第 1 章 初识Vue.js

学习目的与要求

本章主要讲解 Vue.js 的安装方法、开发环境、生命周期以及插值与表达式。通过本章的学习，希望读者掌握 Vue.js 的安装方法，了解 Vue.js 的生命周期。

本章主要内容

- Vue.js 是什么
- 如何安装 Vue.js
- 如何安装 Visual Studio Code 及其插件
- Vue.js 的生命周期
- 插值与表达式

目前广泛应用的 Web 前端三大主流框架是 Angular.js、React.js 和 Vue.js。

Angular.js 由 Google 公司开发，诞生于 2009 年，之前多用 jQuery 开发，其最大特点是把后端的一些开发模式移植到前端实现，例如 MVC、依赖注入等。

React.js 由 Meta 公司开发，正式版于 2013 年推出，比 Angular.js 晚 4 年，采用函数式编程，门槛稍高，但更灵活，开发具有更多可能性。

Vue.js 作为后起之秀（于 2014 年推出），借鉴了前辈 Angular.js 和 React.js 的特点，并做了相关优化，使其更加方便，更容易上手，比较适合初学者。

不管读者学习哪种前端框架，建议都事先了解 HTML、CSS 和 JavaScript 知识。在学习本章之前，假设读者已了解有关 HTML、CSS 和 JavaScript 的知识。

1.1 网站交互方式

Web 网站有单页应用程序（Single-page Application，SPA）和多页应用程序（Multi-page Application，MPA）两种交互方式。

1.1.1 多页应用程序

多页应用程序，顾名思义就是由多个页面组成的站点。在多页应用程序中，每个网页在每次收到相应的请求时都会重新加载。多页应用程序很大，因为它包含了不同页面的数量和层数，这有时甚至被认为很麻烦，读者可以在大多数电子商务网站上找到 MPA 的示例。

多页应用程序以服务端为主导，前、后端混合开发，例如.php、.aspx、.jsp。技术堆栈包括 HTML、CSS、JavaScript、jQuery，有时还包括 AJAX。

❶ 多页应用程序的优点

多页应用程序的优点如下：

（1）搜索引擎优化效果好。搜索引擎在做网页排名时需要根据网页内容给网页添加权重。搜索引擎可以识别 HTML 内容，而多页应用程序的每个页面的所有内容都放在 HTML 中，所以排名效果较好。

（2）更容易扩展。在多页应用程序中，添加到现有应用程序的页面数几乎没有限制。如果需要显示很多信息，建议使用 MPA，因为它可以确保将来能够更轻松地扩展。

（3）深入的数据分析。有许多数据分析工具可以为 MPA 提供有关客户行为、系统功能和其他重要内容的深刻解析，可以分析每个功能的性能，每个网页的受欢迎程度，每个功能所花费的时间，每日和每月的用户数量，以及按年龄、城市、国家/地区划分的受众群体等。

❷ 多页应用程序的缺点

多页应用程序的缺点如下：

（1）开发及维护更加困难且昂贵。与单页应用程序相比，多页应用程序具有更多功能，因此创建它们需要更多的精力和资源，开发时间与要构建的页面数和实现的功能成比例地增加。

Web 应用程序的开发工具更新快，同时市场上还引入了其他库、框架、编程语言或者至少发布了新版本，而多页应用程序通常需要在开发过程中使用多种技术，因此使维护 Web 系统变得更加困难且昂贵。

（2）页面切换慢。多页应用程序每次跳转时都会发出一个 HTTP 请求，如果用户的网速较慢，在页面之间来回跳转时将发生明显的卡顿现象。

（3）较低的绩效指标。多页应用程序中的内容会不断重新加载，增加了服务器的负载，对网页速度和整体系统性能产生负面影响。

1.1.2 单页应用程序

单页应用程序就是只有一个 Web 页面的应用。单页应用程序是加载单个 HTML 页面

并在用户与应用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，都由 JavaScript 来控制，因此对单页应用程序来说模块化的开发和设计显得相当重要。单页应用程序开发技术复杂，所以诞生了许多前端开发框架，例如 Angular.js、React.js、Vue.js 等。

