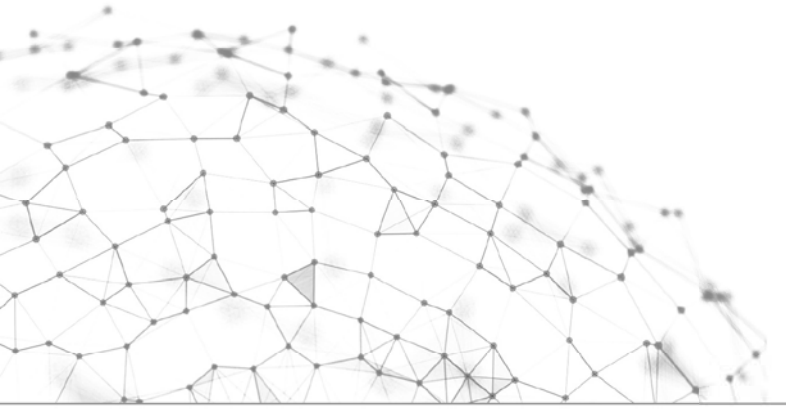


第 1 章

Vue.js 开发基础

【本章概述】

随着这几年前端的快速发展，大多数的前端开发不再执着于 PC 端的应用，在移动端的前端页面中也得到了广泛的应用。但是在开发大型项目时，模块依赖和组件化开发一直没有得到很好的解决，而 Vue.js 凭借构建用户界面的渐进式框架，自底向上逐层应用，只关注视图层，易于上手，方便与第三方库或既有项目整合，能够为复杂的单页应用提供驱动，并且提供了非常简洁、易于理解的 API，因此一经推出，便迅速走红。Vue.js 作为最适合初学者学习的 MVVM 框架之一，其学习成本低，效果好，是前端开发的首选。

【知识导读】

本章要点(已掌握的在方框中打钩)

- ☐ Vue.js 的发展现状
- ☐ 三种架构模式的比较
- ☐ 前端开发工具的安装
- ☐ 前端开发调试工具

1.1 背 景 知 识

Vue.js、Angular.js 和 React.js 是前端三大主流框架，而 Vue.js 作为当今最流行的主流前端框架，相信大多数的人都听说过，那么 Vue.js 到底能做些什么呢？下面我们就来了解关于 Vue.js 的基础以及构成等知识。

1.1.1 客户/服务器体系结构

C/S 就是 Client/Server 的缩写，即客户/服务器模式。在这种结构下，用户界面完全通过浏览器实现。

从硬件角度来看，客户/服务器体系结构主要是在两台及两台以上的机器中进行工作，在客户/服务器体系结构中，有一个总是打开的主机，称为服务器(Server)，它服务来自许多其他称为客户的主机的请求。其中客户(Client)主要是用来提供用户的接口和前端应用所需要的程序。

从软件角度来看，客户/服务器体系结构可以看成是一个信息处理体系，客户部分主要是专门解决应用问题、界面设计、人机交互等方面的问题，而服务器方面主要处理服务操作的实现、数据的组织以及系统性能等。

客户/服务器体系结构如图 1-1 所示。

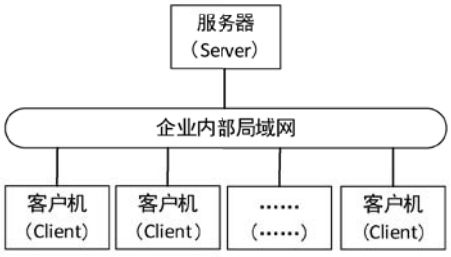


图 1-1 客户/服务器体系结构

1.1.2 HTML、CSS 与 JavaScript

一个 Vue 的页面主要由三部分组成：①Template 标签包裹的界面展示代码(HTML 代码)；②Style 标签包裹的界面布局代码(CSS 样式代码)；③Script 标签包裹的业务实现代码(JS 脚本代码)。

下面主要介绍 HTML、CSS、JavaScript 的具体内容。

HTML 也被称为超文本文档标记语言，英文全称为：Hyper Text Markup Language。HTML 在前端开发中主要制作超级文本文档的简单标记语言，常常用来制作静态网页。HTML 是由 Web 的发明者 Tim Berners-Lee 和同事 Daniel W. Connolly 于 1990 年创立的一种标记语言，它是标准通用化标记语言 SGML 的应用。使用 HTML 编写的超文本文档也被称为 HTML 文档，它能在各个操作系统平台独立，比如 UNIX、Windows 等。在使用 HTML 的过程中，通常将所要表达的信息按某种规则写成 HTML 文件，通过专用的浏览器

来进行识别，这些 HTML 文件就可以“翻译”成可以识别的信息，即我们所见到的网页。

CSS 也被称为层叠样式表，英文全称为 Cascading Style Sheets，其开发目的是用来表现 HTML 或 XML 等文件样式的计算机语言。在前端开发过程中，最初的 HTML 只包含很少的显示属性，随着 HTML 的不断成长，页面设计师的需求也不尽相同，为了能够满足各个设计师的要求，HTML 增加了许多显示功能，考虑到 HTML 页面过于烦琐和臃肿的缺点，于是 CSS 便诞生了。CSS 不仅可以对静态的网页进行修饰，而且可以配合各种脚本语言动态地对网页中的各元素进行格式化。从 HTML 被发明开始，样式就以各种形式存在，不同的浏览器结合它们各自的样式语言为用户提供页面效果的控制。

JavaScript 简称 JS，它是一种直译式的脚本语言，具有函数优先特性，并且是轻量级、解释型或者说即时编译型的编程语言。JavaScript 在过去是作为 Web 页面的脚本语言被人所熟知。它的解释器被称为 JavaScript 引擎，为浏览器的一部分，广泛用于客户端的脚本语言，最早是在 HTML(标准通用标记语言下的一个应用)网页上使用，用来给 HTML 网页增加动态功能，JavaScript 脚本主要是嵌入动态文本于 HTML 的页面来实现自身的功能，使静态页面对浏览器事件做出响应。也就是说，JavaScript 是一种基于原型、多范式、单线程的动态语言，并且支持面向对象、命令式和声明式(如函数式编程)风格。

1.1.3　RESTful 架构

REST 在前端开发中通常指的是一种规则，其规则是一组架构约束条件和原则，能够满足这些约束条件和原则的应用程序或设计就是 RESTful。

RESTful 是一种基于 HTTP 的网络应用程序的设计风格和开发方式，可以使用两种格式进行定义：XML 格式和 JSON 格式。RESTful 能够实现第三方 OTT 调用移动网络资源的功能，动作类型为新增、变更、删除所调用资源。

RESTful 架构是对 MVC 架构进行改进后所形成的一种架构，通过使用事先定义好的接口与不同的服务联系起来。在 RESTful 架构中，浏览器使用 POST、DELETE、PUT 和 GET 四种请求方式分别对指定的 URL 资源进行增删改查操作。因此，RESTful 是通过 URI 实现对资源的管理及访问，具有扩展性强、结构清晰的特点。

RESTful 架构将服务器分成前端服务器和后端服务器两部分，前端服务器为用户提供无模型的视图；后端服务器为前端服务器提供接口。浏览器向前端服务器请求视图，通过视图中包含的 AJAX 函数发起接口请求来获取模型。

在项目开发中引入 RESTful 架构，有利于团队并行开发。在 RESTful 架构中，将多数 HTTP 请求转移到前端服务器上，可降低服务器的负荷，使视图获取后端模型失败也能呈现。但 RESTful 架构却不适用于所有的项目，当项目比较小时无须使用 RESTful 架构，因为它会使项目变得更加复杂。

1.1.4　大前端时代的来临

大前端时代主要就是 Web 统一的时代，即利用 HTML5 或者更高的版本去开发传统的网站，制作一些动态效果，并使用 BootStrap 架构进行手机端、智能设备等开发。大前端时代最大的特点在于开发中再也不用为一个 App 需要使用 Android 和 iOS 两种模式而忧心。

随着现在移动端各种终端设备的崛起，其已经超过了 PC 端。设备的不同必然导致开

发语言的不统一，开发越来越困难。比如制作一个游戏，需要开发 Android、iOS 等几个不同的版本，非常浪费人力、物力。大前端时代应运而生，它的出现恰恰解决了这些困难，目前各家公司都在研发利用 HTML5 开发各种需求。另外，云计算的迅猛崛起必然导致未来一切云端化，比如操作系统，各种应用程序未来都将云端化，而云端化的前端主力技术就是 Web 前端开发技术。

1.2 MVC、MVP 和 MVVM 架构模式

在 MV 系列的框架中，M 是指 Model 层，V 则是 View 层，但其功能会因为框架的不同而变化。Model 层是数据模型，用来存储数据；View 层是视图，用来展示 Model 层的数据。虽然在不同的框架中，Model 层和 View 层的内容会有所差别，但是基本功能不会有太大改变，变的只是数据的传输方式。本节主要介绍关于 MV 系列的三大架构。

1.2.1 MVC 架构模式

MVC 的全称是 Model View Controller，即模型-视图-控制器。MVC 作为三种架构中最早产生的框架，其他两个框架都是以它为基础发展而来的。

MVC 开始是存在于桌面程序中的，M 是指业务模型，V 是指用户界面，C 则是控制器。使用 MVC 的目的是将 M 和 V 的实现代码分离，从而让同一个程序可以使用不同的表现形式。比如一批统计数据可以分别用柱状图、饼图来表示。而 C 存在的目的则是确保 M 和 V 同步，一旦 M 改变，V 应该同步更新。

模型-视图-控制器(MVC)是 Xerox PARC 在 20 世纪 80 年代为编程语言 Smalltalk-80 发明的一种软件设计模式，目前已被广泛使用，后来被推荐为 Oracle 旗下 Sun 公司 Java EE 平台的设计模式，并且受到越来越多的使用 ColdFusion 和 PHP 的开发者的欢迎。模型-视图-控制器模式是一个有用的工具箱，当然也会存在一定的优点和缺点。

下面将详细地讲解 MVC 各部分的作用。

1. 模型

模型表示企业数据和业务规则。在 MVC 的三个部件中，模型拥有最多的处理任务。例如它可能用像 EJBs 和 ColdFusion Components 这样的构件对象来处理数据库。被模型返回的数据是中立的，就是说模型与数据格式无关，这样一个模型就可以为多个视图提供数据，由于应用于模型的代码只需写一次就可以被多个视图重用，因此大大减少了代码的重复性。

2. 视图

视图是用户看到并与之交互的界面。对以前的 Web 应用程序来说，视图就是由 HTML 元素组成的界面。在最新的 Web 应用程序中，HTML 依旧在视图中扮演着重要的角色，但一些新的技术已层出不穷，它们包括 Adobe Flash 和像 XHTML、XML/XSL、WML 等一些标识语言及 Web Services。

3. 控制器

控制器接受用户的输入并调用模型和视图去完成用户的需求，所以当单击 Web 页面中

的超链接和发送 HTML 表单时，控制器本身不会输出任何东西和做任何处理。它只是接受请求并决定调用哪个模型构件去处理请求，然后再确定用哪个视图来显示返回的数据。

MVC 架构如图 1-2 所示。

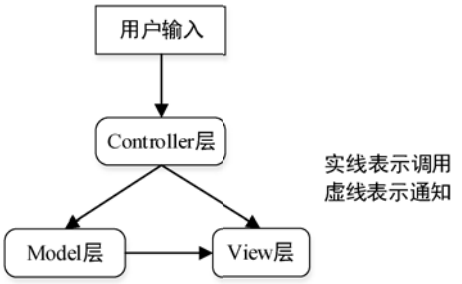


图 1-2　MVC 架构

说明

从图 1-2 中可以发现，各部分之间的通信都是单向的，呈三角形的状态。

MVC 架构流程如图 1-3 所示。

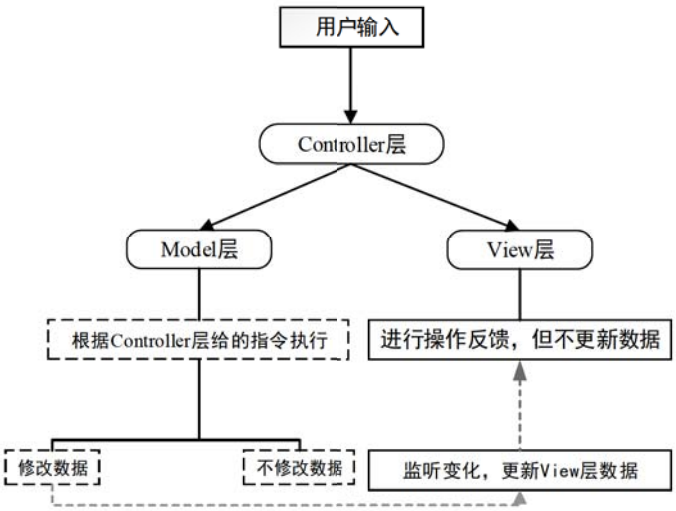


图 1-3　MVC 架构流程

从图 1-3 中可以看出，当 Controller 层触发 View 层时，View 层并不会发生改变，但是当 Controller 层触发 Model 层时，Model 层发生改变，此时，View 层通过监听 Model 层的数据变化进行更新，与 Controller 层无关。换言之，Controller 存在的目的是确保 M 和 V 的同步，一旦 M 改变，V 应该同步更新。

1.2.2　MVP 架构模式

MVP 是 Model View Presenter 的首字母的缩写，分别表示模型层、视图层、表示层，

它是 MVC 架构的一种演变。作为一种新的模式，MVP 与 MVC 有着一个重大的区别：在 MVP 中 View 并不直接使用 Model，它们之间的通信是通过 Presenter(MVC 中的 Controller) 来进行的，所有的交互都发生在 Presenter 内部，而在 MVC 中 View 会直接从 Model 中读取数据而不是通过 Controller。具体介绍如下。

