以使得其形式与 ISO/IEC16262 国际标准一致

（续表）

版　　本	发表日期	与之前版本的差异
3	1999 年 12 月	强大的正则表达式，更好的词法作用域链处理，新的控制指令，异常处理、错误定义更加明确，数据输出的格式化及其他改变
4	放弃	由于语言的复杂性出现分歧，第 4 版被放弃，其中的部分成为第 5 版及 Harmony 的基础
5	2009 年 12 月	新增严格模式（Strict Mode）。在该版本中提供更彻底的错误检查，以避免因语法不规范而导致的结构出错。澄清了许多第 3 版中的模糊规范，增加了部分新功能，比如 getters 及 setters，支持 JSON 以及在对象属性上更完整的反射
6	2015 年 6 月	多个新的概念和语言特性。ECMAScript Harmony 将会以 ECMAScript 6 发布
6.1	2016 年 6 月	多个新的概念和语言特性

ECMAScript 6 是对语言的重大更新，是自 2009 年 ES 5 标准化以来语言的首次更新。有关 ES 6 的完整规范，请参阅 ES6 标准。

3.1.2　为什么要使用 ES 6

ES 6 是一次重大的版本升级，与此同时，由于 ES 6 秉承着最大化兼容已有代码的设计理念，过去编写的 JS 代码还能正常运行。事实上，许多浏览器已经支持部分 ES 6 特性，并继续努力实现其余特性。这意味着，在一些已经实现部分特性的浏览器中，开发者符合标准的 JavaScript 代码已经可以正常运行，可以更加方便地实现很多复杂的操作，提高开发人员的工作效率。

以下是 ES 6 排名前 10 位的最佳特性列表（排名不分先后）：

- Default Parameters（默认参数）。
- Template Literals（模板文本）。
- Multi-line Strings（多行字符串）。
- Destructuring Assignment（解构赋值）。
- Enhanced Object Literals（增强的对象文本）。
- Arrow Functions（箭头函数）。
- Promises。
- Block-Scoped Constructs Let and Const（块作用域构造 Let 和 Const）。
- Classes（类）。
- Modules（模块）。

3.2　块作用域构造 let 和 const

块级声明用于声明在指定块的作用域之外无法访问的变量。这里的块级作用域是指函数内部或者字符块{}内的区域。

在 ES 6 中，let 是一种新的变量声明方式。在函数作用域或全局作用域中，通过关键字 var 声明

的变量，无论在哪里声明，都会被当成在当前作用域顶部声明的变量，这就是 JavaScript 的变量提升机制。

```
//函数内部
function calculateTotalAmount (vip) {
    //使用 var 方式定义变量
    if(vip) {
        var amount = 0;
    }
    else{
        console.log(amount);  //此处访问 amount，其值为 undefined
    }
}
//字符块中
{
    var amount = 0;
}
console.log(amount);    //此处访问 amount，其值为 0
// for 循环中
for(var i=0;i<100;i++){
}
console.log(amount);    //此处访问 amount，其值为 100
```

这种变量提升机制给开发工作带来了很多麻烦。在 ES 6 中，用 let 限制块级作用域，而 var 限制函数作用域。

```
//函数内部
function calculateTotalAmount (vip) {
    //使用 let 方式定义变量
    if(vip) {
        let amount = 0;
    }
    else{
        console.log(amount);  //此处不能访问 amount，报错 amount is not defined
    }
}
//字符块中
{
    let amount = 0;
}
console.log(amount);    //此处不能访问 amount，报错 amount is not defined
// for 循环中
for(let i=0;i<100;i++){
}
console.log(amount);    //此处不能访问 amount，报错 amount is not defined
```

使用 let 声明变量，还可以防止变量的重复声明。如果在某个作用域下已经存在某个标识符，此时再次使用 let 关键词声明它就会报错。例如以下代码：

```
var amount = 0;
var amount = 100;
var amount = 1000; //可以重复声明
let amount = 10000; //不能再次声明，报错 Inentifier 'amount' has already been declared
```

