软件开发类人才培养系列教材
“人工智能+”新形态一体化教材

Vue3 全栈生态项目案例教程：AI赋能

主编　代　飞　梅　园　邓小政

上海交通大學出版社
SHANGHAI JIAO TONG UNIVERSITY PRESS

前言

本书深度贯彻国家“十四五”规划及后续发展战略，融入习近平新时代中国特色社会主义思想和党的二十大精神。党的二十大报告中提出“科技是第一生产力、人才是第一资源、创新是第一动力”，在此指导下，教育需培养高素质人才，助力国家发展。目前，顺应 Web（万维网）技术迭代，Vue3 成为前端开发主流框架。同时，AI（人工智能）工具链的普及，企业对“全栈化、工程化”人才的迫切需求，推动前端开发模式转型。本书响应产业变革，聚焦“AI 赋能开发”“产教深度融合”“真实项目驱动”，培育实战型“新前端”人才。

本书主要特色如下。

1. AI 赋能：重构开发工具链生态

将 DeepSeek-R1 代码生成工具、Fitten Code 智能调试助手融入开发全流程，借助“AI 辅助代码补全”“自动化错误诊断”等场景化教学，让读者掌握“人机协同”模式技能，提升编码效率与规范性，接轨企业前沿开发范式。

2. 产教融合：课岗对接的立体化资源

联合互联网头部企业技术专家编写，“综合案例”涵盖真实业务（如员工考勤管理等）与爱国主题场景；“项目实训”对接前端岗位典型任务。本书配套案例源码、项目实训、思政素养园地等资源，助力实现教学、实战、思政教育与职场需求衔接，有需要者可发邮件至教学助手 2393867076@qq.com 获取。

3. 技术前沿：覆盖 Vue3 生态最新技术栈

紧跟技术演进，解析 Vue3 组合式 API（应用程序接口）、Vite 极速构建等核心技术，讲解动态插槽名等进阶特性，帮助读者掌握从基础到企业级架构完整技术体系，避免知识滞后。

4. 能力进阶：三级结构化培养路径

构建基础筑基（项目 1~ 项目 4，建立 Vue3 开发认知）、进阶强化（项目 5~ 项目 9，培养复杂业务开发能力）、实战破局（项目 10，积累真实项目经验）三级培养路径。本书适合作为高等院校计算机相关专业的教材，也可作为开发者技术升级、企业培训的参考资料及 Vue3 应用开发爱好者的自学用书。教学建议采用“案例驱动 + 分组实训”模式，依托 AI 工具与企业案例库，引入企业专家线上指导，衔接校内学习与企业实践。

本书不仅是对 Vue3 技术栈的系统性梳理，更是对“AI+ 教育”模式的一次创新探索。希望通过本书，推动教育向“技术 + 工具 + 思维”三位一体的方向发展，为行业企业输送更多具备实战能力与创新意识的复合型人才。

本书编写过程中，得到了网易有道信息技术有限公司、华为云计算技术有限公司、北京博海迪信息科技股份有限公司等企业的大力支持，这些企业为本书提供了产业案例。同时，感谢上海交通大学出版社编辑团队的专业指导。因技术发展迅速，书中可能存在疏漏，恳请广大读者批评指正，以便再版时修订完善。

编　者

2025 年 3 月

目录

基础筑基篇

进阶强化篇

实战破局篇

项目 1 Vue3 框架基础

【项目导读】

在建设数字中国的征程中，掌握核心科技方能筑牢强国之基。作为国产主流框架，Vue3 不仅彰显着中国前端技术的创新力量，也彰显着新时代开发者“科技自立自强”的使命担当。通过组件化开发实践，本项目助力开发者锻炼自主可控的数字化建设能力，以代码书写科技报国的时代担当，为建设网络强国注入青春动能。

本项目聚焦 Vue3 框架基础，系统梳理 Vue 框架的发展脉络。从 Vue 的技术演进历程中，帮助开发者了解其从诞生到如今的蜕变；深入解析 Vue3 核心框架，分析响应式系统、组合式 API 等特性；详细讲解如何搭建开发环境、配置开发环境及借助 Vite 快速构建项目，提升开发效能；在实际操作中，帮助开发者掌握 Vue3 框架的基础知识，为后续深入学习打下坚实根基。

【项目目标】

知识目标

（1）了解 Vue 从 1.x 到 3.x 版本的技术演进历程，了解每个版本的重大更新和改进之处。

（2）掌握 Vue3 核心框架中的响应式系统、虚拟 DOM 原理及组合式 API 的设计理念。

（3）了解开发环境搭建所需的各类工具，如 Node.js、npm 的作用和安装与配置方法。

技能目标

（1）能够运用 Vue3 核心框架的知识，进行组件的开发、数据的绑定与交互和页面的布局。

（2）能够独立完成 Vue3 开发环境的搭建，包括工具的安装、版本的选择和环境变量的配置。

（3）能够熟练运用 Vite 快速搭建 Vue3 项目，掌握项目结构和配置文件的配置与使用。

素质目标

（1）培养对前端新技术的学习能力和探索精神，能够自主跟进 Vue 技术的发展动态。

（2）增强问题解决能力，在开发环境搭建和项目构建过程中，能够独立排查和解决遇到的问题。

（3）树立良好的代码规范和编程习惯，注重代码的可读性、可维护性和性能优化。

1.1 Vue3 技术演进认知

2014 年，尤雨溪推出创新前端框架 Vue.js（简称 Vue）。凭借渐进式架构与开发者友好的特性，该框架迅速成为行业关注焦点。Vue 基于响应式数据绑定和虚拟 DOM 算法这两大核心技术，构建了高效灵活的组件化开发模式。在 Vue 1.x 至 Vue 3.x 的技术演进历程中，组合式 API（Composition API）的引入，在逻辑复用、TypeScript 支持及代码可维护性等方面都发挥了积极作用。其背后依托的 RFC（Request for Comments，请求评论，由互联网工程任务组颁布的一系列备忘录）协作机制，将社区智慧融入技术演进，确保框架升级既保持兼容性又具备前瞻性。