Model：模型层，用于数据存储以及业务逻辑。

View：视图层，用于展示与用户实现交互的页面，通常实现数据的输入和输出功能。

Presenter：表示层，用于连接 M 层、V 层，完成 Model 层与 View 层的交互，还可以进行业务逻辑的处理。

MVP 架构如图 1-4 所示。

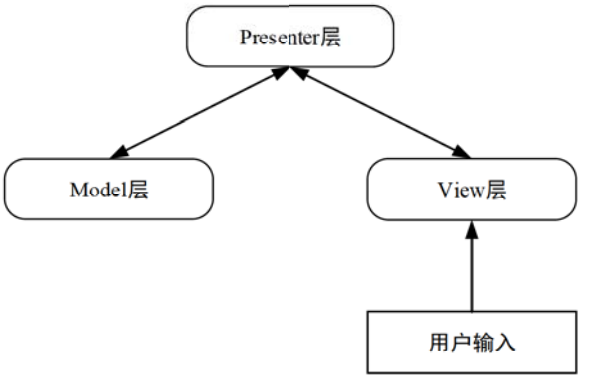


图 1-4　MVP 架构

由图 1-4 可知，MVP 架构各部分之间的通信都是双向的。但是在 MVP 架构中，View 层不会直接访问 Model 层，而是通过 Presenter 层提供的接口去访问。

MVP 架构流程如图 1-5 所示。

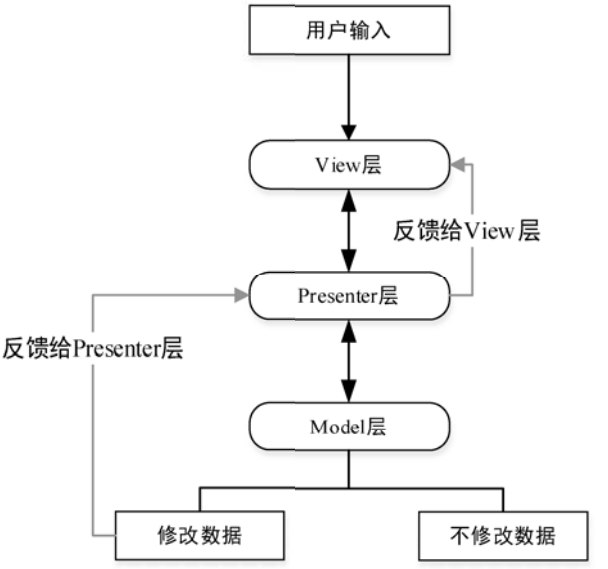


图 1-5　MVP 架构流程

从图 1-5 中可以看出，Model 层和 View 层两者是互不干涉的，而是通过 Presenter 层进行访问的。

1.2.3　MVVM 架构模式

MVVM 是 Model-View-ViewModel 的简写，它本质上就是 MVC 的改进版。MVVM 就是将其中的 View 的状态和行为抽象化，让视图 UI 和业务逻辑分开。当然这些事 ViewModel 已经帮我们做了，它可以在取出 Model 数据的同时帮忙处理 View 中由于需要展示内容而涉及的业务逻辑。微软的 WPF 带来了新的技术体验，如 Silverlight、音频、视频、3D、动画等，这导致了软件 UI 层更加细节化、可定制化。同时，在技术层面，WPF 也带来了诸如 Binding、Dependency Property、Routed Events、Command、DataTemplate、ControlTemplate 等新特性。MVVM 框架的由来便是 MVP(Model-View-Presenter)模式与 WPF 结合的应用方式而发展演变过来的一种新型架构框架。它立足于原有 MVP 框架并且把 WPF 的新特性糅合进去，以应对用户日益复杂的需求变化。当下流行的 MVVM 框架有：Vue.js、AngularJS 等。

MVVM 是一种新型的软件架构模式。MVVM 有助于将图形用户界面的开发与后端业务逻辑的开发分离开来。MVVM 的视图模型是一个值转换器，这意味着视图模型负责从模型中转换数据对象，以便轻松管理和呈现对象，视图模型实现了中介的功能。具体介绍如下。

View 层：视图层，前端开发中的 DOM 层，其作用是为用户展示各种信息。

Model 层：数据层，数据可以是我们固定的写死的数据，但更多的是来自服务器从网络上请求下来的数据。

ViewModel 层：视图模型层，是 View 层和 Model 层沟通的桥梁，一方面它实现了数据绑定(Data Binding)，将 Model 层的数据改变实时地反映到 View 层中；另一方面它实现了对文档对象模型的监听(DOM Listener)，当 DOM 发生一些事件(单击、滚动、touch 等)时，可以监听，并在需要的情况下改变对应的 Model 层的数据。

MVVM 的架构如图 1-6 所示。

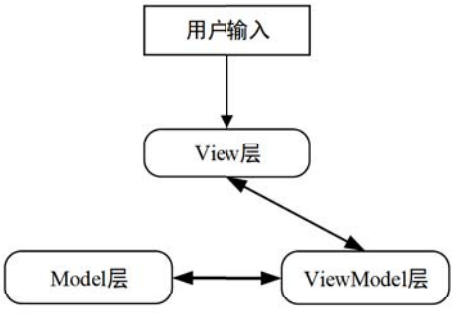


图 1-6　MVVM 架构

由图 1-6 可知，在 MVVM 架构模式中有两个方向，第一个方向是模型→视图，通过数据绑定的方式来实现；第二个方向是视图→模型，通过 DOM 事件进行监听。这种存在两个方向都实现的情况叫做数据的双向绑定，双向绑定会根据数据的变化进行实时的渲染。

MVVM 架构流程如图 1-7 所示。

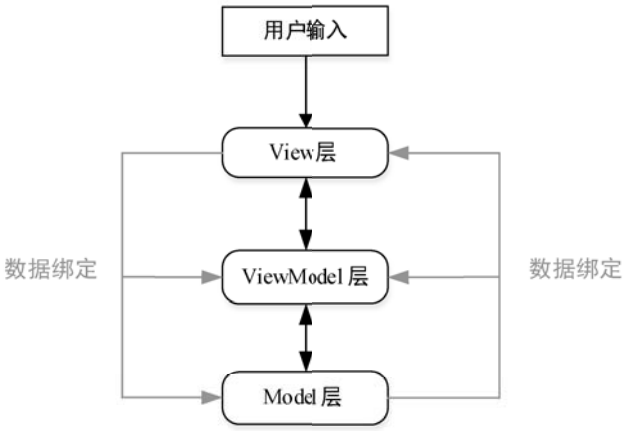


图 1-7　MVVM 架构流程

由图 1-7 可知，在 MVVM 架构模式中，三层之间的通信都是互通的。MVVM 架构与 MVP 架构有一定的相似度，两种架构模式都是通过 View 层开始触发，但是不同的是，ViewModel 双向绑定了 View 层和 Model 层，当 View 层的数据发生变化的同时系统也会修改 Model 层的数据，反之，当 Model 层的数据发生改变，View 层也会随之变化。如图 1-8 所示，View 层和 Model 层之间一旦发生修改就会同步到对方。

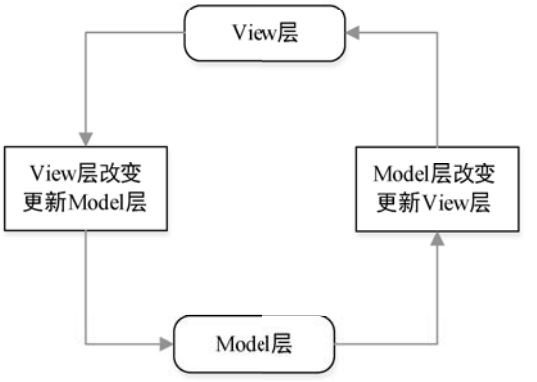


图 1-8　数据的双向绑定

1.2.4　三者的区别和优劣

MVC、MVP、MVVM 三者的主要区别就在于除 View 层和 Model 层之外的第三层，这一层的不同使得 MV 系列框架区分开来。三者的差异在于如何将 View 层和 Model 层连接起来，实现用户的交互。

1. 三种模式第三层的区别

MVC 架构模式中的 Controller 层会根据事件不同，去调用 Model 层的接口进行操作，但是当 Model 层的数据发生改变时并不经过 Controller 层，而是直接通知 View 层，View

层采用观察者的模式监听 Model 层的改变而进行变化。

在 MVP 架构模式中，Presenter 层和 Controller 层的作用一样，对 Model 层进行操作，但与之不同的是，Presenter 层会反作用于 View 层，Model 层在发生变化之后首先会通知 Presenter 层，Presenter 层再去更新 View 层。MVP 架构模式中使用 Presenter 层的目的就是完全切断 View 层和 Model 层之间的联系，由 Presenter 层充当桥梁，做到 View-Model 之间的通信完全自由。

MVVM 架构模式是在原有的 Model 基础上又添加一个 ViewModel 层，通过数据的双向绑定来实现 View 层和 Model 层的自动同步。也就是当 Model 层发生改变时不用再手动操作来改变 View 层，而是改变 Model 层的属性，该属性对应的 View 层也会随之变化。

2. 三种架构模式的优缺点

1) MVC 架构模式的优缺点

MVC 的优点是它可以为应用程序处理各种不同的视图。作为视图来讲，它只是作为一种输出数据并允许用户操纵的方式。MVC 的缺点一方面是它增加了系统结构和实现的复杂性，对于较为简单的页面，严格地遵循 MVC 架构模式，会使模型、视图与控制器分离，从而增加结构的复杂性，产生更多操作，降低运行效率。并且由于模型接口操作的不同，视图可能需要多次调用才能够获得足够的显示数据，而对于没有发生变化的数据进行频繁的访问，容易影响操作性能。目前，一般的高级界面工具或者是构造器不支持 MVC 架构，要想改造工具以适应 MVC 的代价却是非常高的。

2) MVP 模式下表示层的优势

(1) View 与 Model 完全隔离。

Model 和 View 之间具有良好解耦性的设计意味着：如果 Model 或 View 中的一方发生变化，只要交互接口不发生变化，另一方就无须对上述变化做出相应的改变，这使得 Model 层的业务逻辑具有很好的灵活性和可重用性。

(2) Presenter 与 View 的具体实现技术无关。

采用诸如 Windows 表单、WPF、Web 表单等用户界面构建技术中的任意一种来实现 View 层，都无须改变系统的其他部分。甚至为了能同时支持 B/S、C/S 部署架构，应用程序可以用同一个 Model 层来适配多种技术构建的 View 层。

(3) 可以进行 View 的模拟测试。

原来由于 View 和 Model 之间的紧耦合，在 Model 和 View 同时开发完成之前对其中一方进行测试是不可能的。由于同样的原因，对 View 或 Model 进行单元测试也很困难。现在 MVP 模式解决了所有的问题，在 MVP 模式中，View 和 Model 之间没有直接依赖，开发者能够借助模拟对象注入测试两者中的任一方。

3) MVVM 的优点

MVVM 模式和 MVC 模式一样，其主要目的是分离视图(View)和模型(Model)，以下是 MVVM 的优点。

(1) 低耦合。视图(View)可以独立于 Model 的变化和修改，一个 ViewModel 可以绑定到不同的 View 上，当 View 变化的时候 Model 可以不变，当 Model 变化的时候 View 也可以不变。

(2) 可重用性。可以将一些视图逻辑放在一个 ViewModel 里面，让很多 View 重用这段视图逻辑。

(3) 独立开发。开发人员可以专注于业务逻辑和数据的开发(ViewModel)，设计人员可以专注于页面设计，使用 Expression Blend 可以很容易设计界面并生成 XML 代码。

(4) 可测试。界面素来是比较难于测试的，而现在测试可以针对 ViewModel 来进行。

1.3 前端开发调试必备利器

在进行 Vue.js 开发之前，我们首先需要了解 Vue.js 开发中常用的一些工具。下面将通过开发者的眼——Chrome、开发者的手——VS Code 和开发者的心——Terminal 来讲解 Vue.js 开发中常用的工具。

1.3.1 开发者的眼——Chrome

Google 浏览器的英文全称为 Google Chrome，是一款由 Google 公司开发的基于其他开源软件撰写网页的浏览器，它可以大大提高开发的稳定性、安全性和速度。另外，Google Chrome 是以更强大的 JavaScript V8 引擎为基础的浏览器，这是当前 Web 浏览器所无法实现的。

Google Chrome 最大的优势就是多进程架构，能允许多个程序同时运行而互不影响，每个标签、窗口和插件都在一个独立的“沙箱”中运行，当一个标签页崩溃时，其他页面也不会受到影响，进一步提高了系统的安全性。

在进行 Web 开发的过程中，可以通过 Google Chrome 提供的开发者工具 DevTools 查看页面的各种状态，包括样式、网络请求是否成功，网络资源是否加载成功等。

因此，Chrome 就是开发者的一个观察 Web 页面的眼睛。

1.3.2 开发者的手——VS Code

对于开发者来说，选择一个合适的编辑器，编程效率也会得到提升。在选择编辑器时，需要重点考虑以下三个因素。

(1) 编辑器对代码的效率的要求。

(2) 编辑器支持的编程语言，编辑器配置是否复杂。

(3) 编辑器的插件。

VS Code(Visual Studio Code)是一个免费的、开源的、高性能的、跨平台的、轻量级的跨平台编辑器，在性能、语言支持、开源社区方面也较为稳定。Monaco Editor 被 Erich Gamma 移植到桌面平台上，成为如今的 VS Code 编辑器。VS Code 的定位就是编辑器，但又并不局限于此。

首先是完全开源开发的平台，VS Code 的源代码以 MIT 协议(开源中国)开源为基础，这意味着用户可以免费获取 VS Code 的核心代码，社区可以基于 VS Code 的代码开发自己的产品，而 VS Code 也经常能从一些知名的项目中吸取宝贵的经验。

其次，VS Code 的源代码托管在 GitHub 上，同时使用 GitHub 的开发计划和测试，使每个用户都可以在 GitHub 上了解 VS Code 的开发进度，作为用户，可以更好地了解产品的发展情况。

最后，VS Code 自带 Type Script 和 Node.js 的支持，用户下载 VS Code 后能立即获得 JavaScript 和 Node.js 的智能提示，且无须任何配置即可调试 nodejs，VS Code 还为编程工作者提供了统一的 API(即 Language Server Protocol 和 Code Debugging Protocol)，使得每一个语言都能得到更好的支持。

VS Code 是一款高效的多语言多平台开发工具，从这个意义上来说，VS Code 作为开发者的手当之无愧。

1.3.3　开发者的心——Terminal

Terminal 也称为终端，是用来启动和查看系统运行状态的命令行工具。VS Code 同样可以创建终端，在 VS Code 设计之初，就一直在思考如何让 VS Code 和终端能够更紧密地联系在一起。第一种方式就是从终端中以命令行的形式打开 VS Code；第二种方式就是允许用户从资源管理器里调出系统终端。