在进行单页应用程序开发时，软件工程师通常采用 HTML5、Angular.js、React.js、Vue.js、Ember.js、AJAX 等技术。

❶ 单页应用程序的优点

单页应用程序的优点如下：

（1）用户体验好。使用单页应用程序就像使用一个原生的客户端软件一样，在切换过程中不会频繁地有被“打断”的感觉。

（2）前后端分离。单页应用程序开发效率高，可维护性强。服务端不关心页面，只关心数据；客户端不关心数据及数据操作，只关心通过接口用数据和服务端交互，处理页面。

（3）局部刷新。单页应用程序只需要加载局部视图即可，不需要整页刷新。

（4）完全的前端组件化。单页应用程序的前端开发不再以页面为主，更多地采用组件化的思想，代码结构和组织方式更加规范化，便于修改和调整。

（5）API 共享。如果服务是多端的（浏览器端、Android、iOS、微信等），单页应用的模式便于在多端共用 API，可以显著地减少服务端的工作量。

（6）组件共享。在某些对性能体验要求不高的场景下或者产品处于快速试错阶段时，借助于一些技术（例如 Hybrid、React Native）在多端共享组件，便于产品的快速迭代，能够节约资源。

❷ 单页应用程序的缺点

单页应用程序的缺点如下：

（1）首次需加载大量资源。首次需要在一个页面上为用户提供产品的所有功能，在加载该页面时，首先加载大量的静态资源，加载时间相对比较长。

（2）对搜索引擎不友好。单页应用的界面数据绝大部分都是异步加载的，很难被搜索引擎搜索到。

（3）安全问题。单页应用更容易受到所谓的跨站点脚本（XSS）攻击，这意味着黑客可以将各种恶意脚本注入应用程序中，原因是缺乏经验的 Web 开发人员将某些功能和逻辑移至客户端。

1.2 MVVM 模式

MVVM 是 Model-View-ViewModel 的缩写，它是一种基于前端开发的架构模式，其核心是提供对 View 和 ViewModel 的双向数据绑定，这使得 ViewModel 的状态改变可以自动传递给 View，即所谓的数据双向绑定。

MVVM 由 Model、View、ViewModel 三部分构成，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，负责将数据模型转化成 UI 展现出来；ViewModel 是一个同步 View 和 Model 的对象。

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。Model、View、ViewModel 三者之间的关系如图 1.1 所示。

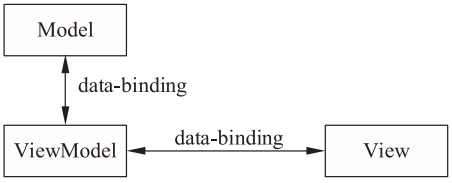

图 1.1　Model、View、ViewModel 三者之间的关系

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。

1.3　Vue.js 是什么

Vue（读音/vjuː/，类似于 view）是一套构建用户界面的渐进式框架。与其他重量级框架不同的是，Vue.js 采用自底向上增量开发的设计。Vue.js 的核心库仅关注视图层，它不仅易于上手，还便于与第三方库或既有项目整合。另一方面，Vue.js 采用单文件组件和 Vue.js 生态系统支持的库开发复杂的单页应用。

Vue.js 本身只是一个 JavaScript 库，其目标是通过尽可能简单的 API 实现相应的数据绑定和组合的视图组件。Vue.js 可以轻松构建 SPA（Single-page Application），通过指令扩展 HTML，通过表达式将数据绑定到 HTML，最大程度地解放 DOM 操作。

Vue.js 具有简单、易用、灵活、高效等特点，用户在掌握 HTML、CSS、JavaScript 的基础上可快速上手。

1.4　安装 Vue.js

将 Vue.js 添加到项目中主要有 4 种方法，即本地独立版本方法、CDN 方法、NPM 方法和命令行工具（Vue CLI）方法。

❶ 本地独立版本方法

用户可通过网址“https://unpkg.com/vue@next”将最新版本的 Vue.js 库（vue.global.js）下载到本地，然后在界面文件中引入 Vue.js 库。示例代码如下：

```
<script src="js/vue.global.js"></script>
```