通过集成 Vite 新一代构建工具与 Pinia 状态管理库，Vue 生态形成了完整的现代工具链，在开发效率与工程化方面树立新标杆。服务端渲染（Server-Side Rendering，SSR）的深度优化，解决了首屏加载性能与搜索引擎优化（Search Engine Optimization，SEO）的痛点，拓宽了框架在全栈开发场景的应用边界。从选项式 API 到组合式 API 的范式迁移，不仅重塑了代码组织方式，更培养了开发者应对前端工程化迭代的技术敏锐度与架构设计能力，为技术选型提供依据，进一步稳固了 Vue 作为现代 Web 开发首选框架的地位。

1.2 Vue3 核心框架解析

1.2.1 Vue 概念解析

Vue 是一款渐进式的用户界面构建框架。与其他大型框架不同的是，它支持自底向上逐步应用，具有十足的灵活性。Vue 核心库聚焦视图层，上手简单且能轻松与第三方库或已有项目集成。同时，在搭配现代化工具链和各类支持类库的情况下，Vue 完全能够胜任复杂单页应用的开发驱动工作。

1.2.2 Vue 特性概览

1. 轻量级

与 React 和 Angular 相比，Vue 是更为轻量级的前端库。它的 API 简单且灵活，开发者能够轻松上手和运用。

2. 响应式更新

Vue 拥有一套强大的响应式系统，它能够自动追踪模板表达式及计算属性所依赖的数据。一旦这些依赖的数据发生任何变化，Vue 便会立即做出响应，自动更新视图，从而完美实现数据驱动视图更新的高效机制。

3. 指令

Vue 实现与页面交互的关键途径是指令。借助指令这一强大工具，开发者能够便捷地操控响应式数据，同时灵活决定模板 DOM（文档对象模型）元素的呈现样式与行为，极大地提升了开发效率与页面交互的灵活性。

4. 组件化开发

组件是 Vue 最为强大的功能之一。它对 HTML（超文本标记语言）元素进行了拓展，将可重复利用的代

码块巧妙封装起来。通过传参，组件能够接收来自父级的数据和向父级传递信息，从而构建起了父子组件间的通信桥梁。在使用方式上，组件支持局部注册、全局注册及动态组件等多种形式，在实际开发中极大地提升了代码的复用率，让代码的维护变得更加轻松高效。

1.2.3 Vue 版本迭代

1. Vue 1.x

Vue 1.x 发布于 2015 年，其主要特性如下。

（1）双向绑定：能让数据与视图实时同步更新，极大地提升开发效率。

（2）指令系统：赋予开发者便捷操控 DOM 元素的能力，轻松实现各种交互效果。

（3）组件化：将代码模块化，可复用性大大增强。

凭借这些出色特性，Vue 1.x 自发布之初就备受瞩目，迅速成为前端开发领域的热门框架。

2. Vue 2.x

Vue 2.x 发布于 2016 年，其在发展过程中引入了一系列重要特性，主要特性如下。

（1）虚拟 DOM 的加入，显著提升了渲染性能，优化了页面加载速度。

（2）单文件组件的采用，将组件相关的模板、脚本和样式整合在一处，极大地方便了组件的开发与维护。

（3）在 Vue 2.x 版本中，类型支持得到进一步强化，错误处理机制也更为完善，让开发者在构建应用时能获得更好的开发体验，代码质量也更有保障。

3. Vue 3.x

Vue 3.x 发布于 2020 年。Vue 3.x 带来了重大更新，新增了诸多关键特性，如组合式 API、TypeScript 原生支持、Tree-Shaking 优化及性能提升等。其中，组合式 API 提供了一种全新的代码组织与复用方式，与 React 的 Hooks 类似，能够帮助开发者更清晰地梳理复杂业务逻辑，有效减少代码冗余。

在性能方面，Vue 3.x 表现卓越。通过编译时优化和运行时性能改进，它显著提升了应用的运行速度，缩短了初始加载时间，同时降低了内存占用，为用户带来更加流畅的使用体验。

1.3 开发环境搭建

1.3.1 高效编辑器

VS Code 安装流程

1. Visual Studio Code 编辑器

Visual Studio Code，简称 VS Code，是微软公司于 2015 年 4 月 30 日在 Build 开发者大会上正式发布的一款跨平台源代码编辑器。它能在 macOS、Windows 及 Linux 操作系统上运行，尤其适用于编写现代 Web 和云应用程序，并且可在桌面环境中流畅使用，无论是 macOS、Windows 及 Linux 操作系统的用户都能轻松上手。

VS Code 对 JavaScript、TypeScript 及 Node.js 有着内置支持，其生态系统中还拥有丰富的扩展，涵盖了众多其他语言，如 C++、C#、Java、Python、PHP、Go 等。基于这些强大的功能和丰富的

扩展，本书在进行 Vue 项目开发时，选择了 VS Code 编辑器，期望借助它高效地完成项目开发工作。

2. Trae CN 编辑器


Trae CN 安装流程

Trae CN 是字节跳动公司于 2025 年 3 月 3 日推出的国内首款 AI 原生集成开发环境（AI IDE），基于开源项目 Visual Studio Code 1.97.2 深度定制开发，支持 macOS、Windows 和 Linux 等多平台运行。该工具深度融合国产 AI 大模型（如豆包 Doubao-1.5-Pro、深度求索 DeepSeek R1/V3），具备通过自然语言生成代码的“Builder 模式”和智能交互的“Chat 模式”，可快速构建项目框架（如 10 分钟生成贪吃蛇游戏代码）。其核心优势在于全中文界面设计、本土化语义优化（中文变量名解析准确率达 92%），以及零配置开发环境，特别适合 Web 开发、自动化脚本编写（如文件批量处理）和教学场景（如动画效果生成）。相较于传统 IDE，Trae CN 通过 AI 能够实现从需求描述到代码生成的全流程智能化，目前免费开放使用。