虽然在同一个作用域下不能重复声明已经存在的标识符，但是在不同的作用域下是可以的。

```
var amount = 0;
{
    let amount = 100;  //可以重复声明
}
```

JavaScript 中的 var 只能声明一个变量，这个变量可以保存任何数据类型的值。ES 6 之前并没有定义声明常量的方式，ES 6 标准中引入了新的关键字 const 来定义常量。

使用 const 定义常量后，常量将无法改变，const 常量的用法说明如下。

1. const 常量只能赋一次值

```
const PI=3.14159;
PI=3.14;             //报错 Assignment to constant variable
```

2. 对象常量

如果使用 const 声明变量，对象本身的绑定不能修改，但对象的属性和值可以修改：

```
const obj={name:"kerr"};
obj.name="tom";
obj={name:"xiaoming"};  //报错 Assignment to constant variable
```

3.3　模板字面量

ES 6 引入了模板字面量（Template Literals），主要通过多行字符串（Multi-line Strings）和字符串占位符对字符串的操作进行增强。

3.3.1　多行字符串

在 ES 5 中，如果一个字符串面量要分为多行，那么可以采用以下两种方法来实现。

（1）在一行的结尾添加反斜杠（\）表示承接上一行的代码。这种方式是利用 JavaScript 的语法 Bug 来实现的：

```
//多行字符串
var roadPoem ="江南好，\
风景旧曾谙。";
```

（2）使用加号（+）来拼接字符串。

```
//多行字符串
var roadPoem ="江南好，风景旧曾谙。"
              +"日出江花红胜火，春来江水绿如蓝。"
              +"能不忆江南？";
```

ES 6 的多行字符串是一个非常实用的功能。模板字面量的基础语法就是使用反引号（`）替换字符串的单、双引号。例如：

```
let roadPoem =`江南好，风景旧曾谙。`;
```

如果需要在字符串中使用反引号，可以使用反斜杠（\）将它转义。

```
let roadPoem =`江南好，\` 风景旧曾谙。`;
```

在 ES 6 中，使用模板字面量语法可以非常方便地创建多行字符串。

```
//多行字符串
let roadPoem =` 江南好，风景旧曾谙。
                日出江花红胜火，春来江水绿如蓝。`;
console.log(roadPoem);
```

输出结果为：

```
江南好，风景旧曾谙。
日出江花红胜火，春来江水绿如蓝。
```

3.3.2 字符串占位符

在一个模板字面量中，可以将 JavaScript 变量或 JavaScript 表达式嵌入占位符并将其作为字符串的一部分输出到结果中。

在 ES 6 中，占位符是使用语法${NAME}的，并将包含的 NAME 变量或者表达式放在反引号中：

```
let name = "xiaoming";
let names = `zhang ${name}c;

let price = 18.8;
let num = 8;
let total = `商品的总价是：${price * num}`;
```

由于模板字面量本身也是 JavaScript 表达式，因此也可以在一个模板字面量中嵌入另一个模板字面量。

```
let price = 10;
let num = 8;
let total = `经过计算，${
            `商品的总价是：${price * num}`
}`;
console.log(total);
```

输出结果为：

```
经过计算，商品的总价是：80
```

3.4 默认参数和 rest 参数

在 ES 5 中，JavaScript 定义默认参数的方式如下：

```
//以前的 JavaScript 定义默认参数的方式
function fun1(height, color, url) {
    var height = height || 50;
    var color = color || "red";
    var url = url || "http://www.baidu.com";
```

```
    函数的其他部分
}
```

但在 ES 6 中，可以直接把默认值放在函数声明中：

```
//新的 JavaScript 定义方式
function fun2 (height=50, color="red", url="http://www.baidu.com"){
    函数的其他部分
}
```

在 ES 6 中，声明函数时，可以为任意参数指定默认值，在已指定默认值的参数后还可以继续声明无默认值的参数。

```
function fun3 (height=50, color="red", url){
    函数的其他部分
}
```