❷ CDN 方法

读者在进行学习或开发时，在界面文件中可通过 CDN（Content Delivery Network，内

容分发网络）引入最新版本的 Vue.js 库。示例代码如下：

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

对于生产环境，建议使用固定版本，以免因版本不同带来兼容性问题。示例代码如下：

```
<script src="https://unpkg.com/vue@3.0.5/dist/vue.global.js"></script>
```

③ NPM 方法

在用 Vue.js 构建大型应用时推荐使用 NPM 安装最新的稳定版的 Vue.js，因为 NPM 能很好地和 webpack 模块打包器配合使用。示例代码如下：

```
npm install vue@next
```

④ 命令行工具（Vue CLI）方法

Vue.js 提供了一个官方命令行工具（Vue CLI），为单面应用快速搭建繁杂的脚手架。对于初学者来说，不建议使用 NPM 和 Vue CLI 方法安装 Vue.js。NPM 和 Vue CLI 方法的安装过程将在本书后续内容中介绍。

扫一扫

视频讲解

1.5 第一个 Vue.js 程序

对于前端开发工具，极少数程序员使用记事本，大多数程序员使用 JetBrains WebStorm 和 Visual Studio Code（VSCode）。JetBrains WebStorm 是收费的，本书推荐使用 VSCode。

1.5.1 安装 VSCode 及其插件

用户可通过网址“https://code.visualstudio.com”下载 VSCode，本书使用的安装文件是 VSCodeUserSetup-x64-1.52.1.exe（双击即可安装）。VSCode 中的许多插件需要用户安装，例如 Vue.js 的插件 Vetur。打开 VSCode，单击左侧最下面的一个图标，按照图 1.2 所示的步骤安装即可。

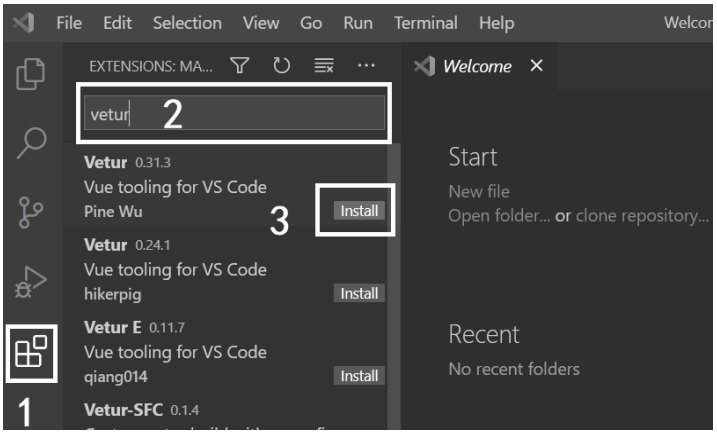

图 1.2 VSCode 中插件的安装

其他插件的安装方法与图 1.2 类似，这里不再赘述。在 VSCode 中，Vue.js 的部分插件的具体描述如下：

（1）Vetur。此插件能够在.vue 文件中实现语法错误检查、语法高亮显示以及代码自动 补全。

（2）ESLint。此插件能够检测代码的语法问题和格式问题，对统一项目的代码风格至关重要。

（3）EditorConfig。EditorConfig 是一种被各种编辑器广泛支持的配置，使用此配置有助于项目在整个团队中保持一致的代码风格。

（4）Path Intellisense。此插件能够在编辑器中输入路径时实现自动补全。

（5）View In Browser。此插件能够在 VSCode 中使用浏览器预览、运行静态文件。

（6）Live Server。此插件很有用，在安装后可以打开一个简单的服务器，而且会自动更新。在安装后，在文件上右击会出现一个名为 Open with Live Server 的选项，此时会自动打开浏览器，默认端口号是 5500。