1.3.2 AI 赋能助手工具链

1. DeepSeek-R1 AI 编程助手


插件安装及代码编写

DeepSeek-R1 是一款强大的具有深度思考功能的语言大模型，接入它能为开发者提供智能代码补全、语法纠错等功能，帮助开发者提高代码编写的效率和准确性。它还可以理解代码上下文，提供相关的代码建议和文档注释，有助于开发者更好地理解和维护代码。此外，接入 DeepSeek-R1 能够为 VS Code 中已支持的多种编程语言提供更智能的辅助能力，为不同领域的开发者提供统一的智能编程体验，提升开发的便利性和流畅性，进而推动软件开发过程的智能化和高效化。

2. Fitten Code AI 编程助手


模型安装及代码编写

Fitten Code 又迎来全新重磅升级，新增自主编程智能体（Agent）功能。智能体具备自主决策、主动迭代式调用工具的功能，可以根据任务需求完成项目级代码生成和修改。无论是经验丰富的程序员，还是刚刚踏入编程领域的新人，自主编程智能体都能成为强大的编程助手。

自主编程智能体功能拥有非凡的“主动性”和强大的“执行力”，智能体能够根据任务需求调用合适的工具，主动获取并理解任务背景信息，同时分析并制定解决方案。通过迭代式的执行过程，智能体可以分解复杂问题并逐步执行，实现高效且精细的任务处理，提升自动化编程的智能化水平。在使用该功能时，用户只需要输入需要完成的开发任务，如“给这个项目加上多语言功能支持”“将这个 jsp 项目转换为 vue 项目”“修正以下报错”等，智能体就可以根据任务调用不同工具逐步完成开发任务。

1.3.3 Node.js 环境配置与版本管理

Node.js 是依托 Chrome V8 引擎构建的 JavaScript 运行环境，其核心特性包括事件驱动和非阻塞式 I/O 模型。这些设计使其能够高效处理高并发场景，推动 JavaScript 从客户端语言扩展到服务端领域。Node.js 的出现打破了 JavaScript 仅限于浏览器端的限制，使其成为服务端开发的重要工具，并实现了 JavaScript 与 PHP、Python、Ruby 等传统服务端语言的并列，成为开发服务端应用的得力脚本语言。

为适配特定应用场景，Node.js 精心优化，提供了独特的 API，有效助力 Chrome V8 引擎在非浏览器

环境中实现卓越性能。Chrome V8 引擎执行 JavaScript 时速度惊人，性能十分出色。Node.js 正是借助这一优势，打造出便于搭建网络应用的平台，所构建的应用具备响应迅速、易于扩展的显著特点。接下来将为大家详细讲解 Node.js 的安装流程，帮助大家快速开启 Node.js 的开发之旅。

在浏览器网址栏输入“https://nodejs.org/zh-cn”，打开 Node.js 官网，如图 1-1 所示。单击“Get Node.js®”按钮，在跳转后的下载界面中，选择合适的版本进行下载。

下载完成后，会得到一个后缀为“.msi”的安装包文件，双击运行，如图 1-2 所示。

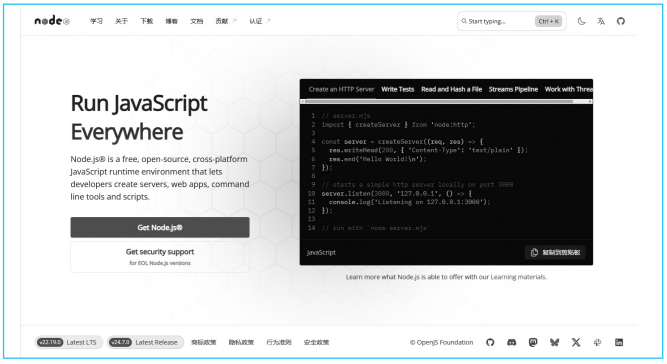

图 1-1　Node.js 官网

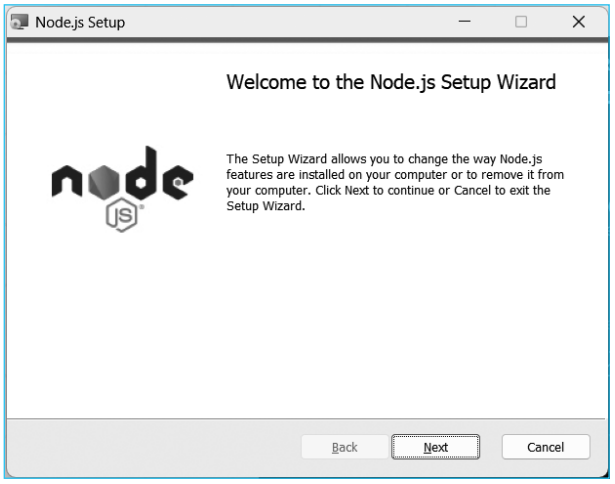

图 1-2　安装包文件运行页面

全部使用默认选项，并按照提示进行下一步操作，即可成功安装。

如果想要测试 Node.js 是否安装成功，可以在 Windows 系统中，按“Win+R”键，弹出“运行”对话框。在对话框中输入“cmd”并按“Enter”键，打开命令提示符窗口。在该窗口中输入“node -v”（这里的“v”是“version”的缩写，意为版本），完成输入后按“Enter”键。若窗口正常显示 Node.js 的版本信息，则表明 Node.js 已成功安装。

1.3.4　包管理工具

在 Vue 开发的广阔天地里，软件包管理工具（如 npm 与 yarn）占据着举足轻重的地位。它们极大地简化了依赖的安装与管理流程，成为提升开发效率、保障项目可维护性的得力助手。