在使用终端之前需要了解怎样打开和创建一个集成终端，最简单的方式就是按 Ctrl + ' 组合键，一个新的终端就被创建出来，如图 1-9 所示(此操作为 VS Code 编辑器的操作)。

图 1-9　创建终端

终端被创建之后可以执行命令，在 VS Code 终端中可以执行启动命令，例如：

```
//本地启动
npm run dev
//编译
npm run build
//配置 node.js 运行环境缺失的文件 (例如 node_modules)
npm install
```

除此之外，VS Code 终端还可以执行 git 命令，例如：

```
//查看代码修改情况
git status
//将修改添加到暂存区
git add .\src\
//更新版本，把 git 上的代码拉取下来
git pull
//把代码提交到 git 创建的远程仓库中
git push
```

由此看来，Terminal 作为直接沟通开发或生产环境的工具，是开发过程绕不开的核心。

1.4 搭建编程测试环境

要想进行 Vue.js 的产品开发，开发之前首先需要进行环境的搭建和代码编辑器的选择。下面将详细讲解 Vue.js 的安装方式、node 的安装方式和支持 Vue.js 的一些开发工具的使用方法。

1.4.1 Vue.js 的安装

Vue.js 的安装有三种方式：第一种是直接下载.js 文件，通过<script>标签引入使用；第二种是使用 CDN 安装；第三种则是使用 NPM 安装。不同的安装方法存在着不同的使用方法，对项目的编写方式也不同。下面就详细地介绍每一种安装方式。

1. 使用独立版本安装

(1) 打开 Vue.js 官网，下载 Vue.js 的开发版本和生产版本，如图 1-10 所示。

图 1-10 下载 Vue.js

(2) 将下载好的 Vue.js 引入 Vue 项目中，再在 index.html 的<script></script>标签中引入，此时，Vue 会被注册为全局变量，如图 1-11 所示。

图 1-11 使用标签引入

2. 使用 CDN 安装

CDN 的全称是 Content Delivery Network，即构建在网络之上的内容分发网络，也可称为内容传送网络。在互联网上可能会产生影响数据传输速率和稳定性的因素，使得内容传

输速度过慢，并且不稳定，可以通过在网络各处加入节点服务器，构成一层基于现有互联网的智能虚拟网络，而 CDN 系统能够通过网络流量和各节点的连接、负载情况、用户距离、回应时间等综合信息，将用户的请求重新连接到距离最近的节点服务器上。这就使用户可以就近选择服务器，避免网络拥堵，提高用户访问速度。使用 CDN 安装的方法也有三种，具体方式如下。

(1) Staticfile CDN(国内)：https://cdn.staticfile.org/vue/2.2.2/vue.min.js。

(2) unpkg：https://unpkg.com/vue@2.6.14/dist/vue.min.js。

(3) cdnjs：https://cdnjs.cloudflare.com/ajax/libs/vue/2.1.8/vue.min.js。

3. 使用 NPM 安装

在使用 Vue 构建一些大型应用时，通常是使用 NPM 进行安装，NPM 可以很好地和 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。

由于 NPM 的仓库远在国外，资源传输速率会比较低且会受到限制，通常使用淘宝镜像 CNPM。下面将重点介绍使用 NPM 搭建 Vue 运行环境。

1.4.2　使用 NPM 搭建 Vue 运行环境

脚本语言通常都是需要一个解析器才能运行，例如写入 HTML 的 JS 语言，浏览器就是它的解析器角色。而对于需要独立运行的 JS 文件，Node.js 就是一个解析器的角色。

每一种解析器都可以说是一个运行环境，它不但允许 JS 定义各种数据结构并且进行各种计算，还允许 JS 使用环境提供的内置对象和方法做一些操作。NPM 是一个 Node.js 的包管理工具，在 Node.js 的安装过程下默认被安装。

下面就来介绍使用 NPM 搭建 Vue 运行环境的过程。

1. 安装 Node.js

(1) 从官网下载 Node.js。下载地址为 https://nodejs.org/en/，如图 1-12 所示。

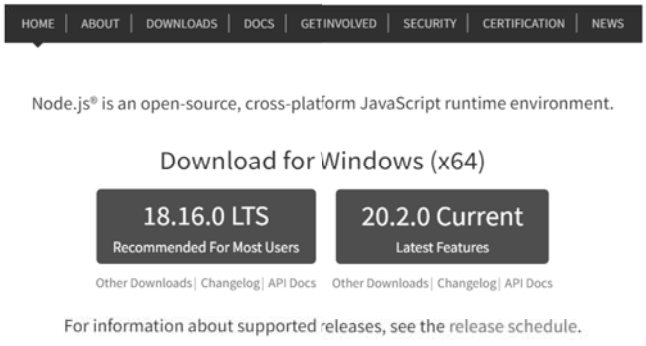


图 1-12　Node.js 的下载网站

(2) 不建议下载最新版本，此处安装的 Node 版本为 v16.20.0，如图 1-13 所示。

(3) 下载 Node.js 之后即可安装 Node.js。双击安装包，打开安装界面，如图 1-14 所示。

Previous Releases

io.js & Node.js

Releases 1.x through 3.x were called "io.js" as they were part of the io.js fork. As of Node.js 4.0.0 the former release lines of io.js converged with Node.js 0.12.x into unified Node.js releases.

Looking for latest release of a version branch?

Version	LTS	Date	V8	npm	NODE_MODULE_VERSION[1]			
Node.js 20.2.0		2023-05-16	11.3.244.8	9.6.6	115	Releases	Changelog	Docs
Node.js 19.9.0		2023-04-10	10.8.168.25	9.6.3	111	Releases	Changelog	Docs
Node.js 18.16.0	Hydrogen	2023-04-12	10.2.154.26	9.5.1	108	Releases	Changelog	Docs
Node.js 17.9.1		2022-06-01	9.6.180.15	8.11.0	102	Releases	Changelog	Docs
Node.js 16.20.0	Gallium	2023-03-28	9.4.146.26	8.19.4	93	Releases	Changelog	Docs
Node.js 15.14.0		2021-04-06	8.6.395.17	7.7.6	88	Releases	Changelog	Docs

图 1-13　选择 Node.js 的安装版本

图 1-14　Node.js 的安装欢迎界面

(4) 单击 Next 按钮后，打开用户协议许可界面，如图 1-15 所示。

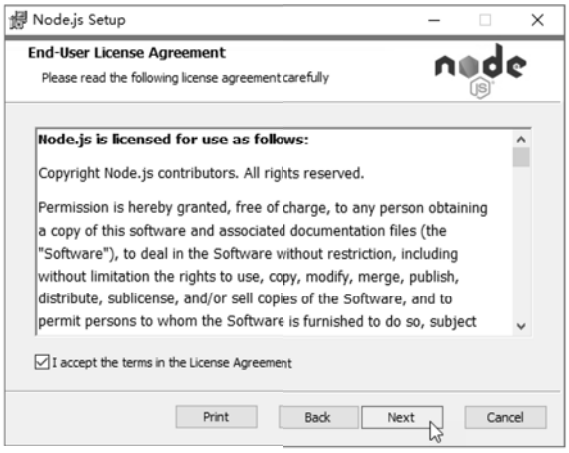


图 1-15　接受 Node.js 的安装许可协议

(5) 接受用户许可协议之后，单击 Next 按钮，然后在打开的界面中选择 Node.js 的安装目录(建议使用默认目录)，如图 1-16 所示。

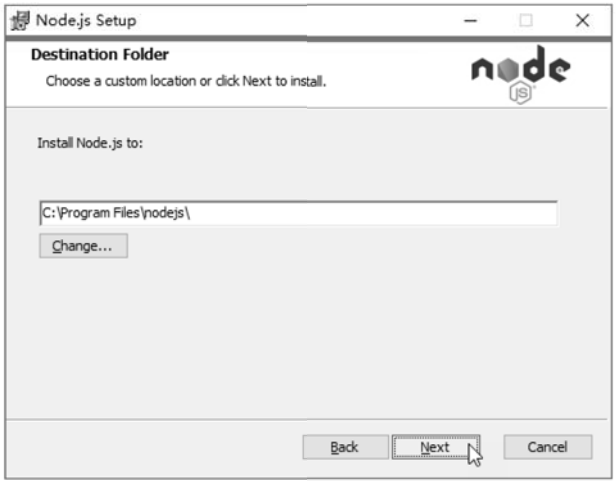


图 1-16　选择安装目录

(6) 在图 1-16 中单击 Next 按钮，然后在打开的界面中选择需要的功能进行安装，如图 1-17 所示。

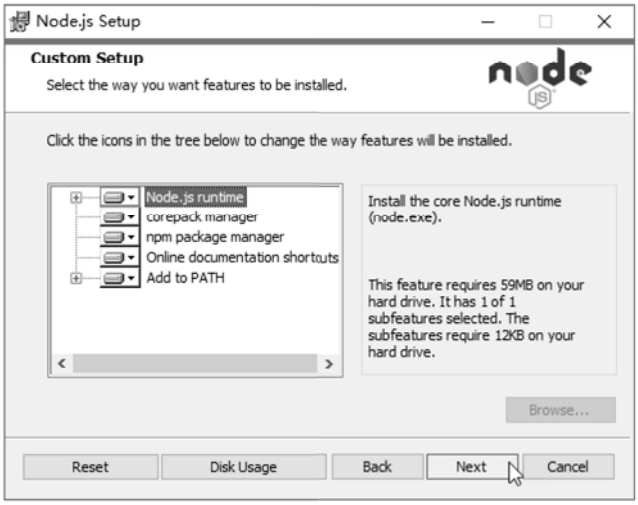


图 1-17　选择需要的功能模块

(7) 在图 1-17 中单击 Next 按钮，然后在打开的界面中选择安装工具(可根据个人需求选择)，如图 1-18 所示。

(8) 在图 1-18 中单击 Next 按钮，进入功能安装界面，如图 1-19 所示。

(9) 在图 1-19 中单击 Install 按钮，进入如图 1-20 所示的界面。

(10) 单击图 1-20 中的 Finish 按钮，完成 Node.js 的安装。

接下来我们验证 Node.js 是否安装成功。

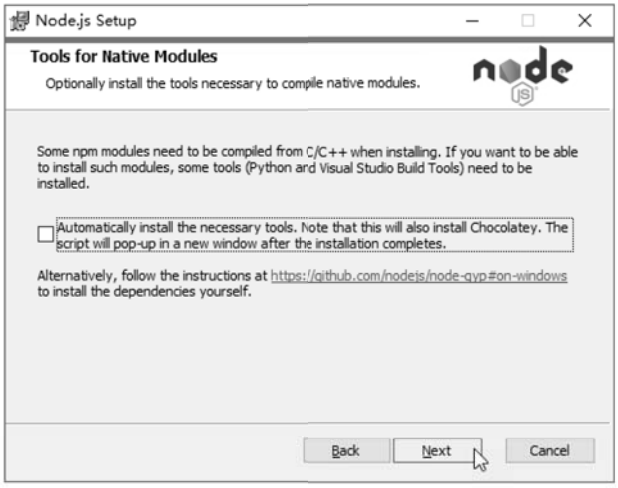


图 1-18　选择安装工具

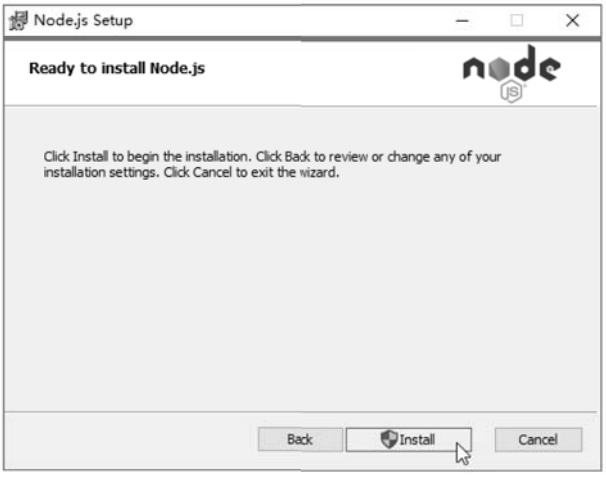


图 1-19　功能安装界面

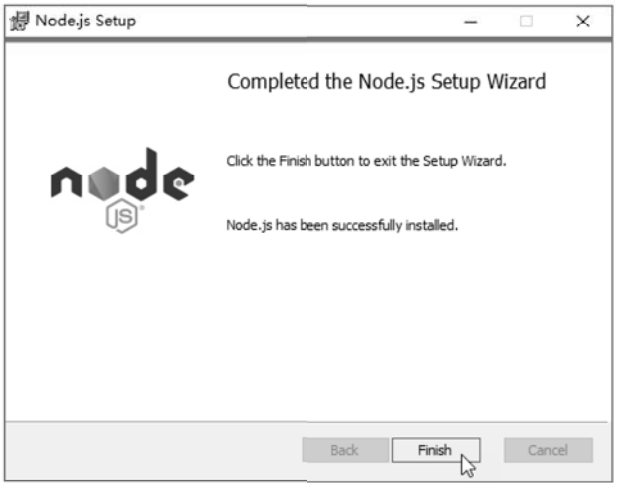


图 1-20　完成安装

2. 验证安装 Node.js

(1) 按 Win+R 组合键，输入“cmd”，然后按 Enter 键，将打开命令行界面，如图 1-21 所示。

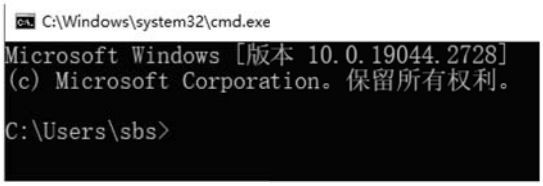


图 1-21　命令行界面

(2) 进入命令提示符窗口后，分别输入以下指令，如果显示版本号，则表示安装成功，如图 1-22 所示。

```
//查看 node 版本
node -v
//查看 npm 版本
npm -v
```

图 1-22　查看版本号

从图 1-22 中可以看出，Node.js 的版本号为 v16.20.0，NPM 的版本号为 9.6.5。输出版本号即表示安装成功，否则安装失败。

1.4.3　项目开发工具

Vue 开发工具有很多，比如 Vue CLI、Vue Press、Vuex、VS Code、WebStorm 等，下面将介绍四种 Vue 开发工具。

1. 命令行

Vue.js 提供一个官方的命令行工具，可以快速地搭建大型单页应用。具体步骤如下。

(1) 全局安装 vue-cli，在命令行工具中输入下面的命令安装 vue-cli 并查看版本，如图 1-23 所示。

```
//安装 vue-cli
npm install -g @vue/cli
//查看安装的 vue-cli 版本
vue -version
```