（7）GitLens。此插件可查看.git 文件提交的历史。

（8）Document This。此插件能够生成注释文档。

（9）HTML CSS Support。此插件能够在编写样式表时自动补全代码，缩短编写时间。

（10）JavaScript Snippet Pack。针对 JavaScript 的插件，包含 JavaScript 的常用语法关键字。

（11）HTML Snippets。此插件包含 HTML 标签。

（12）One Monokai Theme。此插件能够让编者选择自己喜欢的颜色主题编写代码。

（13）vscode-icons。此插件能够让编者选择自己喜欢的图标主题。

1.5.2 创建第一个 Vue.js 应用

每个 Vue.js 应用都是通过用 createApp 函数创建一个新的应用实例开始，具体语法如下：

```
const app = Vue.createApp({ /* 选项 */ })
```

传递给 createApp 的选项用于配置根组件（渲染的起点）。在 Vue.js 应用创建后，调用 mount 函数将 Vue.js 应用实例挂载到一个 DOM 元素（HTML 元素或 CSS 选择器）上。例如，如果把一个 Vue.js 应用实例挂载到 上，应传递#app。示例代码如下：

```
const HelloVueApp = {}                          //配置根组件
const vueApp = Vue.createApp(HelloVueApp)       //创建 Vue 应用实例
```

下面使用 VSCode 开发第一个 Vue.js 程序。

【例 1-1】使用 VSCode 新建一个名为 hellovue.html 的页面，在此页面中使用“<script src="js/vue.global.js"></script>”语句引入 Vue.js。

hellovue.html 的具体代码如下：

```
<div id="hello-vue" class="demo">
     {{ message }}
</div>
<script src="js/vue.global.js"></script>
<script>
   const HelloVueApp = {
```

```
        data() {//Vue 实例的数据对象，ES 语法，等价于 data: function () {}
            return {
                message: 'Hello Vue!!'
            }
        }
    }
    //每个 Vue.js 应用都是通过用 createApp 函数创建一个新的应用实例开始
    //mount 函数把一个 Vue.js 应用实例挂载到<div id="hello-vue"></div>上

</script>
<style>
.demo {
    font-family: sans-serif;

</style>
```

从上述代码中可以看出，hellovue.html 文件由 HTML、JavaScript 和 CSS 3 部分组成，所以读者在学习 Vue.js 之前应掌握 HTML、JavaScript 和 CSS 等内容。

从上述代码中还可以看出，创建一个 Vue.js 应用程序只需要 3 个步骤，即引入 Vue.js 库文件、创建一个 Vue 实例、渲染 Vue 实例。

在 VSCode 中安装 Live Server 插件后，在 hellovue.html 代码上右击会出现一个名为 Open with Live Server 的选项，此时会自动打开浏览器，默认端口号是 5500，如图 1.3 所示。

图 1.3　hellovue.html 的运行效果

在 VSCode 中可以设置 Live Server 插件默认打开的浏览器，首先选择 File → Preferences → Settings 命令打开 Search settings 界面，然后展开 User 下的 Extensions，选中 Live Server Config，在 Settings: Custom Browser 下方修改默认打开的浏览器，如图 1.4 所示。

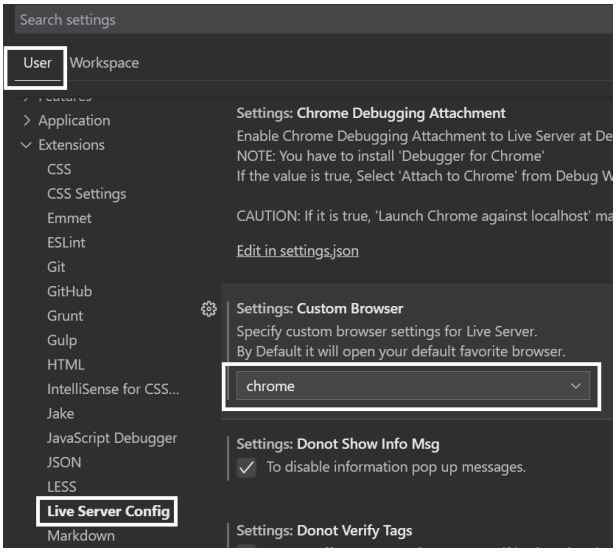

图 1.4　设置 Live Server 插件默认打开的浏览器

1.5.3 声明式渲染

Vue.js 的核心是采用简洁的模板将数据渲染到 DOM 中，例如在 1.5.2 节的 hellovue.html 文件中，通过模板 {{message}} 声明将属性变量 message 的值“Hello Vue!!”渲染到页面显示。