Vue 项目的顺利运转，往往离不开众多库和插件的支撑。借助软件包管理工具，开发者只需轻松操作，就能完成依赖项的安装、更新与卸载，为项目的稳定性与安全性提供可靠保障。不仅如此，这些工具还具备版本锁定功能，如同给项目上了一把坚实的锁，有效规避了版本不一致引发的各类棘手问题。

npm 作为 Node.js 默认的软件包管理工具，凭借广泛的使用范围和强大的社区支持，在开发者群体中广受欢迎。而 yarn 另辟蹊径，致力于攻克 npm 在特定场景下的性能瓶颈与安全隐患，展现出了更快的包安装速度和更为出色的依赖管理能力。

综上所述，软件包管理工具已然成为 Vue 开发中不可或缺的利器。它们为开发者搭建了高效便捷的依赖管理桥梁，助力打造高质量且易于维护的 Vue 应用。当 Node.js 安装成功后，npm 包管理工具也随之安装成功。本书将以 npm 为主要工具，详细展开相关内容的讲解，引领大家深入探索 Vue 开发的奥秘。

在命令提示符窗口中执行命令“npm -v”，查看 npm 是否已经跟随 Node.js 安装，并查看相应版本，如图 1-3 所示。

npm 提供了很多命令，可以使用“npm help”来查看所有命令。表 1-1 仅展示和说明部分 npm 常用命令，其中包名需自行替换为真实包名（如 axios 包）。

表 1-1　npm 常用命令

命令	功能
npm install 包名	用于为项目安装指定名称的包
npm install -g 包名	全局安装指定包，可在系统范围使用
npm uninstall 包名	用于卸载指定名称的包
npm update 包名	用于更新指定名称的包

在下载 npm 安装包的过程中，或许会出现下载速度迟缓的情况，其根源在于提供安装包的服务器在国外。其实，只需将镜像源配置为国内服务器，就能显著提升包的下载速度。为 npm 设置镜像地址的命令如下：

```
npm config set registry https://registry.npmmirror.com
```

若想验证镜像地址是否配置成功，可使用如下命令查看。

```
npm config get registry
```

配置成功效果如图 1-4 所示。

图 1-3　查看对应 npm 版本

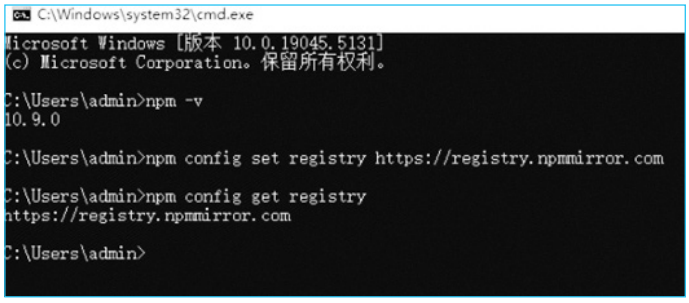

图 1-4　配置成功效果

1.4　Vite 快速构建项目

1.4.1　下一代前端构建工具——Vite

Vite 作为一款崭露头角的前端构建工具，正以其卓越性能为前端开发体验带来质的飞跃。它主要涵盖两大核心部分，分别是开发服务器和构建指令。这两部分共同构成了 Vite 的核心架构，使其在现代前端开发中脱颖而出。

（1）Vite 拥有一个功能强大的开发服务器，该服务器依托原生 ES 模块构建。它具备一系列丰富且实用的内建功能，其中模块热替换（HMR）尤为引人注目，其速度快得超乎想象。

（2）Vite 还配备了一套构建指令，这些指令借助 Rollup（一个轻量且高效的 JavaScript 打包工具）来打

包代码。同时，它们已预先完成配置，能够将代码打包成高度优化的静态资源，直接用于生产环境，确保交付的产品在性能上达到最佳状态。

Vite 是一款极具针对性的前端工具，拥有合理且贴心的默认设置，能让开发者快速上手，无需进行烦琐的初始配置。

同时，借助插件机制，Vite 具备强大的兼容性，可以轻松地与其他框架或工具实现集成，无论是流行的前端框架，还是实用的开发工具，都能与 Vite 完美协作。

此外，Vite 的扩展性十分出色。开发者可以利用其插件 API 和 JavaScript API 对 Vite 进行深度定制和扩展，满足各种个性化的开发需求。Vite 还提供了完整的类型支持，这为使用 TypeScript 进行开发的开发者提供了极大的便利，能够有效提升代码的可读性和可维护性，让开发过程更加高效、稳定。

1.4.2 Vue3 项目初始化全流程

在开始本小节的学习或操作之前，请务必确认已按照前文文档的指引，正确完成了相关配置，并且确保所有配置均能正常运行。只有这样，才能为后续的学习或操作提供稳定的基础，避免因前期配置问题导致的错误或阻碍。在本小节中，将借助 VS Code 编辑器自带的终端来创建 Vue3 项目，并详细讲解整个创建流程，配以相关图片，帮助大家更直观、清晰地理解每一个步骤。

1. 环境准备

（1）在桌面新建一个文件夹来保存创建的项目，将其命名为“project”。

（2）运行 VS Code 编辑器，在编辑器界面左上角，单击“文件”选项卡，选择“打开文件夹”命令，如图 1-5 所示。在弹出的窗口中，定位到桌面，选中刚刚创建的“project”文件夹，单击“选择文件夹”按钮，切换到“project”文件夹路径。

（3）单击 VS Code 编辑器工具栏中的“终端”选项卡，在弹出的菜单中选择“新建终端”命令，如图 1-6 所示。

图 1-5 选择“打开文件夹”命令

图 1-6 新建终端

（4）此时，需要留意终端窗口的右上角，查看当前使用的工具是否为 cmd 或者 powershell，如图 1-7 所示。倘若不是，可以单击工具名称，在弹出的选项卡中进行切换操作。