图 1-23　全局安装 vue-cli 并查看版本

从图 1-23 的输出结果可知，安装的 vue-cli 版本为 5.0.8。

(2) 输入以下命令创建 Vue 3 项目，将项目命名为 vue3.0-test，输入命令之后按 Enter 键即可。结果如图 1-24 所示。

```
//创建 Vue 项目
vue create vue3.0-test
```

图 1-24　创建 Vue 3 项目

(3) 按键盘上的↑和↓方向键选择安装方式，此处选择的是 Default 方式。运行结果如图 1-25 所示。

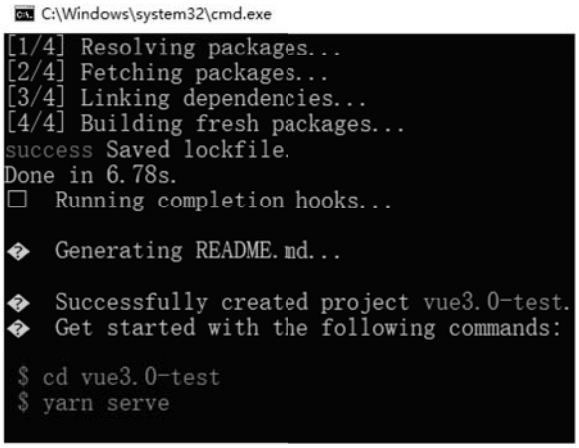


图 1-25　运行结果

(4) 输入以下命令，进入项目并运行，如图 1-26 所示。

```
//进入项目目录
cd vue3.0-test
//运行项目
npm run serve
```

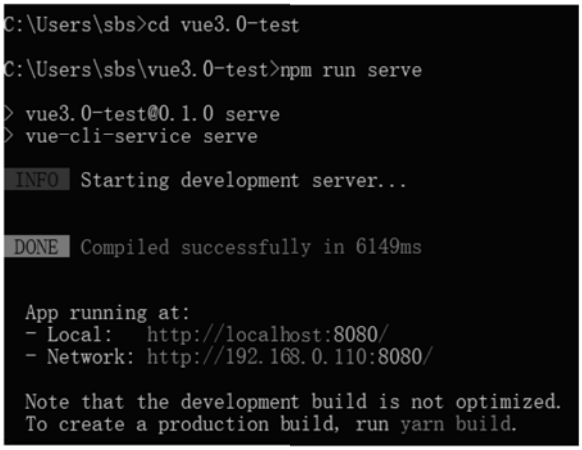


图 1-26　运行项目

(5) 成功执行以上命令后访问 http://localhost:8080/，输出结果如图 1-27 所示。

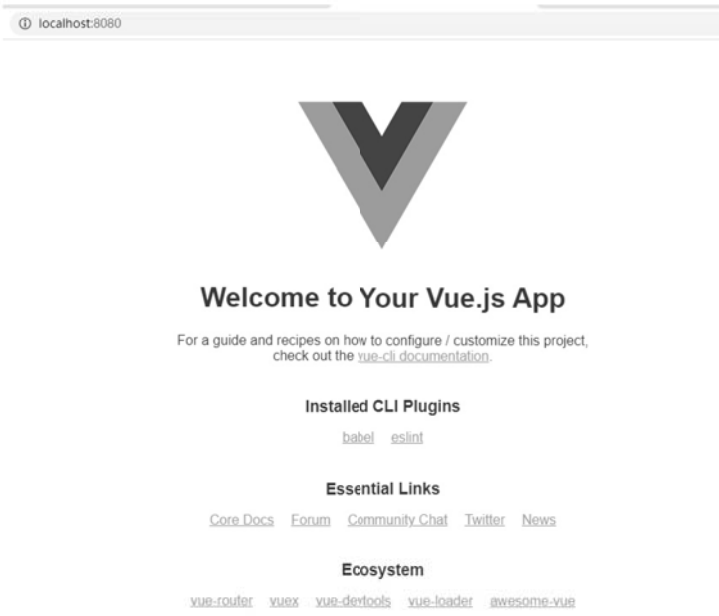


图 1-27　输出结果

2. Vite

Vite 是一个 Web 开发的构建工具，它可以用其原生的 ES 模块导入方式，实现闪电般的冷服务器启动。使用 Vite 创建 Vue 项目的具体步骤如下。

(1) 通过终端运行以下命令，快速使用 Vite 工具构建 Vue 项目，并将项目命名为

“vue3.0-test2”，如图 1-28 所示。

```
//创建 Vue 项目
npm init vite-app vue3.0-test2
```

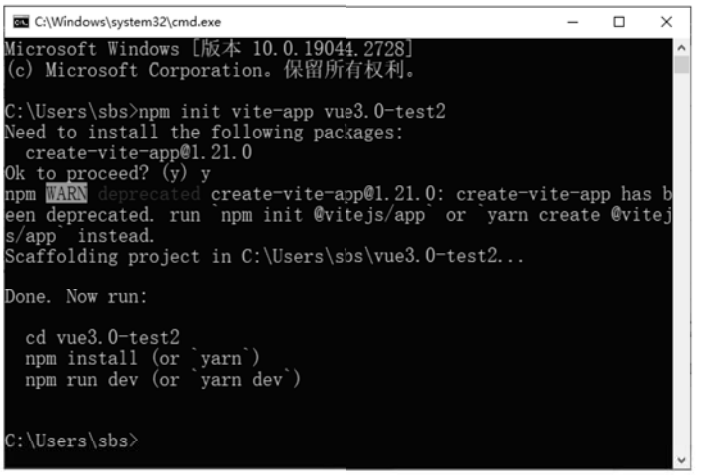


图 1-28　使用 Vite 构建项目

(2) 输入以下命令，进入项目，安装依赖，并运行项目，如图 1-29 所示。

```
//进入项目目录
cd vue3.0-test2
//安装依赖
npm install
//运行项目
npm run dev
```

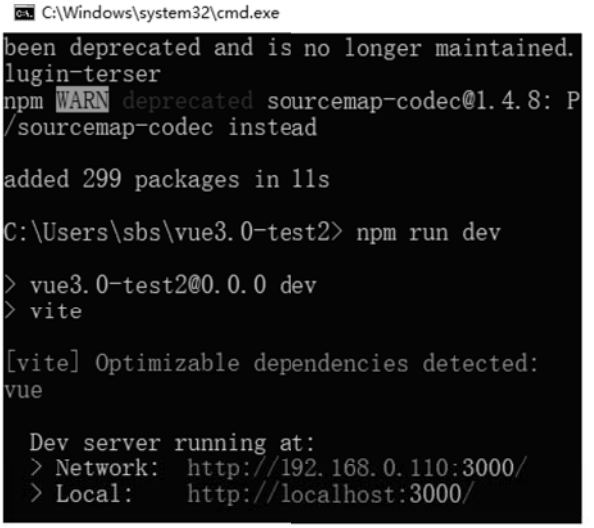


图 1-29　安装并运行项目

(3) 成功执行以上命令之后访问 http://localhost:3000/，输出结果如图 1-30 所示。

图 1-30　输出结果

3. VS Code

使用 VS Code 创建 Vue 项目和前两种方式输出的指令是一样的，只不过它是在 VS Code 终端输入命令，进行项目的初始化。具体步骤如下。

(1) 打开 VS Code，在终端输入以下指令，创建项目 vue3.0-test3，如图 1-31 所示。

```
//创建 Vue 项目
npm init vite-app vue3.0-test3
```

图 1-31　使用 VS Code 创建 Vue 项目

(2) 输入以下命令，进入项目，安装依赖，并运行项目，如图 1-32 所示。

```
//进入项目目录
cd vue3.0-test3
//安装依赖
npm install
//运行项目
npm run dev
```

(3) 成功执行以上命令之后访问 http://localhost:8080/，输出结果如图 1-33 所示。

图 1-32　安装依赖并运行项目

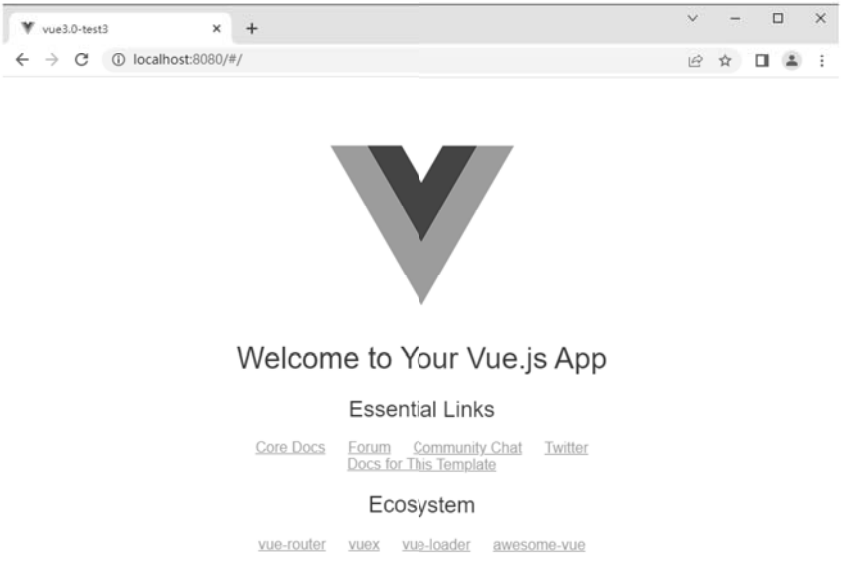


图 1-33　输出结果

4. HbuilderX

HbuilderX 是一款前端开发者服务的通用 IDE，它可以用来开发 DCloud 出品的 uni-app 项目、Wap2App 项目等。使用 HbuilderX 创建 Vue 项目的具体步骤如下。

(1) 打开 HbuilderX 编辑器，选择左上角的“文件”→“新建”→“项目”命令，新建 Vue 项目，如图 1-34 所示。

(2) 选择新建的普通项目，模板为 Vue 项目(3.2.8)，将项目命名为 vue3.0-test4，如图 1-35 所示。

(3) 项目创建完成后，在终端中运行项目，如图 1-36 所示。

图 1-34　新建 Vue 项目

图 1-35　选择 Vue 模板

图 1-36　在终端中运行项目

(4) 运行成功后，结果如图 1-37 所示。

```
终端-外部命令

vite v2.5.3 dev server running at:

> Local: http://localhost:3000/
> Network: use `--host` to expose

ready in 698ms.
```

图 1-37　运行结果

(5) 访问 http://localhost:3000/，输出结果如图 1-38 所示。

图 1-38　项目运行结果

1.4.4　源码管理机制

创建 Vue 项目的时候，需要明白一个 Vue 项目中包含什么，各个部分又分别能做些什么。在上一小节中我们使用 VS Code 创建了一个 Vue 项目，打开该项目，其目录结构如图 1-39 所示。

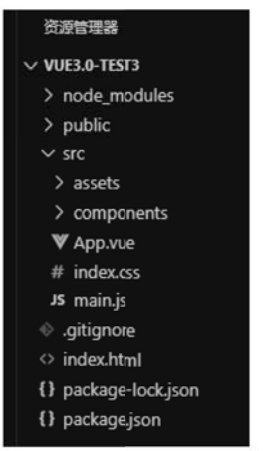


图 1-39　Vue 项目目录结构

部分目录/文件介绍如表 1-1 所示。

表 1-1　Vue 项目目录解析

目录/文件	说　明
node_modules	NPM 加载的项目依赖模块
public	公共资源目录
src	主要是源码目录。 assets：放置一些图片，如 Logo 等。 components：目录里面放了一个组件文件，可以不用。 App.vue：项目入口文件，也可以直接将组件写在这里，而不使用 components 目录。 index.css：样式文件。 main.js：项目的核心文件
.xxxx 文件	是一些配置文件，包括语法配置、git 配置等
index.html	首页入口文件，可以添加一些 meta 信息或统计代码
package.json	项目配置文件

1.5　代码调试方法

在日常开发中，可能会遇到很多千奇百怪的问题，这些问题可能是代码版本冲突造成的，也有可能是我们在编码时粗心造成的。在遇到这些问题时只凭肉眼是很难发现的，那么此时就需要使用代码调试功能来查找这些问题的原因了。下面将通过 Console 工具和调试工具来讲解代码的调试。

1.5.1　使用 Console 工具

Console 也被称为控制台，Console 是增强 Windows 控制台的窗口，在 JS 开发中起着非常重要的作用。它包括多个标签、文本编辑器、不同类型的背景，其主要作用就是用来显示在网页加载过程中产生的信息，可以查看错误信息、打印调试信息、调试 JS 代码，还可以当作 JavaScript API 查看。

Console 对象中主要使用的方法有四个，分别为：console.log()、console.info()、console.warn()和 console.error()。

1. console.log()

console.log 的主要作用是在浏览器控制台打印信息，它主要输出的是普通信息和日志信息。在前端开发中，它常被用来调试和分析代码，如果在 JS 代码中调用 console.log()，就可以在 Web 浏览器的控制台打印常量、变量、数组、对象、表达式等值。通过以上方式还可以进行单个变量(表达式)、多个变量以及换行输出。

在源码中定义一个 console.log()函数，并使输出值为“Hello.vue.3.0”，在浏览器中打开控制台界面就会打印出所要输出的内容，如图 1-40 所示。

图 1-40　使用 console.log()函数打印信息

2. console.info()

console.info()是日常开发中用得最多的一种方法，通常会用它来打印某个对象或变量。console.info()方法的运用场景非常广泛，它也是最广泛的输出模式。

在源码中定义一个 console.info()函数，将输出值设为“Hello, vue3.0”，在浏览器中打开控制台界面就会打印出所要输出的内容，如图 1-41 所示。

图 1-41　使用 console.info()函数打印信息

3. console.warn()

console.warn()的输出与 console.log()的输出基本没有什么区别，但是该条打印信息是属于警告级别而不是普通信息级别，最明显的是它的左侧会有一个警告图标，并且背景色和文字颜色也会不一样。

在源码中定义一个 console.warn()函数，将输出值设为“Hello.vue.3.0”，在浏览器中打开控制台界面就会打印出所要输出的内容，如图 1-42 所示。