Vue.js 框架在进行声明式渲染时做的主要工作就是将数据和 DOM 建立关联，一切皆响应。例如，例 1-2 中的 counter 属性每秒递增。

【例 1-2】使用 VSCode 新建一个名为 ch1_2.html 的页面，在该页面中使用时钟函数 setInterval 来演示响应式程序。

ch1_2.html 的具体代码如下：

```
<div id="counter" class="demo">
    <!--通过模板获取变量 counter 的值-->
    {{ counter }}
</div>
<script src="js/vue.global.js"></script>
<script>
    const CounterApp = {
        data() {
            return { //声明需要响应式绑定的数据对象，若定义多个键值对，之间用逗号分隔
                counter: 0
            }
        },
        /*mounted 是一个钩子函数，在挂载到实例上后（初始化页面后）调用该函数，一般是第一个业务逻辑在这里开始*/
        mounted() {
            setInterval(() => {
                this.counter++
            }, 1000)
        }
    }
    Vue.createApp(CounterApp).mount('#counter')
</script>
<style>
    .demo {
        font-family: sans-serif;

</style>
```

1.5.4 Vue.js 的生命周期

每个 Vue.js 实例在被创建时都要经过一系列的初始化过程，例如数据监听、编译模板、将实例挂载到 DOM 并在数据变化时更新 DOM 等。同时在这个过程中也会调用一些叫生命周期钩子的函数，在适当的时机执行用户的业务逻辑。

例如，created 钩子函数可用来在一个 Vue.js 实例被创建后执行代码（Vue.js 实例创建后被立即调用，即 HTML 加载完成前）：

```
Vue.createApp({
    data() {
        return {
            message: '测试钩子函数'
        }
    },
    created() {
        //this 指向调用它的 Vue.js 实例

    }
})
```