在这种情况下，只有在没有为 height 和 color 传值，或者主动为它们传入 undefined 时才会使用它们的默认值。

在 ES 5 中，无论在函数定义中声明了多少形参，都可以传入任意数量的参数，在函数内部可以通过 arguments 对象接收传入的参数。

```
function data() {
    console.log(arguments);
}
data("li", "ab", "cc");
```

ES 6 引入了 rest 参数，在函数的命名参数前添加了 3 个点，用于获取函数的实参。rest 参数是一个数组，包含自它之后传入的所有参数，通过这个数组名就可以逐一访问里面的参数。

```
function data(...args) {
    console.log(args);
}
data("苹果", "香蕉", "橘子");
```

rest 参数必须放到参数最后位置：

```
function fn(a, b, ...args) {
    console.log(a);
    console.log(b);
    console.log(args);
}
fn(100, 200, 300, 400, 500, 600);
```

3.5　解 构 赋 值

在 ES 5 中，如果需要从某个对象或者数组中提取需要的数据赋给变量，可以采用如下方式：

```
let goods = {
    name: "苹果",
    city: "烟台",
    price: "烟台"
```

```
}
//提取对象中的数据赋给变量
let name = goods.name;
let city = goods.city;
let price = goods.price;
//提取数组中的数据赋给变量
let arr = [100,200,300,400];
let a1 = arr[0], a2 = arr[1], a3 = arr[2], a4 = arr[3];
```

在 ES 6 中，通过使用解构赋值的功能，可以从对象和数组中提取数值，并对变量进行赋值。

1. 对象解构

对象解构的方法是在一个赋值操作符的左边放置一个对象字面量。

```
let goods = {
    name: "苹果",
    city: "烟台",
    price: "烟台"
}
//使用解构赋值的功能
let {name,city,price} = goods;
```

如果变量已经声明了，之后想要用解构语法给变量赋值，则需要把整个解构赋值语句放到一个圆括号中。

```
let goods = {
    name: "苹果",
    city: "烟台",
    price: "烟台"
}
//先声明变量，然后解构赋值
let name,city,price;
({name,city,price} = goods);
```

2. 数组解构

因为没有对象属性名的问题，所以数组解构相对比较简单，使用方括号即可。

```
let arr = [100,200,300,400];
let [a1,a2,a3,a4] = arr;
```

由于变量值是根据数组中元素的顺序进行选取的，因此，如果需要获取指定位置的元素值，可以只为该位置的元素提供变量名。

```
let arr = [100,200,300,400];
//获取第 4 个位置的元素
let [,,,a4] = arr;
console.log(a4);    //输出 400
```

和对象解构不同，如果为已经声明过的变量进行数组解构赋值，不需要把整个解构赋值语句放到一个圆括号中。

```
let arr = [100,200,300,400];
let a1,a2,a3,a4;
[a1,a2,a3,a4] = arr;
```

3.6　展开运算符

展开运算符（Spread Operator）也是 3 个点，允许一个表达式在某处展开。展开运算符在多个参数（用于函数调用）、多个元素（用于数组字面量）或者多个变量（用于解构赋值）的地方可以使用。

1. 在函数调用中使用展开运算符

在 ES 5 中可以使用 apply()方法将一个数组展开成多个参数：

```
function test(a, b, c) { }
var args = [100, 200, 300];
test.apply(null, args);
```

上面的代码中，把 args 数组当作实参传递给了 a、b 和 c。

在 ES 6 中可以更加简洁地来传递数组参数：

```
function test(a,b,c) { }
var args = [100,200,300];
test(...args);
```

这里使用展开运算符把 args 直接传递给 test()函数。

2. 在数组字面量中使用展开运算符

在 ES 6 中，可以直接加一个数组并合并到另一个数组中：

```
var arr1=['a','b','c'];
var arr2=[...arr1,'d','e'];           //['a','b','c','d','e']
```

展开运算符也可以用在 push()函数中，可以不需要再使用 apply()函数来合并两个数组：

```
var arr1=['a','b','c'];
var arr2=['d','e'];
arr1.push(...arr2);                   //['a','b','c','d','e']
```