图 1-7 终端为 powershell

当你顺利完成上述所有步骤，就为接下来的项目创建奠定了基础，现在，我们就可以正式开启使用 Vite 快速创建 Vue3 项目的学习之旅啦！

2. 创建项目

Vite 为我们提供了两种便捷创建项目的命令方式：一种是手动创建项目的命令，另一种则是借助模板自动创建项目的命令。接下来，将对这两种方式展开详细讲解，帮助开发者清晰掌握创建项目的流程。倘若在创建过程中遇到了错误（bug）也不必担心，在相关内容讲解结束后，会针对几种常见错误给出具体的解决方案，帮助开发者顺利推进项目创建。

1）手动创建项目

打开命令提示符对话框，运用 npm 软件包管理工具，在终端中输入如下命令，完成输入后，按“Enter”键执行该命令操作。

```
npm create vite@latest
```

提示

在利用 npm 安装第三方软件包时，借助 @ 符号，能够灵活指定所需安装的版本。其中，若使用“latest”，则代表安装该软件包的最新版本。

按下“Enter”键后，终端界面会依次弹出提示信息，要求输入项目名称（见图 1-8），从给定选项中挑选合适的项目环境（见图 1-9）、语言环境（见图 1-10）等，创建成功界面如图 1-11 所示。

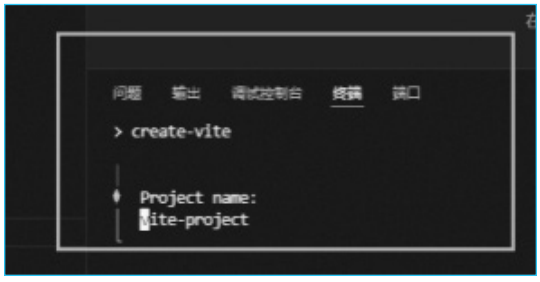

图 1-8 输入项目名称

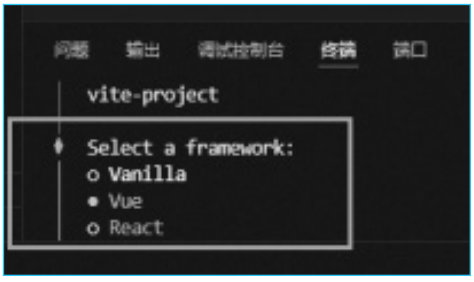

图 1-9 选择项目环境

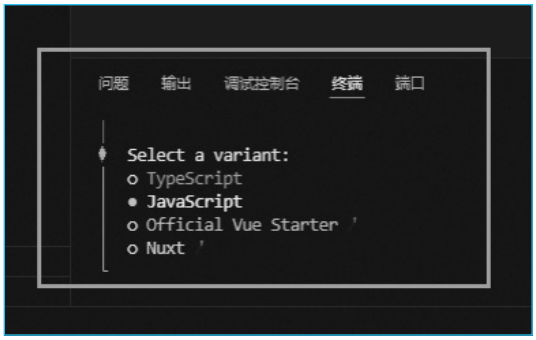

图 1-10 选择语言环境

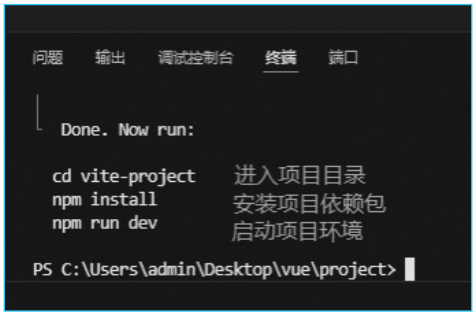

图 1-11 创建成功界面

多学一招

在创建项目的过程中，可能会碰到一些错误。接下来，将深入分析这些错误产生的原因，并详细介

绍相应的解决办法。

bug1：倘若在项目创建进程中，系统提示某个第三方包尚未安装，此时请依照提示信息，在终端中输入“Y”或“y”，确认安装该第三方包，待安装完成后，便可继续推进项目的创建流程，如图 1-12 所示。

图 1-12　bug1

bug2：在安装操作期间，若出现类似图 1-13 所示的“无法加载文件”提示，需要检查当前所在的对话框是否使用的是 powershell 工具。如果确实是 powershell 环境，可以采用以下两种解决方案。

（1）切换为 cmd 工具，正常使用 npm。

（2）若使用的是 powershell 工具，则需要更改执行策略以允许脚本运行。可以在具备管理员权限的 powershell 窗口中执行以下命令。

```
Set-ExecutionPolicy RemoteSigned
```

更新完成后即可正常执行 npm 命令。

图 1-13　bug2

2）借助模板自动创建项目

Vite 允许我们通过附加命令行选项的方式，直接指定项目名称和模板，这样就无需再手动填写项目名称、选择框架及语言环境等内容，大大简化了项目创建流程。Vite 也提供了丰富的模板预设。借助这些预设，用户能够轻松创建出 Vite+React+TS、Vite+Vue、Vite+Svelte 等不同类型的项目。通过附加命令行选项直接指定项目名称和模板的基本语法格式如下：

```
npm create vite@latest < 项目名称 > -- --template < 模板名称 >
```

注意

在这个命令中，第一个参数为空。需要注意的是，“--”是由两个减号组成，“--”后面不需要添加空格，但“--”与后续参数之间需要有一个空格。

“--template < 模板名称 >”表明此时正在借助 Vite 模板来创建项目。例如，如果想要创建一个 Vue3 项目，只需将模板名称设定为“Vue”即可。

接下来，为大家详细演示如何利用模板来自动创建一个 Vue3 项目。

（1）切换至 VS Code 的终端界面，找到存放 Vue 项目的文件夹路径，并进入该文件夹目录。

（2）在终端输入“npm create vite@latest vue_template_starter -- --template vue”命令（使用 cmd 命令行执行），来创建一个基于 Vite+Vue 类型的、名为“vue_template_starter”的项目。项目创建成功后，呈现的界面效果如图 1-14 所示。