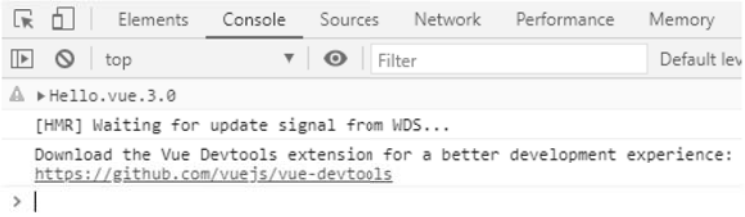


图 1-42　使用 console.warn()函数打印信息

4. console.error()

console.error()的输出与 console.log()的输出类似，只是 console.error()输出的信息属于报错级别，背景和文字颜色为红色，它主要是打印一条错误信息。

在源码中定义一个 console.error()函数，将输出值设为“Hello, vue3.0”，在浏览器中打开控制台界面就会打印出所要输出的内容，如图 1-43 所示。

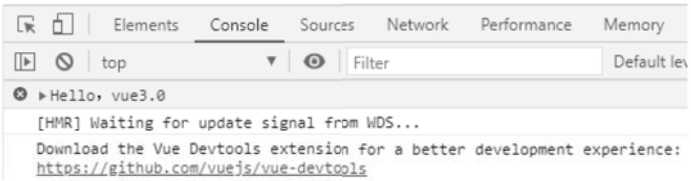


图 1-43　使用 console.error()打印信息

1.5.2　使用调试工具

vue-devtools 是一款基于 chrome 浏览器的插件，用于调试 Vue 应用，它可以极大地提高调试效率。Vue 官方提供的 vue-devtools 调试工具能够方便开发者对 Vue 项目进行调试与开发。下面来介绍 devtools 的安装。

(1) 在 github 上下载压缩包。github 的下载地址为 https://github.com/vuejs/devtools.git，如图 1-44 所示。

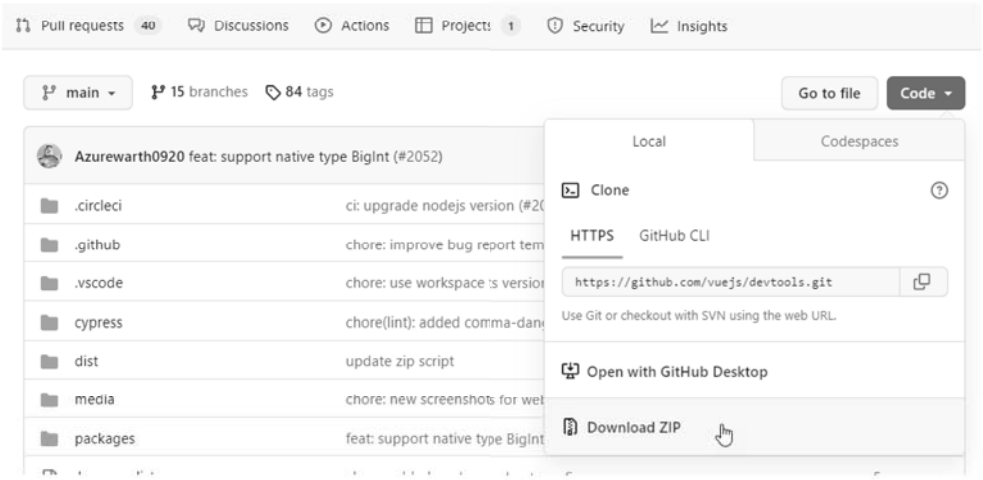


图 1-44　下载 devtools 压缩包

(2) 下载完成后将文件解压，文件解压完成后按 Win+R 组合键，输入“cmd”，打开命令行窗口，进入 devtools-main 文件夹，如图 1-45 所示。

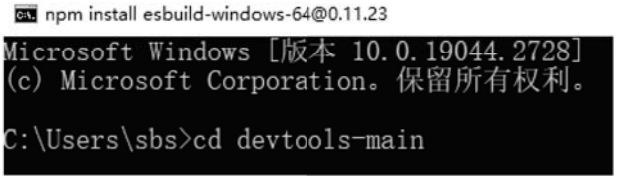


图 1-45　打开命令行窗口

(3) 在命令行窗口中分别输入以下代码，对 devtools-main 进行打包。运行结果如图 1-46 所示。

```
//安装依赖
yarn install
```

```
//打包
yarn run build:watch
//运行
yarn run dev:chrome
```

图 1-46　运行结果

出现图 1-46 所示界面时，即表示打包运行完成了，按 Ctrl+C 组合键退出即可。

(4) 打开 Chrome 浏览器的扩展程序，单击【加载已解压的扩展程序】按钮，如图 1-47 所示。

图 1-47　打开 Chrome 浏览器的扩展程序

(5) 选择 devtools-main\packages\shell-chrome 中的文件，如图 1-48 所示。

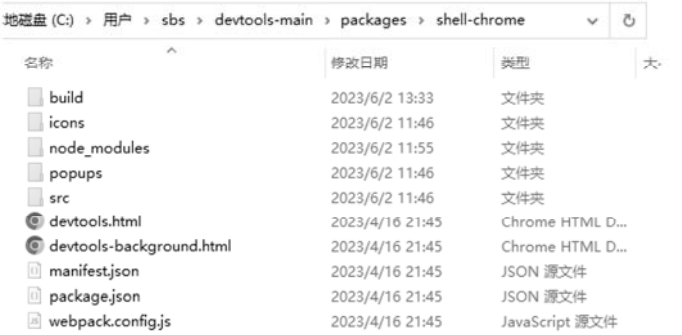


图 1-48　选择文件

(6) 将 shell-chrome 中的文件添加到 Chrome 浏览器的扩展中，如图 1-49 所示。

图 1-49　添加浏览器的扩展

至此，Vue 的扩展就安装完成了。在打开 Vue 项目时，就可以使用 vue-devtools 进行调试了，如图 1-50 所示。

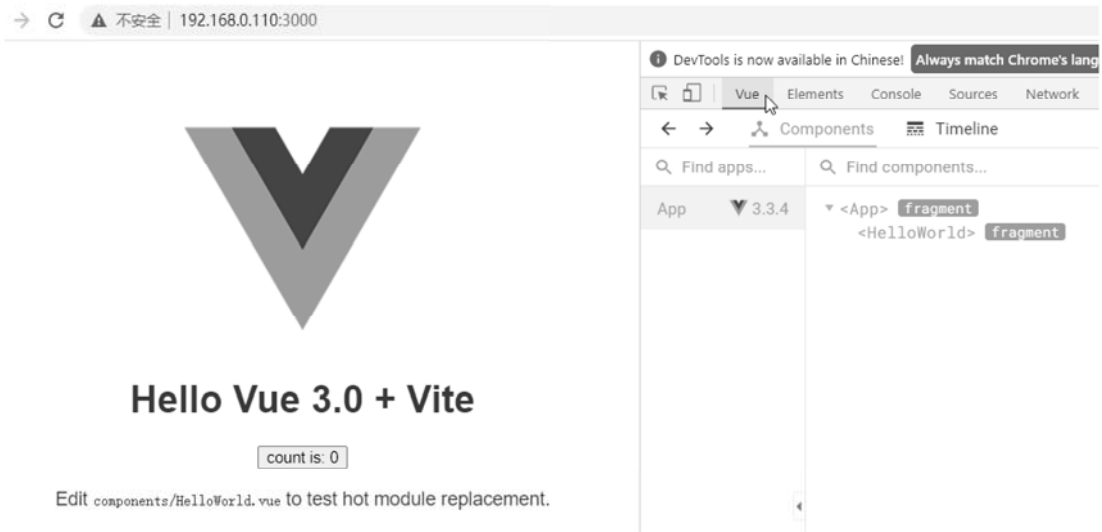


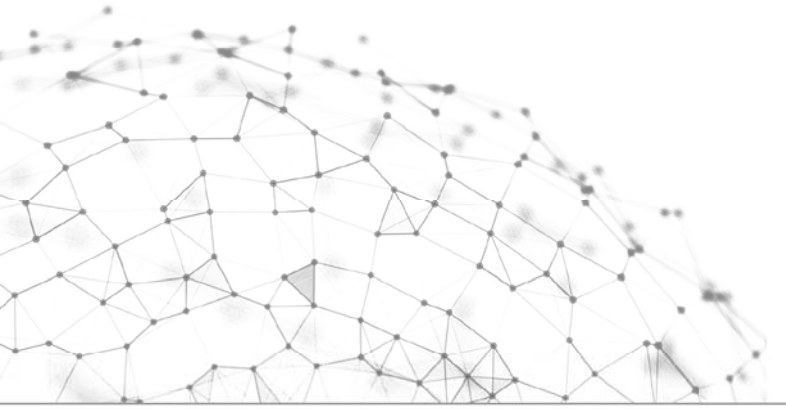
图 1-50　调试工具界面

1.6　本 章 小 结

本章主要讲解了一些有关 Vue.js 的基础知识，包括在学习 Vue.js 前应该了解什么知识，也回顾了前端的发展历程以及三大主流框架和前端开发所使用的工具，包括开发工具和调试工具。当然，对于前端开发来说，仅仅了解这些还远远不够。当今社会，正处于一个大前端时代，随着移动业务的拓展，手机端也越来越重要，小程序以及手机上的宣传页大多是采用网页的方式来制作的，方便快捷并且无须使用客户端，这就更加彰显出前端的重要性了。

随着时代的进步，对架构模式也有一定的改良，从最初的 MVC 架构模式实现数据的单向传输，到 MVP 架构模式实现数据的双向传输，再到今天 Vue.js 所使用的 MVVM 架构模式，实现了数据的双向绑定。

对于开发工具来说，本章提供了两种工具进行 Vue 项目的开发，除此之外，还有官方给出的两种方法可以创建 Vue 项目，读者根据自己的需要进行选择即可。



第 2 章

熟练使用 Vue 对象、组件与库

【本章概述】

在第 1 章主要介绍了 Vue 的组成、安装以及在开发一个 Vue 项目之前需要做的准备工作，本章主要介绍关于 Vue.js 指令的知识，为之后开发 Vue 项目做铺垫。通过本章的学习，读者可以了解 Vue.js 的挂载、操作关联数据、组件基础、现有组件以及现有库。

【知识导读】

本章要点(已掌握的在方框中打钩)

- ☐ Vue.js 的挂载
- ☐ Vue.js 的操作关联数据
- ☐ 处理生命周期
- ☐ Vue 组件基础
- ☐ Vue 现有组件、现有库

2.1 挂载 Vue 对象

在 Vue 2.0 中，Vue.js 的构造函数中有一个 el 选项，该选项的作用是为 Vue 实例提供挂载元素。定义挂载元素之后，接下来的操作均可以在该元素内部进行，元素的外部，则不会受到影响。例如，在页面中定义一个 div：

```
<div id= "app"></div>
```

如果将该元素作为 Vue 实例的挂载元素，可以设置为 el:'#app'、el: ='.app'或者 el:'div'。而在 Vue 3.0.0 中通常都是使用 createApp()函数来创建应用，其语法格式如下：

```
const app = Vue.createApp({})
```

在上述代码中，传递给 createApp 的选项是用来配置根组件的，在使用 mount()应用时，该组件起着渲染的作用，例如：

```
Vue.createApp(HelloVue3.0.0App).mount( '#Hello-Vue3.0.0')
```

在上述代码中，createApp 的根组件是 HelloVue3.0.0App，应用时，该组件起着渲染的作用。并且使用 mount('#Hello-Vue3.0.0')将 Vue 应用 HelloVue3.0.0App 挂载到 中。

【例 2-1】createApp()函数的用法(实例文件 chapter-02\demo-01\index1.html)。

```
<!—createApp()函数的用法 -->
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>index1</title>
<!-- 引用 Vue.js 文件 -->
<script src="https://cdn.staticfile.org/vue/3.2.36/vue.global.min.js"></script>
</head>
<body>
<div id="vue">
  {{ message }}
</div>
<script>
const HelloVueApp = {
  data() {
    return {
      message: 'Hello Vue3.0!'
    }
  }
}
Vue.createApp(HelloVueApp).mount('#vue')
</script>
</body>
</html>
```

在浏览器中的运行结果如图 2-1 所示。

其中，{{}}用于输出对象属性和函数返回值，{{message}}对应应用中的 message 的值。

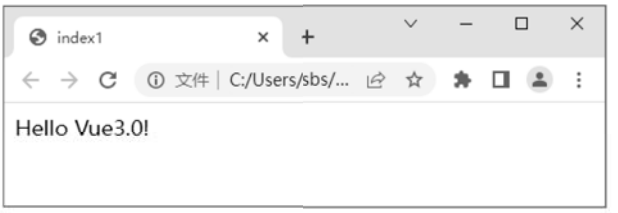


图 2-1　createApp()函数用法实例运行结果

2.2　操作关联数据

Vue 2.0 是通过 new View({})来声明实例的，而 Vue 3.0 是通过 Vue.createApp(App)来创建实例的，Vue 3.0 引入 createApp()函数的目的就是为了解决 Vue 2.0 全局配置代理的一些问题。应用程序的实例就是 MVVM 架构模式中的 ViewModel。createApp()是全局 API，它接受一个根组件选项对象作为参数，该对象可以包含数据、方法、组件生命周期钩子等，然后返回应用程序实例本身。

创建应用实例之后，可以调用实例的 mount()方法，指定一个 DOM 元素，在 DOM 元素上挂载应用程序的根组件，这样 Vue 框架就会监听 DOM 元素中的数据变化。

2.2.1　data 成员

在 Vue 组件中，data 选项是一个函数，Vue 在创建新的组件时，在实例过程中会调用此函数。它返回一个对象，然后 Vue 通过响应性系统将其包裹起来，将数据以$data 的形式存储在组件实例中。

【例 2-2】data 成员的用法(实例文件 chapter-02\demo-02\index1.html)。

```
<!-- data 成员的用法 -->
<!DOCTYPE html>
<html>
<head>
   <meta charset="utf-8">
   <title>index1</title>
   <!-- 引入 Vue.js 文件 -->
   <script src="https://cdn.staticfile.org/vue/3.2.36/vue.global.min.js"></script>
</head>
<body>
   <div id="app"></div>
   <script>
      const app = Vue.createApp({
         data() {
            return { count: 1 }
         }
      })
      const mm = app.mount('#app')
      document.write(mm.$data.count)
      document.write("<br>")
      document.write(mm.count)
      document.write("<br>")
      // 修改 mm.count 的值也会更新 $data.count
```

```
        mm.count = 2
        document.write(mm.$data.count)
        document.write("<br>")
        // 反之亦然
        mm.$data.count = 3
        document.write(mm.count)
    </script>
</body>
</html>
```