3. 用于解构赋值

解构赋值也是 ES 6 中新添加的一个特性，这个展开运算符可以用于部分情景：

```
let [arg1,arg2,...,arg3] = [1, 2, 3, 4];
arg1 //1
arg2 //2
arg3 //['3','4']
```

展开运算符在解构赋值中的作用跟之前的作用看上去是相反的，它将多个数组项组合成了一个新数组。

不过要注意，解构赋值中的展开运算符只能用在最后。

```
let [arg1,...,arg2,arg3] = [1, 2, 3, 4];     //报错
```

4. 类数组对象变成数组

展开运算符可以将一个类数组对象变成一个真正的数组对象：

```
var list=document.getElementsByTagName('div');
var arr=[..list];
```

list 是类数组对象，这里通过使用展开运算符使其变成了数组。

3.7 增强的对象文本

ES 6 添加了一系列功能来增强对象文本，从而使得处理对象更加轻松。

1. 通过变量进行对象初始化

在 ES 5 中，对象的属性通常是由具有相同名称的变量创建的。例如：

```
var
    a = 100, b = 200, c = 300;
    obj = {
        a: a,
        b: b,
        c: c
    };
// obj.a = 100, obj.b = 200, obj.c = 300
```

在 ES 6 中，简化如下：

```
const
    a = 100, b = 200, c = 300;
    obj = {
        a
        b
        c
    };
```

2. 简化定义对象方法

在 ES 5 中，定义对象的方法需要 function 语句。例如：

```
var lib = {
    sum: function(a, b) { return a + b; },
    mult: function(a, b) { return a * b; }
};
console.log( lib.sum(100, 200) );  // 300
console.log( lib.mult(100, 200) ); // 20000
```

在 ES 6 中，定义对象的方法简化如下：

```
const lib = {
    sum(a, b)  { return a + b; },
    mult(a, b) { return a * b; }
};
```

```
console.log( lib.sum(100, 200) );  // 300
console.log( lib.mult(100, 200) ); // 20000
```

这里不能使用 ES 6 的箭头函数（=>），因为该方法需要一个名称。如果直接命名每个方法，则可以使用箭头函数（=>）。例如：

```
const lib = {
    sum:  (a, b) => a + b,
    mult: (a, b) => a * b
};
console.log( lib.sum(100, 200) );  // 300
console.log( lib.mult(100, 200) ); // 20000
```

3. 动态属性键

在 ES 5 中，虽然可以在创建对象之后添加变量，但是不能使用变量作为键名称。例如：

```
var
    key1 = 'one',
    obj = {
        two: 200,
        three: 300
    };
obj[key1] = 100;
//表示 obj.one = 100，obj.two = 200，obj.three = 300
```

通过在方括号（[]）内放置表达式，可以在 ES 6 中动态分配对象键。例如：

```
const
  key1 = 'one',
  obj = {
    [key1]: 100,
    two: 200,
    three: 300
  };
//表示 obj.one = 100，obj.two = 200，obj.three = 300
```

4. 解构对象属性中的变量

在 ES 5 中，可以将对象的属性值提取到另一个变量中。例如：

```
var myObject = {
    one:   '洗衣机',
    two:   '冰箱',
    three: '空调'
};
var
    one   = myObject.one, // '洗衣机'
    two   = myObject.two, // '冰箱'
    three = myObject.three; // '空调'
```

在 ES 6 中，通过解构可以创建与等效对象属性同名的变量。例如：

```
const myObject = {
    one:   '洗衣机',
    two:   '冰箱',
```

```
    three: '空调'
};
const { one, two, three } = myObject;
//表示 one = '洗衣机', two = '冰箱', three = '空调'
```