Vue.js 实例的生命周期共分 8 个阶段（图 1.5），即对应 8 个与 created 类似的钩子函数。

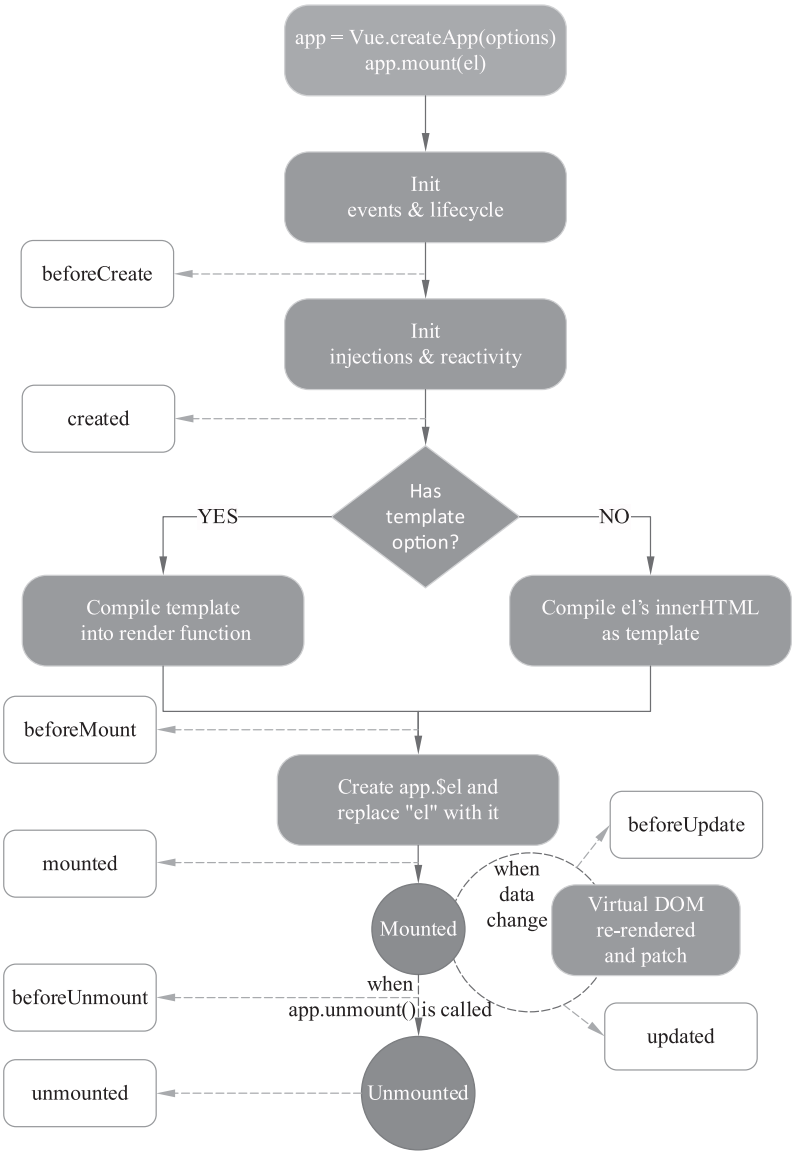

图 1.5　Vue.js 实例的生命周期

（1）beforeCreate（创建前）：在 Vue.js 实例初始化后，数据观测和事件配置前调用，此时 el 和 data 并未初始化，因此无法访问 methods、data、computed 等上的方法和数据。

（2）created（创建后）：在 Vue.js 实例创建后被立即调用，即 HTML 加载完成前，此时 Vue.js 实例已完成数据观测、属性和方法的运算、watch/event 事件回调、data 数据的初始化，然而挂载阶段还没有开始，el 属性目前不可见。这是一个常用的生命周期钩子函数，可以调用 methods 中的方法、改变 data 中的数据、获取 computed 中的计算属性等，通常在此钩子函数中对实例进行预处理。

（3）beforeMount（载入前）：在挂载开始前被调用，Vue.js 实例已完成编译模板、把 data 里面的数据和模板生成 HTML、el 和 data 初始化，注意此时还没有挂载 HTML 到页面上。

（4）mounted（载入后）：在页面加载后调用该函数，这是一个常用的生命周期钩子函数，一般是第一个业务逻辑在此钩子开始，mounted 只会执行一次。

（5）beforeUpdate（更新前）：在数据更新前被调用，发生在虚拟 DOM 重新渲染和打补丁之前，可以在该钩子中进一步更改状态，这不会触发附加的重渲染过程。

（6）updated（更新后）：在由数据更改导致虚拟 DOM 重新渲染和打补丁时调用，在调用时 DOM 已经更新，所以可以执行依赖于 DOM 的操作，注意应该避免在此期间更改状态，否则可能会导致更新无限循环。

（7）beforeUnmount（销毁前）：在 Vue.js 实例销毁前调用（离开页面前调用），这是一个常用的生命周期钩子函数，一般在此时做一些重置的操作，例如清除定时器和监听的 DOM 事件。

（8）unmounted（销毁后）：在实例销毁后调用，调用后事件监听器被移出，所有子实例也被销毁。

图 1.5 展示了 Vue.js 实例的生命周期，读者现在不需要弄明白所有阶段的钩子函数，可以随着不断学习和使用慢慢理解它们。

1.6 插值与表达式

Vue 的插值表达式“{{ }}”的作用是读取 Vue.js 中的 data 数据，显示在视图中，数据更新，视图也随之更新。在“{{ }}”中只能放表达式（有返回值），不能放语句。例如，{{ var a = 1 }}和{{ if (ok) { return message } }}都是无效的。

1.6.1 文本插值

数据绑定最常见的形式就是使用“Mustache（小胡子）”语法（双花括号）的文本插值，它将绑定的数据实时显示出来。例如，例 1-2 中的{{ counter }}，无论何时，绑定的 Vue.js 实例的 counter 属性值发生改变，插值处的内容都将更新。

用户可通过使用 v-once 指令执行一次性插值，即当数据改变时插值处的内容不会更新。示例代码如下：

```
<span v-once>{{ counter }}</span>
```