在浏览器中的运行结果如图 2-2 所示。

图 2-2　data 成员用法实例运行结果

在上述代码中，当修改 mm.count 的值时，$data.count 也会更新；反之，在$data.count 进行变化的同时，mm.count 也同样进行更新。

2.2.2　computed 成员

computed 是 Vue 的计算属性，类似于一个过滤器，对绑定到 View 的数据进行处理，它根据依赖关系进行缓存计算，只有在相关依赖发生变化的时候才会进行更新操作。当 computed 作为 Vue 的计算属性时，每一个计算属性都将被保存起来，当计算属性所依赖的属性发生变化，计算属性将会自动重新执行，并进行视图更新。

【例 2-3】computed 成员的用法(实例文件 chapter-02\demo-02\index2.html)。

```
<!-- computed 成员的用法 -->
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>index2</title>
    <!-- 引入 Vue.js 文件 -->
    <script src="https://cdn.staticfile.org/vue/3.0.5/vue.global.js"></script>
</head>
<body>
    <div id="app">
        <p>原始字符串: {{ message }}</p>
        <p>反转后的字符串: {{ reversal }}</p>
    </div>
    <script>
        const app = {
            data() {
                return {
                    message: '你好!'
                }
            },
```

```
        computed: {
            // 计算属性的 getter
            reversal: function () {
                // 返回值
                return this.message.split('').reverse().join('')
            }
        }
    }
    Vue.createApp(app).mount('#app')
</script>
</body>
</html>
```

在浏览器中的运行结果如图 2-3 所示。

图 2-3　computed 成员用法实例运行结果

2.2.3　methods 成员

在 Vue 项目的开发中可以使用 methods 选项向 Vue 组件中添加方法，该选项包括所需方法的对象。

【例 2-4】methods 成员的用法(实例文件 chapter-02\demo-02\index3.html)。

```
<!-- methods 成员的用法 -->
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>index3</title>
    <!-- 引入 Vue.js 文件 -->
    <script src="https://cdn.staticfile.org/vue/3.2.36/vue.global.min.js"></script>
</head>
<body>
    <div id="app"></div>
    <script>
        const app = Vue.createApp({
            data() {
                return { count: 1 }
            },
            methods: {
                increment() {
                    // 指向该组件实例
                    this.count++
                }
            }
        })
        const mm = app.mount('#app')
        // 结果为 1
```

```
        document.write(mm.count)
        document.write("<br>")
        mm.increment()
        // 结果为 2
        document.write(mm.count)
    </script>
</body>
</html>
```

在浏览器中的运行结果如图 2-4 所示。

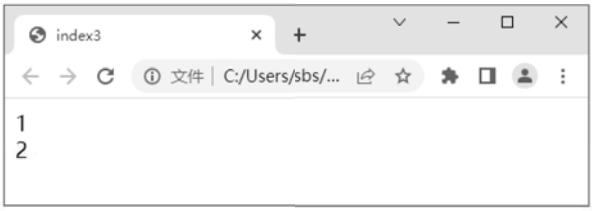


图 2-4　methods 成员用法实例运行结果

2.2.4　watch 成员

watch 函数是一种用来侦听特定数据源的函数，当被监视的属性发生变化时，回调函数中自动调用，进行操作。watch 的监视属性，就是观察监视属性是否发生变化，一旦属性发生变化，则进行相关操作；如果没有发生变化，监视属性也不会发生变化。

Vue 3.0 中的 watch 属性与 Vue 2.0 中的基本一致。不同的是 Vue 2.0 中的 watch 属性是作为一个配置项来使用，而在 Vue 3.0 中是作为函数来使用的，在 Vue 3.0 中 watch 属性可以继续使用 Vue 2.0 的语法，但是不写成配置项的方式，而是组合式 API。

在 Vue 3.0 中，watch 属性以组合式 API 的形式出现在 setup()中。下面来讲解 watch 属性在 Vue 3.0 中的基本使用。

【例 2-5】watch 成员的用法(实例文件 chapter-02\demo-02\index4.html)。

```
<!-- watch 成员的用法 -->
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>index4</title>
    <!-- 引入 Vue.js 文件 -->
    <script src="https://cdn.staticfile.org/vue/3.2.36/vue.global.min.js"></script>
</head>
<body>
    <div id="conversion">
        米 : <input type="text" v-model="meters"
        @focus="currentlyActiveField = 'meters'">
        厘米 : <input type="text" v-model="centimetre"
        @focus="currentlyActiveField = 'centimetre'">
    </div>
    <p id="info"></p>
    <script>
        const conversion = {
            data() {
                return {
```

```
                // 厘米
                centimetre: 0,
                // 米
                meters: 0
            }
        },
        watch: {
            centimetre: function (newValue, oldValue) {
                // 判断是否为当前输入框
                if (this.currentlyActiveField === 'centimetre') {
                    this.centimetre = newValue;
                    // 将厘米转换为米
                    this.meters = newValue / 100
                }
            },
            meters: function (newValue, oldValue) {
                // 判断是否为当前输入框
                if (this.currentlyActiveField === 'meters') {
                    // 将米转化为厘米
                    this.centimetre = newValue * 100;
                    this.meters = newValue;
                }
            }
        }
    }
    vm = Vue.createApp(conversion).mount('#conversion')
    // watch 方法
    vm.$watch('centimetre', function (newValue, oldValue) {
        // 这个回调将在 vm.centimetre 改变后调用
        document.getElementById("info").innerHTML = "修改前的数值为: " +
        oldValue + ", 修改后的数值为: " + newValue;
    })
  </script>
</body>
</html>
```

在浏览器中的运行结果如图 2-5 所示。

图 2-5　watch 成员用法实例运行结果

2.3　处理生命周期

一个实例的生命周期是 Vue 实例从创建、运行到销毁的整个过程。在 Vue 实例的创建、运行、销毁期间，总是伴随着各种各样的事件，这些事件统称为生命周期。在 Vue 2.0 中，生命周期函数总共有 8 个，如表 2-1 所示。

表 2-1　Vue 2.0 中生命周期函数

函　数	说　明
beforeCreate	实例初创建，初始化环境事件和生命周期钩子函数
created	实例已经被创建完成，开始数据注入和监测，并初始化 data 和 method
beforeMount	挂载之前，模板已经编译，但还没关联到页面
mounted	挂载完成，模板已经关联到页面上
beforeUpdate	DOM 更新之前
updated	DOM 更新完成
beforeDestroy	实例销毁之前
Destroyed	实例已经销毁

Vue 3.0 生命周期做了一些改变，下面总结一下 Vue 3.0 与 Vue 2.0 的生命周期函数的不同之处，如表 2-2 所示。

表 2-2　Vue 3.0 与 Vue 2.0 生命周期函数对比

Vue 2.0	Vue 3.0
beforeCreate	setup()：开始创建组件之前，创建的是 data 和 method
created	setup()
beforeMount	onBeforeMount：组件挂载到节点之前执行的函数
mounted	onMounted：组件挂载完成之后执行的函数
beforeUpdate	onBeforeUpdate：组件更新之前执行的函数
updated	OnUpdated：组件更新完成之后执行的函数
beforeDestroy	onBeforeUnmount：组件卸载之前执行的函数
Destroyed	onUnmounted：组件卸载执行的函数

通过表 2-2 可知，Vue 3.0 提供了组合式 API 形式的生命周期钩子，可以通过在生命周期钩子前面加上“on”来访问。在 Vue 3.0 中也增加了 setup()函数，由于 setup()函数是围绕 beforeCreate 和 created 生命周期钩子运行的，因此在这些生命周期钩子中的任何代码都应该直接在 setup()函数中编写。这些函数接受一个回调函数，当钩子被组件调用时将会执行。下面通过代码来演示以组合式 API 形式写的生命周期钩子。

【例 2-6】生命周期钩子的使用(实例文件 chapter-02\demo-03\test1)。

具体实现步骤如下。

(1) 使用 VS Code 创建一个 Vue 项目，具体创建步骤可参考 1.4.3 小节。

(2) 修改 HelloWorld.vue 文件中的代码。修改后的代码如下：

```
<template>
  <div>
    <h1>我是 HelloWorld 组件</h1>
    <h3>当前的值是：{{sum}}</h3>
    <!-- 修改数据触发更新阶段的生命周期钩子 -->
    <button @click="sum++">单击加 1</button>
  </div>
```

```
</template>
<!-- 引入 api -->
<script>
import {ref,onBeforeMount,onMounted,onBeforeUpdate,onUpdated,
onBeforeUnmount,onUnmounted} from 'vue'
export default {
  setup() {
    let sum = ref(0);
    //通过组合式 API 的形式去使用生命周期钩子
    onBeforeMount(() => {
      console.log("---onBeforeMount---");
    });
    onMounted(() => {
      console.log("---onMounted---");
    });
    onBeforeUpdate(() -> {
      console.log("---onBeforeUpdate---");
    });
    onUpdated(() => {
      console.log("---onUpdated---");
    });
    onBeforeUnmount(() => {
      console.log("---onBeforeUnmount---");
    });
    onUnmounted(() => {
      console.log("---onUnmounted---");
    });
    return {
      sum,
    };
  },
};
</script>
```

(3) 在 App.vue 中使用一个 HelloWorld 组件，并添加一个 v-if 判断具体演示组件是否被卸载，具体代码如下：

```
<template>
  <div>
    <button @click="isShow = !isShow ">单击是否显示组件</button>
    <!-- if 对应的表达式为假，组件将被卸载 -->
    <HelloWorld v-if="isShow"/>
  </div>
</template>
<script>
import { ref } from "vue";
import HelloWorld from './components/HelloWorld.vue'
export default {
  name: 'App',
  components: {
    HelloWorld
  },
  setup() {
    // 演示组件是否被卸载
    let isShow = ref(true)
    return {
      isShow
    }
  }
```

```
}
</script>
```