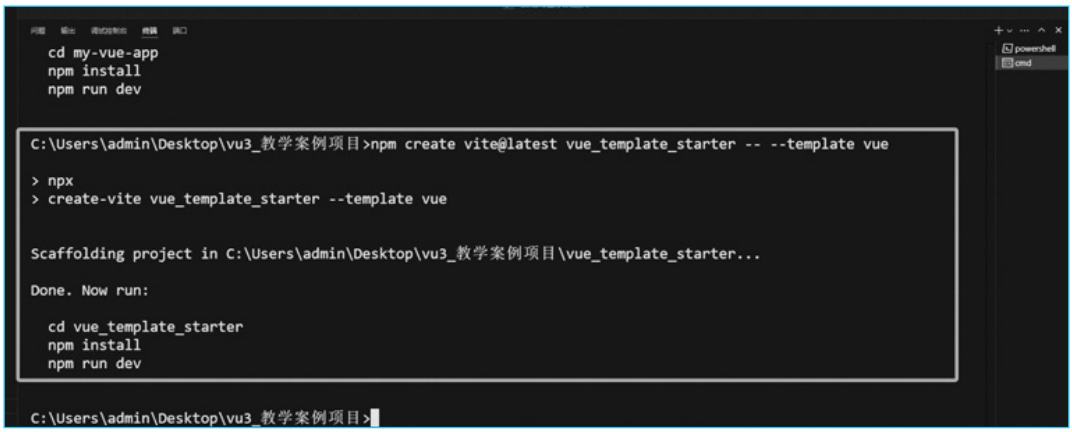

图 1-14 通过模板自动创建项目成功界面

注意

若在创建项目时使用的终端是 powershell，则在自动创建项目的过程中，可能仍需要像手动创建那样输入项目参数。这是因为 powershell 对执行脚本命令有着较高的规范性要求。不过，将命令中的 npm 替换为 npx，便可解决问题。

```
npx create vite@latest vue_template_starter -- --template vue
```

1.4.3 Vue3 标准项目结构解析

接下来，将依据 1.4.2 小节所介绍的自动创建项目模式，对项目结构展开详细讲解。Vue3 标准项目结构的具体呈现如图 1-15 所示。

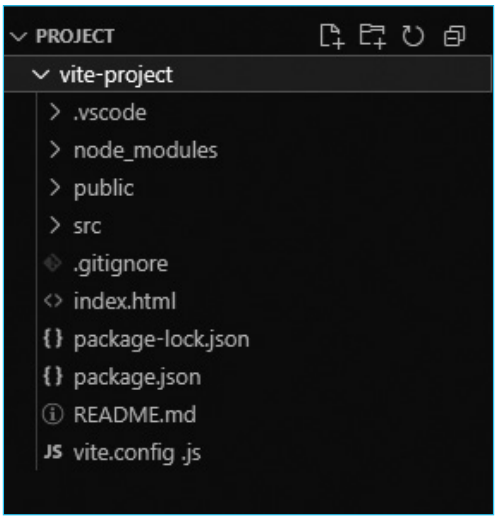

图 1-15 Vue3 标准项目结构的具体呈现

目录详细解析说明如表 1-2 所示。

表 1-2　目录详细解析说明

目录 / 文件	说明
node_modules/	存放项目的所有依赖包，由 npm 或 yarn 自动生成和管理
public/	静态文件目录，里面的文件不会被 Webpack 处理，最终会原样复制到打包目录下
public/favicon.ico	网站的图标
public/index.html	应用的主 HTML 文件，Vue CLI 会在构建时自动注入生成的静态资源链接
src/	源代码目录，存放应用的主要代码
src/assets/	存放静态资源，如图像、字体等。这些文件会由 Webpack 处理，可以通过相对路径引用
src/assets/logo.png	示例图像文件
src/components/	存放 Vue 组件，每个组件都是一个独立的 .vue 文件
src/components/HelloWorld.vue	默认生成的示例组
src/views/	存放视图组件，通常对应路由，每个视图都是一个独立的 .vue 文件
src/views/Home.vue	默认生成的主页组件
src/router/	存放路由配置文件
src/router/index.js	路由的配置文件，定义了应用的路由规则
src/App.vue	根组件，整个应用的入口组件
src/main.js	应用的入口文件，负责创建 Vue 实例并挂载到 DOM 上
.gitignore	Git 忽略文件列表，指定哪些文件和目录不被包含在版本控制中
babel.config.js	Babel 配置文件，指定 Babel 的编译规则
package.json	项目的依赖、脚本和其他元数据
README.md	项目的说明文件，通常用于描述项目、存储项目、如何安装和使用等信息
vue.config.js	Vue CLI 的配置文件，用于修改默认配置
yarn.lock 或 package-lock.json	锁定安装的依赖版本，确保项目依赖的一致性

综合案例：构建 Vue3 项目

构建 Vue3 项目

【项目小结】

本项目全面覆盖 Vue3 框架基础认知。从 Vue 技术简介与发展历程切入，梳理其从 Vue 1.x 到 Vue 3.x 的演进逻辑，凸显响应式系统、组合式 API 等核心特性；在开发环境配置环节，详解 Visual Studio Code 与 Trae CN 编辑器的协同应用；结合 DeepSeek-R1 与 Fitten Code AI 编程助手提升开发效率，同步讲解 Node.js 环境配置、版本管理及包管理工具使用；重点演示基于 Vite 快速创建 Vue3 项目的流程，展现其模块化开发、热更新及高效构建的优势，构建从环境准备到项目初始化的完整技术链条，为实战开发筑牢理论与工具基础。