3.8 箭 头 函 数

在 ES 6 中，可以使用箭头（=>）定义函数。下面将讲述不同类型的箭头函数。

```
// 函数没有参数
let hello1 = () => "秋风扫落叶";
console.log(hello1("秋风扫落叶"));      //秋风扫落叶

//函数只有一个参数
let hello2 = mess => mess;
console.log(hello2("秋风扫落叶"));      //秋风扫落叶

// 函数有多于一个的参数
let hello3 = (name,mess) => `${name},${mess}`;
console.log(hello3("古诗欣赏","秋风扫落叶"));      //古诗欣赏,秋风扫落叶

// 空函数
let hello4 = () => {};
```

上面代码的功能和下面的代码一样。

```
// 函数没有参数
function hello1 () {
    return "秋风扫落叶";
}

//函数只有一个参数
function hello2 (mess) {
    return mess;
}
// 函数有多于一个的参数
function hello3 (name,mess) {
    return name+","+mess;
}
// 空函数
function hello4(){}
```

JavaScript 中的 this 并不指向对象本身，其指向是可以改变的，往往会因为上下文的变化而变化。通过使用箭头函数，this 可以按照需要进行设置。

有了箭头函数，就不必像使用 that = this 或 self = this、_this = this、.bind(this)那么麻烦了。例如，下面的代码使用 ES 5 就不是很优雅：

```
var _this = this;
$ ('.btn') .click(function(event){
    this.sendData();
})_
```

在 ES 6 中则不需要使用_this = this：

```
$ ('.btn').click((event) =>{
    this.sendData();
})
```

这个用法并不是完全否定之前的方案，ES 6 委员会决定，以前的 function 的传递方式也是一个很好的方案，所以它们仍然保留了以前的功能。

下面是另一个例子，通过 call 传递文本给 logUpperCase()函数，在 ES 5 中：

```
var logUpperCase = function() {
    var this = this;
    this.string = this.string.toUpperCase();
    return function () {
        return console.log(_this.string);
    }
}
logUpperCase.call({ string: 'ES6 rocks ' })();
```

而在 ES 6 中并不需要用_this 浪费时间：

```
var logUpperCase = function () {
    this.string = this.string.toUpperCase();
    return () => console.log(this.string);
} logUpperCase.call({string: 'ES6 rocks' })();
```

注意：在 ES 6 中，“=>”可以混合和匹配旧的函数一起使用。当在一行代码中用了箭头函数后，它就变成了一个表达式，将返回单个语句的结果。如果结果超过一行，则需要明确使用 return。

3.9　Promise 实现

JavaScript 引擎是基于单线程事件循环的概念构建的，如果是异步操作，则可以使用 ES 6 提供的 Promise 解决方案。

一个 Promise 可以通过 Promise 构造函数创建，这个构造函数只接收一个参数：包含初始化 Promise 代码的执行器函数，在该函数内包含需要异步执行的代码。执行器函数接收两个参数，分别是 reslove()函数和 reject()函数，这两个函数不用编写，由 JavaScript 引擎提供。

下面使用 setTimeout()函数实现异步延迟加载函数：

```
setTimeout(function (){
    let a = 100/5;
}, 1000);
```

在 ES 6 中，可以使用 Promise 重写，虽然在此实例中并不能减少大量代码，甚至还多写了数行，但是逻辑却清晰了不少：

```
const promise = new Promise(function(resolve, reject)){
    //开启异步操作
    setTimeout(function(){
```

```
            try{
                let a = 100/5;
                //执行成功时，调用 resolve()函数
                resolve(a);
            }catch(ex){
                //执行失败时，调用 reject()函数
                reject(ex);
            }
        },1000);
});
```

3.10 Classes（类）

在之前的 JavaScript 版本中，JavaScript 不支持类和类继承的特性，只能使用其他模拟类的定义和类的继承。ES 6 引入了类的概念，通过关键字 class 使类的定义更接近面向对象语言。

在 ES 5 中，没有类的概念，可以通过构造函数和原型混合使用的方式来模拟定义类。

```
function Goods(gName,gPrice){
    this.name = gName;
    this.price = gPrice;
}
Goods.prototype.showName = function (){
    console.log(this.name);
};

Var sGoods = new Goods("洗衣机",6800);
sGoods.showname();
```