(4) 在浏览器中运行，结果如图 2-6 所示。

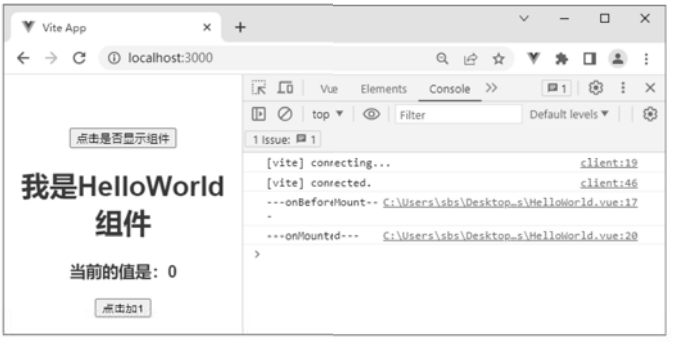


图 2-6　运行结果

(5) 通过图 2-6 可知，输出的信息直接触发挂载时的生命周期钩子，当单击“点击加 1”按钮时，更新的生命周期钩子就会被触发，如图 2-7 所示。

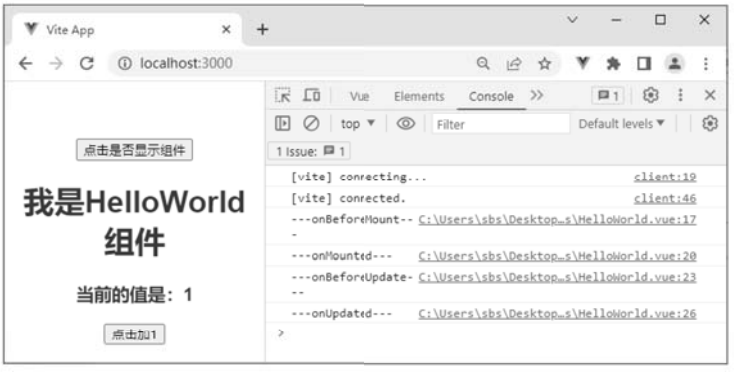


图 2-7　挂载时生命周期钩子被触发

(6) 当单击“点击是否显示组件”按钮时，将会触发卸载阶段的生命周期钩子，如图 2-8 所示。

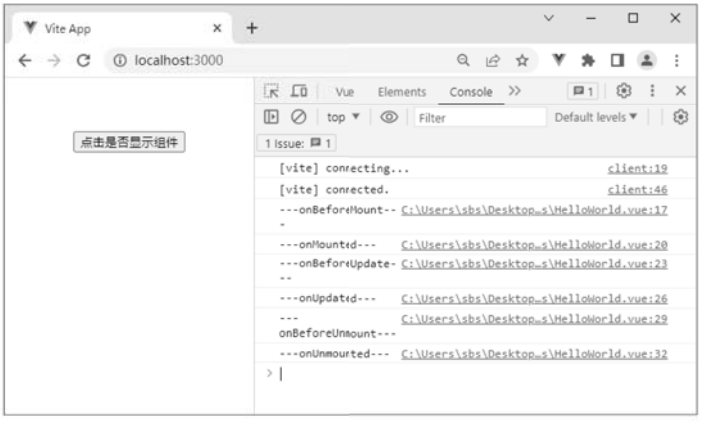


图 2-8　卸载时生命周期被触发

生命周期的流程如图 2-9 所示。

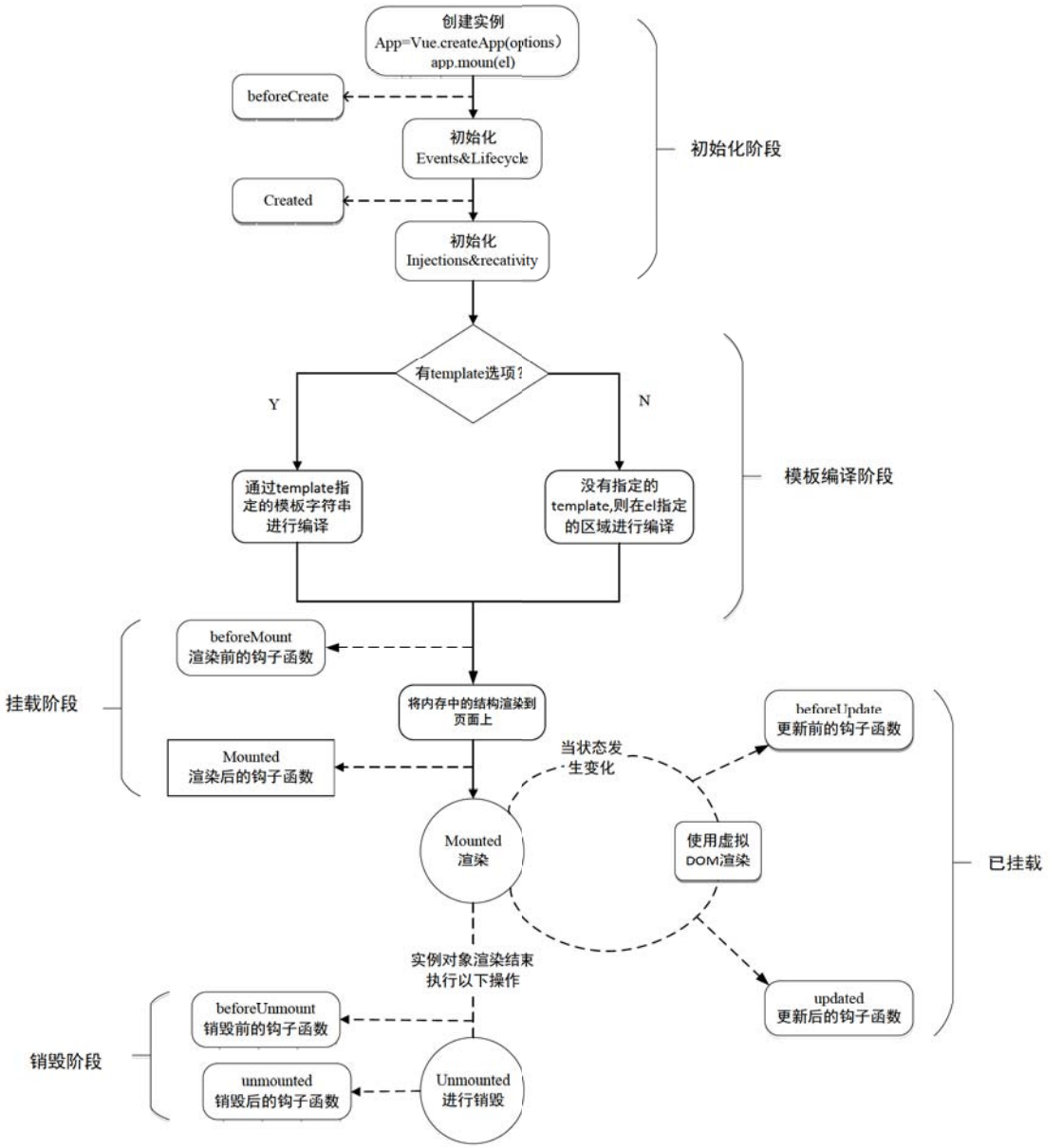


图 2-9　生命周期流程图

2.4　Vue 组件基础

组件在 Vue 中是一个非常重要的概念，可以将 Vue 组件理解为预先设定好的 ViewModel。每个 Vue 组件都可以预先定义很多选项，其核心主要是以下几个部分。

(1) 模板：模板是 Vue 提供的容器标签，主要是声明数据与最终展现给用户 DOM 之间的一种映射关系，在 Vue 3.0 中，<template>支持定义多个根节点。

(2) script：组件中的 JavaScript 行为，其中最重要的是以下两点。

① 数据：组件的初始数据的状态。

② 方法：对数据进行的增删改查等操作在此进行。

(3) style：主要是组件的样式。

2.4.1 创建 Vue 组件

在 Vue 3.0 中，一个组件实例是通过 createApp()函数来创建的，在组件实例被创建后，通过调用 mount()方法将组件实例挂载到页面中。

【例 2-7】全局组件的创建(实例文件 chapter-02\demo-04\index1.html)。

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>index1</title>
    <!-- 引入 Vue.js -->
    <script src="https://cdn.staticfile.org/vue/3.2.36/vue.global.min.js"></script>
</head>
<body>
    <div id="app">
        <!-- 引入组件 -->
        <overall></overall>
    </div>
    <script>
        // 创建一个 Vue 应用
        const app = Vue.createApp({})
        // 定义一个名为 overall 的全局组件
        app.component('overall', {
            template: '<h1>我是新创建的组件</h1>'
        })
        app.mount('#app')
    </script>
</body>
</html>
```

在浏览器中运行以上代码，结果如图 2-10 所示。

图 2-10 运行结果

【例 2-8】局部组件的创建(实例文件 chapter-02\demo-04\index2.html)。

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
```

```
    <title>index1</title>
    <!-- 进入 Vue.js -->
    <script src="https://cdn.staticfile.org/vue/3.2.36/vue.global.min.js"></script>
</head>
<body>
    <div id="app">
        <!-- 引入组件 -->
        <overall></overall>
    </div>
    <script>
        // 定义一个局部组件
        var overallA = {
            template: '<h1>我是新创建的组件!</h1>'
        }
        const app = Vue.createApp({
            components: {
                'overall': overallA
            }
        })
        app.mount('#app')
    </script>
</body>
</html>
```

在浏览器中运行以上代码，结果如图 2-11 所示。

图 2-11　运行结果

【例 2-9】父组件向子组件传递数据的实现(实例文件 chapter-02\demo-04\index3.html)。

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>index3</title>
    <!-- 进入 Vue.js -->
    <script src="https://cdn.staticfile.org/vue/3.2.36/vue.global.min.js"></script>
</head>
<body>
    <div id="app">
        <overall title="我是一号"></overall>
        <overall title="我是二号"></overall>
        <overall title="我是三号"></overall>
    </div>
    <script>
        const app = Vue.createApp({})
        app.component('overall', {
```

```
            props: ['title'],
            template: `<h1>{{ title }}</h1>`
        })
        app.mount('#app')
    </script>
</body>
</html>
```

在浏览器中运行以上代码，结果如图 2-12 所示。

图 2-12　运行结果

2.4.2　Vue 专用组件

Vue 本身对于状态的管理是独立的，各组件只需维护自身状态即可，而作为基于 Vue 状态管理框架的 Vuex，是一个专门为 Vue 定制的状态管理模块，它可以集中地存储和管理应用的所有组件的状态，使状态数据可以按照预期的方式变化。

1. Vuex 的由来

在日常开发中，对于组件较少且简单的 Vue 应用来说，单向数据流的状态管理模式非常高效，但是，当 Vue 应用遇到多个组件共享状态时，使用此方式进行状态管理则会非常困难，主要有以下两种情况。

(1) 多个视图依赖于同一状态。

对于嵌套的多个组件，可以通过传参的方式传递状态，但是对于兄弟组件之间的状态，传递就非常困难。

(2) 多个组件触发动作改变同一状态。

不同的组件若要改变同一状态，最直接的方式就是将触发动作交给上层，对于多层嵌套的组件，一层一层地上传，并在最上层处理状态的更改。

基于以上两种情况，Vuex 应景而生，Vuex 可以将组件间的共享状态抽取出来，以一个全局单例模式进行管理。在此种模式下，组件树构成一个巨大的视图，不管视图在组件树的哪个位置，任何组件都能获取共享状态，也可以直接触发修改动作，以此改变共享状态。

2. Vuex 的安装与使用

【例 2-10】Vuex 的安装与使用(实例文件 chapter-02\demo-04\test1)。

具体实现步骤如下。

(1) 使用 VS Code 创建一个 Vue 项目，具体创建步骤可参考 1.4.3 小节。

(2) 在生成的 Vue 项目中运行以下指令安装 Vuex。

```
npm install vuex@next --save
```

(3) 在 src 文件夹下新建一个 store 文件夹，并在 store 文件夹下新建一个 index.js 文件，然后在 index.js 文件中编写如下代码：

```
import { createStore } from 'vuex';
const store = createStore({
  state: {
    count: 1,
  },
  mutations: {
    // count 值累加
    increment(state) {
      state.count++;
    },
    // count 值累减
    decrement(state) {
      state.count--;
    },
  }
});
export default store;
```

(4) 修改 main.js 文件中的代码，将 store/index.js 引入全局，修改后的代码如下：

```
import { createApp } from 'vue'
import App from './App.vue'
import './index.css'
// 引入 index.js
import store from './store/index.js';
createApp(App).use(store).mount('#app')
```

(5) 修改 App.vue 文件，在文件中调用 store/index.js 文件中的属性和方法，修改后的代码如下：

```
<template>
  <!-- 输出 count 的值 -->
  <h1>Count: {{ $store.state.count }}</h1>
  <!-- 调用 increment -->
  <button @click="$store.commit('increment')">+</button>
  <!-- 调用 decrement -->
  <button @click="$store.commit('decrement')">-</button>
</template>
<script>
export default {
  name: 'App'
}
</script>
```

(6) 运行项目，结果如图 2-13 所示。

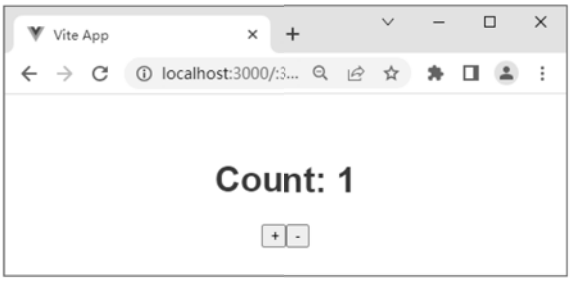


图 2-13　运行结果

当单击“+”或“-”按钮时，Count 的值将会进行累加或累减。

2.5　设计 Vue 组件

组件的设计主要是模块的设计，它体现在项目的业务需求、基本功能和性能上。经过上一节的讲解，相信读者已经了解了 Vue 组件的创建与使用方法。下面将通过一些实例来详细讲解组件中的 v-on 指令、v-model 指令和预留组件插槽功能的使用方法。

2.5.1　面向组件的 v-on 指令

v-on 指令主要用来监听 DOM 事件，在触发时运行 JavaScript 代码。v-on 指令的表达式有两种：一种是一般的 JavaScript 代码的形式，另一种是一个方法的名字或者是一个内联语句。

在使用 v-on 指令对事件进行绑定时，需要在 v-on 指令后面追加名称，例如：

```
<button v-on:click="onclick ">xxxx </button><!--标准写法-->
<button @click=" onclick ">xxxx</button><!-简略写法-->
```

【例 2-11】v-on 指令的使用(实例文件 chapter-02\demo-05\test1)。

具体实现步骤如下。

(1) 使用 VS Code 创建一个 Vue 项目，具体创建步骤可参考 1.4.3 小节。

(2) 在 HelloWorld.vue 文件中编写 v-on 指令，具体代码如下：

```
<template>
  <div>
    {{ num }}
    <button @click="counter">增加 1</button>
  </div>
</template>
<script>
import { defineComponent, ref } from 'vue'
export default defineComponent({
  setup() {
    // 声明双向数据 ref
    const num = ref()
    // js 通过.value 获取值
    num.value = 1;
    const counter = () => {
      num.value += 1;
```

```
    }
    return {
      num,
      counter
    }
  },
})
</script>
```

(3) 运行项目，结果如图 2-14 所示。

图 2-14　v-on 指令实例的运行结果

通过以上实例可知，在运行页面时单击按钮会执行 click 函数，进而可改变 num 的值。

2.5.2　面向组件的 v-model 指令

v-model 指令主要用来进行数据的双向绑定，例如表单<input>、<textarea>及<select>等。它可以根据控件类型自动选取正确的方法来更新元素，主要负责监听用户的输入事件以及更新数据，并对一些极端场景进行特殊处理。

【例 2-12】v-model 指令的使用(实例文件 chapter-02\demo-05\test2)。

具体实现步骤如下。

(1) 使用 VS Code 创建一个 Vue 项目。

(2) 在 HelloWorld.vue 文件中编写 v-model 指令，具体代码如下：

```
<template>
  <div>
    增加后的值：<input v-model="num" />
    <button @click="counter">增加 </button>
  </div>
</template>
<script>
import { defineComponent, ref } from 'vue'
export default defineComponent({
  setup() {
    // 声明双向数据 ref
    const num = ref()
    // js 通过.value 获取值
    num.value = 1;
    const counter = () => {
```

```
      num.value += 1;
    }
    return {
      num,
      counter
    }
  },
})
</script>
```

(3) 运行项目，结果如图 2-15 所示。

图 2-15　v-model 指令实例的运行结果

2.5.3　预留组件插槽

插槽(Slot)是指在 HTML 中起始标签和结束标签中间的部分，是 Vue 为组件封装者提供的一种能力，它允许开发者在封装组件时，把不确定的、希望由用户指定的内容定义为插槽。在使用 标签时，内部的插槽位置既可以放置要显示的文案，也可以放置嵌套的其他标签，可以将其理解为在组件封装时，为用户预留的内容的占位符。

在封装组件时，可以通过<slot>元素来定义插槽。

【例 2-13】预留组件插槽的使用(实例文件 chapter-02\demo-05\test3)。

具体实现步骤如下。

(1) 使用 VS Code 创建一个 Vue 项目。

(2) 在 HelloWorld.vue 文件中创建预留组件插槽，具体代码如下：

```
<template>
  <!-- 声明插槽 -->
  <slot name="one"></slot>
  <slot name="two"></slot>
</template>
<script>
export default {
  name: 'HelloWorld',
  props: {
    msg: String
  }
}
</script>
```

(3) 在 App.vue 文件中调用 HelloWorld.vue 文件，并给预留组件插槽赋值，具体代码如下：