【项目实训】

项目实训

项目 2 Vue3 开发核心基础

【项目导读】

深入了解 Vue3 开发核心基础，不仅是学习一项前沿的前端主流技术生态，更是锤炼建设网络强国、数字中国的关键技能。党的二十大报告强调科技自立自强，号召青年投身创新实践。掌握 Vue3 这一灵活高效的工具，正是响应国家战略性新兴产业发展的需要。

本项目将围绕 Vue3 开发核心基础展开。在本次项目中，开发者将学习单文件组件规范，掌握项目结构搭建；深入探究模板语法与响应式系统，理解数据驱动视图的原理；通过计算属性与依赖缓存机制，优化性能；利用侦听器高级应用精准响应数据变化；掌握动态样式绑定，实现页面美观交互；对比 API 模式，合理选型。在实操中掌握 Vue3 开发核心技能，为开发复杂项目筑牢根基。

【项目目标】

知识目标

（1）掌握单文件组件规范的详细内容，包括组件结构、命名规则和文件组织方式。
（2）理解模板语法与响应式系统的工作原理，知晓数据变化。
（3）掌握计算属性与依赖缓存机制，明白其对性能优化的作用和实现原理。
（4）了解侦听器高级应用场景，如深度监听、立即执行等功能的使用条件。
（5）了解不同 API 模式（选项式 API、组合式 API）的特点、适用场景和区别。

技能目标

（1）能够按照单文件组件规范，独立创建和组织 Vue3 项目中的组件。
（2）能够熟练运用模板语法和响应式系统，实现数据的双向绑定和动态更新。
（3）能够运用计算属性与依赖缓存机制优化代码，提高项目的性能和响应速度。
（4）能够灵活使用侦听器高级应用，精准处理数据变化并执行相应逻辑。
（5）能够根据项目需求，准确对比并选择合适的 API 模式进行开发。

素质目标

（1）培养严谨的代码规范意识，遵循单文件组件规范和良好的编程习惯。
（2）增强对性能优化的敏感度，在开发中主动运用计算属性等机制提升项目性能。
（3）提升逻辑思维能力，通过使用侦听器高级应用精准处理复杂的数据变化逻辑。
（4）树立全局思维，在 API 模式选型时综合考虑项目规模、团队协作等因素。
（5）具备自主学习能力和探索精神，能够持续跟进 Vue3 技术发展，不断更新知识技能。

2.1 单文件组件规范

Vue 是一款对组件化开发支持力度极高的框架，在借助 Vite 顺利搭建 Vue3 项目后，项目目录中存在很多以 .vue 为扩展名的文件。这些 .vue 文件意义非凡，每一个都用于定义一个独立的单文件组件，是 Vue 项目构建的基础模块。

2.1.1 单文件组件的组成部分

Vue 中的单文件组件，是一种别具一格的文件格式。它的独到之处在于，把每个组件的模板（template）、样式（style）及逻辑（script）这三大关键部分，整合于同一个文件中。这种设计使得组件的结构更为紧凑，管理与维护也更加便捷。接下来，我们将分别深入探究模板、样式和逻辑这三个组成部分，详细解析它们各自的功能及具体用法。

1. 模板

在 Vue 组件里，模板（template）扮演着至关重要的角色，它堪称构建组件视图的核心要素，如同绘制组件 DOM 结构的精准“蓝图”。Vue 专门提供了 <template> 标签作为容器，用于容纳模板的内容，通过它来定义组件的 HTML 结构。需要特别注意的是，<template> 标签在实际渲染过程中不会转化为真实的 DOM 元素，它存在的意义主要是有序地组织代码，让开发者能够更清晰、高效地构建组件的视图架构。

在 Vue 单文件组件的规范体系里，每个组件的模板遵循一项严格要求：必须存在且仅能存在一个顶层的根元素。这一设计绝非偶然，它有着重要意义。一方面，该设计保证了组件结构的清晰明了，开发人员在构建和维护组件时，能够通过唯一的根元素迅速把握组件整体布局；另一方面，Vue 内部的优化与管理机制也依赖于此，单一的根元素让 Vue 在处理组件渲染、更新等操作时更加高效，有助于提升应用性能。

Vue3 在模板渲染方面实现了重大突破，显著提升了灵活性。相较于 Vue2，Vue3 引入了一项关键改进：允许 <template> 标签内部存在多个根节点。这一变革为开发者带来了更为广阔的布局空间，极大地增强了设计的灵活性，使开发者能够更自由地构建复杂的用户界面结构。

反观 Vue2 版本，其在模板渲染上有着严格限制，<template> 标签必须拥有一个单一的根节点，所有子节点都需由这个根节点包裹。若违背这一规则，便会触发错误，导致组件无法正常渲染与运行，这在一定程度上束缚了开发者的创意与布局选择。Vue3 的这一改进，有效解决了 Vue2 的局限性，助力开发者更高效地打造优质应用。

2. 样式

在 Vue 组件中，样式部分起着关键作用。它借助 CSS（串联样式表）代码来塑造组件在浏览器页面渲染时的视觉效果。这些 CSS 代码被安置在 <style> 标签内，开发者可以利用这一机制在单文件组件中灵活地定义和管理样式。在 Vue 单文件组件的架构里，每个组件拥有灵活的样式定制能力。开发者可依据实际需求，在组件内添加多个 <style> 标签，从而拥有充裕的空间去定义多样化的样式规则，满足复杂的设计要求。

值得一提的是，如果某个组件在特定场景下无须额外的样式设定，开发者完全可以选择不添加 <style> 标签。这种灵活的设计，既避免了不必要的代码冗余，又赋予了开发者自主决定样式复杂度的权利，有助于提升开发效率，构建更为简洁高效的 Vue 应用。

值得关注的是，Vue 还提供了一项极为实用的功能——在 <style> 标签上添加 scoped 属性。一旦启用该属性，样式便被限定在当前组件内部，仅对组件内的元素生效。这一特性有效规避了样式对全局空间的干扰，极大地增强了样式的模块化程度，让样式的维护工作变得更为轻松高效。开发人员能够更专注、更便