在 ES 6 中，使用类可以改写上面的代码：

```
class Goods{
    constructor(gName,gPrice){
        this.name = gName;
        this.price = gPrice;
    }
    showName(){
        console.log(this.name);
    }
}
let sGoods = new Goods("洗衣机",6800);
sGoods.showname();}
```

在 ES 6 中，可以通过 extends 关键字来继承类：

```
class Goods{
    constructor(gName){
        this.name = gName;
    }
    showName(){
        console.log(this.name);
    }
}
```

```
//通过 extends 关键字来继承类 Goods
class Goods1 extends Goods{
    constructor(gName,gPrice){
        super(gName);      //调用父类的 constructor(gName)
        this.price = gPrice;
    }
}
let g1 = new Goods1("洗衣机",6800);
g1.showname();
```

3.11　Modules（模块）

众所周知，在 ES 6 之前，JavaScript 并不支持模块体系，从而导致无法将一个复杂的应用拆分成不同功能的模块，再组合起来使用。于是 JavaScript 社区制定了 AMD 和 CommonJS 及其他解决方法。如今 ES 6 中可以用模块 import 和 export 操作了。

在 ES 5 中，可以在<script>中直接写可以运行的代码（简称 IIFE）或一些库，如 AMD。然而在 ES 6 中，可以用 export 导入类。

下面举个例子，在 ES 5 中，module.js 有 port 变量和 getAccounts()方法：

```
module.exports = {
    port: 3000, getAccounts: function() {
    }
}
//在 ES5 中，main.js 需要依赖 require('module')导入 module.js
var service = require('module.js');
console.log(service.port);            // 3000
```

但在 ES 6 中，将用 export 和 import 进行模块的引入和抛出。例如，以下是使用 ES 6 写的 module.js 文件库：

```
export var port = 3000;
export function getAccounts(url) {
}
```

如果用 ES 6 将上述的 module.js 导入文件 main.js 中，那么就变得非常简单了，只需使用 import {name} from "my-module"语法即可，例如：

```
import {port, getAccounts} from "module";
console.log(port);        // 3000
```

或者可以在 main.js 中导入整个模块，并命名为：

```
import * as service from "module";
console.log(service.port);        // 3000
```

3.12　疑 难 解 惑

疑问 1：ECMAScript 和 JavaScript 是什么关系？

要清楚这个问题，需要回顾历史。1996 年 11 月，JavaScript 的创造者 NetScape 公司决定将 JavaScript 提交给国际标准化组织 ECMA，希望这种语言能够成为国际标准。1997 年，ECMA 发布 262 号标准文件（ECMA-262）的第一版，规定了浏览器脚本语言的标准，并将这种语言称为 ECMAScript，这个版本就是 1.0 版。

该标准从一开始就是针对 JavaScript 语言制定的，但是之所以不叫 JavaScript，有两个原因：一是商标，Java 是 Sun 公司的商标，根据授权协议，只有 NetScape 公司可以合法地使用 JavaScript 这个名字，且 JavaScript 本身已经被 NetScape 公司注册为商标；二是想体现这门语言的制定者是 ECMA，而不是 NetScape，这样有利于保证这门语言的开放性和中立性。

因此，ECMAScript 和 JavaScript 的关系是，前者是后者的规格，后者是前者的一种实现（其他的 ECMAScript 方言还有 JScript 和 ActionScript）。

疑问 2：普通函数和箭头函数的区别是什么？

普通函数是很早就提出的，而箭头函数是 ES 6 提出的，它们两个在语法上不一样，并且它们 this 的指向也不一样。对于普通函数内的 this 来说，如果没有绑定事件元素，this 指向的是 window，在闭包中 this 指向的也是 window；如果函数绑定了事件，但并没有产生闭包，this 指向的是当前调用的事件对象。箭头函数内的 this 指向的是父作用域。

箭头函数不能使用 arguments，普通函数可以使用，arguments 以集合的方式获取函数传递的参数。

箭头函数不能实例化为构造函数，而普通函数可以进行实例化。