```
<template>
  <HelloWorld>
    <!-- 为插槽赋值 -->
    <template #one>
      <h2>我是第一个预留插槽</h2>
    </template>
    <template #two>
      <h2>我是第二个预留插槽</h2>
    </template>
  </HelloWorld>
</template>
<script>
import HelloWorld from './components/HelloWorld.vue'
export default {
  name: 'App',
  components: {
    HelloWorld
  }
}
</script>
```

(4) 运行项目，结果如图 2-16 所示。

图 2-16　运行结果

2.6　使用现有组件

在日常开发中，除了可以使用自己创建的组件外，还可以使用 Vue 中现有的一些组件。其中现有组件又可以分为内置组件和外部组件。下面将详细介绍内置组件和外部组件在 Vue 项目中的使用。

2.6.1　使用内置组件

Vue 3.0 和 Vue 2.0 一样，都有很多内置组件，例如 componcnt、transition 内置组件等，但是相比于 Vue 2.0，Vue 3.0 新引入了 teleport、fragment 和 suspense 内置组件。下面将详细讲解这三种内置组件在项目中的使用方法。

1. teleport 组件

Teleport 意为传递、传送，是 Vue 3.0 新增的一个组件，通过该组件，开发者可以将相

关行为的逻辑和 UI 封装到同一个组件，以提高代码的聚合性。同样的功能在 Vue 2.0 中则是通过第三方库，或者使用$el 操作 DOM 等来实现。

【例 2-14】teleport 组件的使用(实例文件 chapter-02\demo-06\test1)。

具体实现步骤如下。

(1) 使用 VS Code 创建一个 Vue 项目。

(2) 在 App.vue 文件中使用 teleport 组件，具体代码如下：

```
<template>
  <h1>hello Vue1</h1>
  <h2>hello Vue2</h2>
  <!-- teleport 内置组件，将其渲染在 body -->
  <teleport to="h1">
    <div>teleport 内置组件</div>
  </teleport>
</template>
<script>
export default {
  name: 'App',
}
</script>
```

(3) 运行项目，结果如图 2-17 所示。

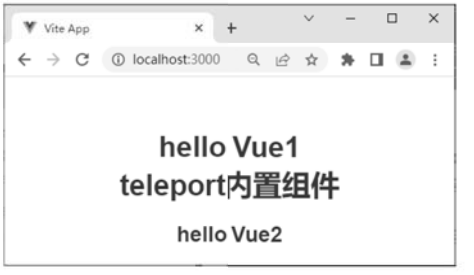


图 2-17　运行结果

teleport 组件中的参数 to 的值为将要移动到的位置。

2. fragment 组件

在 Vue 2.0 中，组件必须要有一个根标签，而在 Vue 3.0 中却不用。在 Vue 3.0 中，内部的多个标签会包含在一个 fragment 虚拟的元素中，它可以减少标签的层级，并减少内存量的使用。

【例 2-15】fragment 组件的使用(实例文件 chapter-02\demo-06\test2)。

具体实现步骤如下。

(1) 使用 VS Code 创建一个 Vue 项目。

(2) 在 App.vue 文件中使用 fragment 组件，具体代码如下：

```
<template>
  <h1>hello Vue1</h1>
  <h1>hello Vue2</h1>
</template>
<script>
export default {
```

```
  name: 'App',
}
</script>
```

(3) 运行项目，结果如图 2-18 所示。

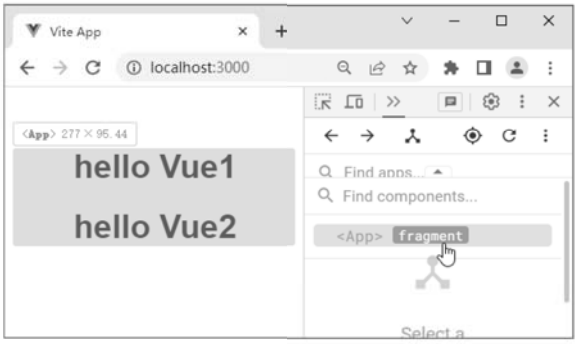


图 2-18　运行结果

3. suspense 组件

suspense 组件的主要作用是在等待异步组件的过程中渲染一些其他内容，通常在组件等待中获取数据(在异步 API 调用中)。在 Vue 2.0 中，需要使用条件来判断数据是否已经加载并显示回退的内容，而在 Vue 3.0 中，可以使用 suspense 组件完成数据加载时的跟踪和相应内容的渲染。

【例 2-16】suspense 组件的使用(实例文件 chapter-02\demo-06\test3)。

具体实现步骤如下。

(1) 使用 VS Code 创建一个 Vue 项目。

(2) 新建组件并命名为 Child.vue，在其中编写如下代码：

```
<template>
    <div class="child">
        <h3>我是 Child 组件</h3>
        name: {{ user.name }}
        age: {{ user.age }}
    </div>
</template>
<script>
export default {
    name: "Child",
    async setup() {
        const NanUser = () => {
            return new Promise((resolve, reject) => {
                // 延迟 2000 毫秒
                setTimeout(() => {
                    resolve({
                        name: "NanChen",
                        age: 20,
                    });
                }, 2000);
            });
        };
        const user = await NanUser();
        return {
```

```
            user,
        };
    },
};
</script>
<!-- 样式 -->
<style>
.child {
    background-color: skyblue;
    padding: 10px;
}
</style>
```

(3) 新建组件并命名为 Home.vue，并在此组件中引入 Child.vue 组件，具体代码如下：

```
<template>
  <div class="home">
    <h3>我是 Home 组件</h3>
    <Suspense>
      <template #default>
        <!-- 使用 Child 组件 -->
        <Child />
      </template>
      <template v-slot:fallback>
        <h3>Loading...</h3>
      </template>
    </Suspense>
  </div>
</template>
<script >
// import Child from './components/Child'//静态引入
import { defineAsyncComponent } from "vue";
const Child = defineAsyncComponent(() => import("./child.vue"));
export default {
  name: "Home",
  components: { Child },
};
</script>
  <!-- 样式 -->
<style>
.home {
  width: 300px;
  background-color: gray;
  padding: 10px;
}
</style>
```

(4) 在 App.vue 组件中引入 Home.vue 组件，具体代码如下：

```
<template>
  <Home />
</template>
<script>
import Home from './components/Home.vue'
export default {
  name: 'App',
  components: {
    Home
  }
```

```
}
</script>
```

(5) 运行项目，结果如图 2-19 所示。

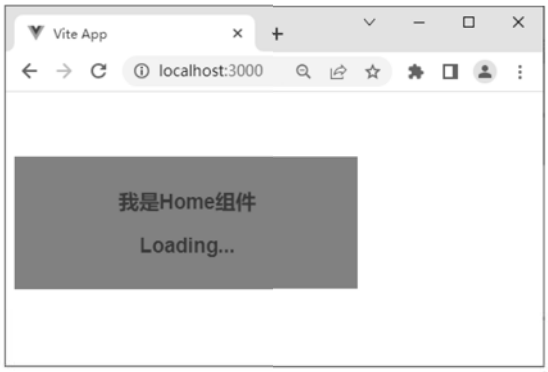


图 2-19　运行结果

(6) 在项目运行 2000 毫秒后，结果如图 2-20 所示。

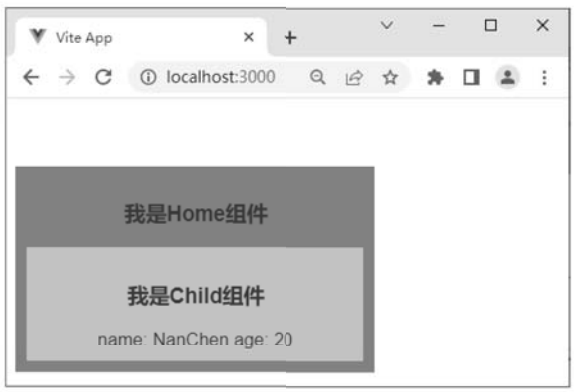


图 2-20　2000 毫秒后的运行结果

2.6.2　引入外部组件

axios 是一个基于 promise、用于浏览器和 Node.js 的 HTTP 客户端。axios 在 Vue 项目中可以向后台发送请求，进行接口 API 的调用，以此获取响应信息。其本质是对原生 XHR 的封装，是 promise 的实现版本，符合最新的 ES 规范。

1. axios 的特点

(1) 从浏览器中创建 XMLHttpRequests。
(2) 从 Node.js 创建 HTTP 请求。
(3) 支持 PromiseAPI。
(4) 拦截请求和响应。
(5) 转换请求数据和响应数据。
(6) 取消请求。

(7) 自动转换 JSON 数据。

(8) 客户端支持防御 XSRF。

2. axios 的安装和使用

【例 2-17】axios 的安装与使用(实例文件 chapter-02\demo-06\test4)。

具体实现步骤如下。

(1) 使用 VS Code 创建一个 Vue 项目。

(2) 在生成的 Vue 项目中运行以下指令安装 axios。

```
npm install axios
```

(3) 在 src 下新建一个 axios 文件夹，并在 axios 文件夹下新建一个 index.js 文件，然后在 index.js 文件中编写如下代码：

```
// 导入 axios
import axios from 'axios'
// 创建 axios 实例
const API = axios.create({
    //请求后端数据的基本地址
    baseURL: 'http://localhost:8080',
    //请求超时时间
    timeout: 2000
})
// 导出 axios 实例模块
export default API
```

(4) 在 main.js 文件中将 axios 全局引入，具体代码如下：

```
import { createApp } from 'vue'
import App from './App.vue'
import './index.css'
// 引入 axios
import axios from './axios/index.js'
const app = createApp(App);
app.mount('#app');
// 配置 axios 的全局引用
app.config.globalProperties.$axios = axios;
```

2.7 使用现有库

Vue 是一个非常强大的前端框架，它除了上节中讲解的现有组件外，还有很多现有的库，在日常开发中，使用这些库不仅可以快速地搭建出良好的界面，还可以有效地简化工作流程。下面通过实例来为读者讲解内置库和外部库的使用。

2.7.1 使用内置库

Element Plus 是前端 UI 框架，它本身是基于 Vue 的 UI 框架，在 Vue 项目中，也可以使用。Element Plus 内置了许多丰富的样式与布局框架，使用它可以有效地降低开发成本。下面将通过实例来介绍 Element Plus 的安装和使用。

【例 2-18】Element Plus 的安装与使用(实例文件 chapter-02\demo-07\test1)。

具体实现步骤如下。

(1) 使用 VS Code 创建一个 Vue 项目。

(2) 在生成的 Vue 项目中运行以下指令安装 Element Plus。

```
npm install element-plus --save
```

(3) 在 main.js 中引入 Element Plus，具体代码如下：

```
import { createApp } from 'vue'
import App from './App.vue'
import './index.css'
// 引入 element-plus
import ElementPlus from 'element-plus'
import '../node_modules/element-plus/theme-chalk/index.css'
createApp(App).use(ElementPlus).mount('#app')
```

(4) 在 HelloWorld.vue 文件中使用 Element Plus 库中的样式，具体代码如下：

```
<template>
  <!-- 使用 element-plus 中的样式 -->
  <el-button type="primary">Primary</el-button>
  <el-button type="success">Success</el-button>
</template>
<script>
export default {
  name: 'HelloWorld',
  props: {
    msg: String
  }
}
</script>
```

(5) 运行项目，结果如图 2-21 所示。

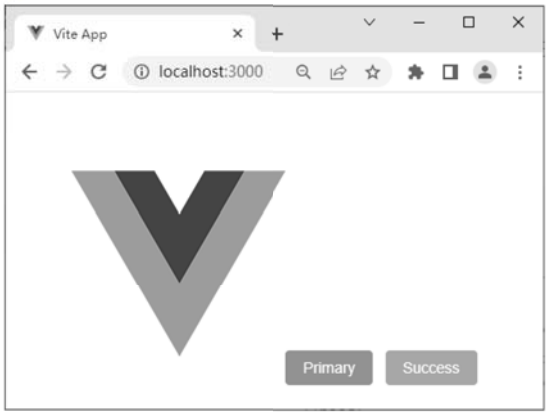


图 2-21　运行结果

2.7.2　引入外部库

Mock.js 是前端的一个模拟数据的库，在前端开发中可以使用这个库来避免大量的后端请求。在 Vue 项目中可以使用它生成随机数据，生成的随机数据支持文本、数字、布尔

值、日期、邮箱、链接、图片、颜色等。下面将通过实例来介绍 Mock.js 的安装和使用。

【例 2-19】Mock.js 的安装与使用(实例文件 chapter-02\demo-07\test2)。

具体实现步骤如下。

(1) 使用 VS Code 创建一个 Vue 项目。

(2) 在生成的 Vue 项目中运行以下指令安装 Mock.js。

```
npm install mockjs
```

(3) 在 src 下新建一个 mock 文件夹，并在 mock 文件夹下新建一个 index.js 文件，然后在 index.js 文件中编写如下代码：

```
// 引入mockjs
import Mock from 'mockjs'
const getdata = () => {
    return {
        id: 1,
        name: '李四'
    }
}
Mock.mock('/mock/get', getdata)
export default Mock
```

(4) 在 main.js 中引入 Mock.js，具体代码如下：

```
import { createApp } from 'vue'
import App from './App.vue'
import './index.css'
// 引入mock
import './mock/index.js'
const app = createApp(App);
app.mount('#app');
```

(5) 在项目中安装并引入 axios，具体安装步骤见 2.6.2 小节。

(6) 在 HelloWorld.vue 文件中调用 Mock.js 中的数据，具体代码如下：

```
<template>
  <div>
    <button @click="getrequ">发送请求</button>
    <div>id:{{ data.id }}</div>
    <div>name:{{ data.name }}</div>
  </div>
</template>
<script setup>
import { shallowReactive } from 'vue';
// 引入axios
import axios from '../axios/index.js'
// 声明data，用来存储请求结果
let data = shallowReactive({})
// 获取数据
const getrequ = async () => {
  let result = await axios({
    url: '/get',
    method: 'GET'
  })
  // 将data与结果合并，形成响应式对象
  Object.assign(data, result.data)
```

```
}
</script>
```

(7) 运行项目，结果如图 2-22 所示。

图 2-22　运行结果

(8) 单击“发送请求”按钮获取数据，结果如图 2-23 所示。

图 2-23　运行结果

2.8　本 章 小 结

本章主要介绍了 Vue 对象的挂载、操作关联数据以及 Vue 组件的创建方法，除此之外还介绍了 Vue 的现有组件以及现有库。通过本章的学习会发现，Vue 3.0 相比 Vue 2.0 增加了一些功能，在 Vue 3.0 中包含了单文件组件 setup()以及组合式 API 的写法，而且在组合式 API 使用过程中组件不需要被注册；在 Vue 2.0 中数据一般会放在 data 函数中，而在 Vue 3.0 中则是放在新增的 setup()函数中，在 setup()函数中采用 ref、reactive 进行初始化；另外，Vue 2.0 和 Vue 3.0 的生命周期也有所不同。在 2.6.1 节对 Vue 3.0 的内置组件进行了详细的介绍。