捷地管理每个组件的样式，进而打造出结构清晰、易于维护的 Vue 应用。

3. 逻辑

在 Vue 组件里，逻辑部分承担着关键职责。它依靠 JavaScript 代码来处理组件的数据，并执行相应的业务逻辑，驱动组件的正常运转。这些 JavaScript 代码统一编写在 <script> 标签内。按照 Vue 单文件组件的规范要求，每个组件只能存在一个 <script> 标签。这一限制并非随意为之，而是为了确保组件逻辑清晰且集中，让开发者能在这唯一的 <script> 标签里，有条理地编排代码，实现数据的精准处理和业务逻辑的顺畅执行，避免因逻辑分散而导致的代码混乱，进而打造出结构明晰、易于维护的 Vue 组件。

在实际 Vue 开发过程中，有时会遇到这样的场景：某个组件通过 <template> 标签已经完整定义好了 HTML 结构，并且该组件仅作为展示使用，所有的行为交互与数据绑定操作都依赖于父组件传递。这种情况下，从理论层面来讲，该组件可以省略 <script> 标签，因为它确实不需要额外编写 JavaScript 逻辑来驱动自身。

不过，从项目整体维护与长远发展的角度考虑，作者强烈建议开发者即便在这种看似无须 JavaScript 逻辑的场景下，也保留一个 <script> 标签，可以是一个空的 <script> 标签，或者在其中至少定义一个基础的组件框架。这样做能够维持组件结构的完整性，为后续可能出现的功能扩展预留空间。例如，未来若需要为该组件添加一些简单的本地交互逻辑，有了预先保留的 <script> 标签，开发工作便能更加顺畅地开展，避免临时调整结构带来的潜在风险与麻烦。

特别值得一提的是，Vue3 重磅推出的特性——组合式 API，进一步革新了组件逻辑的编写方式。它赋予开发者更为灵活、强大的逻辑组织手段，极大地提升了在 <script> 标签内编写复杂业务逻辑的直观性与高效性。开发者通过组合式 API，能够以更为直观的方式对相关逻辑代码进行组合与复用，轻松应对复杂多变的业务场景，显著提高开发效率，为 Vue 应用的开发注入全新活力。

2.1.2 单文件组件的基本结构

Vue 单文件组件是构建 Vue 应用的基础单元，它巧妙地将模板、样式与逻辑整合于一体，形成一个高度内聚的组件模块。

（1）模板部分通过 <template> 标签来定义，类似 HTML5 中的 <body> 标签，用于构建组件结构，代码结构如下：

```
<template>
    /* 此处编写组件结构 */
</template>
```

（2）样式部分则借助 <style> 标签来设定，类似 HTML5 中的 <style> 标签，用于定制组件的视觉外观，代码结构如下：

```
<style>
    /* 此处编写组件的样式代码 */
</style>
```

（3）逻辑部分依托 <script> 标签承载，类似 HTML5 中的 <script> 标签，用于编写实现组件数据处理与业务逻辑的代码，驱动组件的正常运转，代码结构如下：

```
<script>
```

```
    /* 此处编写组件的逻辑代码 */
</script>
```

值得注意的是，在 Vue 单文件组件中，<template>、<style> 和 <script> 这三个标签的顺序并非固定不变，开发者可以根据自身的开发需求及个人习惯灵活调整，这种灵活性为开发工作带来了极大的便利，有助于打造更符合实际业务场景与个人偏好的 Vue 单文件组件。

☆【案例 2–1】在 Vue 单文件组件开发中，通过 <template> 标签搭建“我爱中国”的组件结构，通过 <script> 标签集中处理逻辑，通过 <style> 标签添加样式，运行效果如图 2-1 所示。在开发过程中，按规范组织代码，避免混乱，正确引入组件，即可在浏览器成功渲染“我爱中国”文本。

图 2-1　Vue 单文件组件示例运行效果

【案例实现】

接下来，将以实际操作的方式，直观地展示单文件组件的使用方法，具体步骤如下。

（1）切换至 VS Code 编辑器，打开项目 1 中手动创建的 Vue 项目（项目环境为 Vue，语言环境为 JavaScript）。打开 VS Code 终端，进入项目所在目录（命令格式为 cd 项目目录），具体命令如下：

```
cd .\vite-project\
```

（2）在 VS Code 终端中输入“npm run dev”命令启动项目。待项目成功启动后，在浏览器地址栏输入 URL 地址“http://localhost:5173/”，即可访问该项目。

（3）为避免项目创建时自带的样式干扰本案例的实现效果，需在 VS Code 编辑器中找到 src 文件夹，打开 style.css 文件，全选文件内的样式代码，将这些代码全部清除。

（4）在 VS Code 编辑器内，找到 src 文件夹下的 components 文件夹，打开 HelloWorld.vue 文件。进入文件后，删除其中项目自带的所有代码。随后，在空白处键入如下代码。完成代码输入后，按“Ctrl+S”快捷键进行保存，确保所做更改生效。

```
<template>
    <div>
        <p> 我爱中国 </p>
    </div>
</template>
<script></script>
<style></style>
```

（5）在 VS Code 编辑器里，定位到项目中的 App.vue 文件，此文件通常位于项目根目录下。首先，打开 App.vue 文件后，通过快捷键“Ctrl+A”选中文件内项目自带的全部代码，通过快捷键“Delete”删除这些代码。接着，在文件空白区域输入如下代码。代码输入完毕后，务必按下“Ctrl+S”快捷键，以确保所做的修改成功保存，让后续操作基于最新更改得以顺利推进。

```
<script setup>
import HelloWorld from './components/HelloWorld.vue'
</script>
<template>
    <HelloWorld></HelloWorld>
</template>
<style scoped></style>
```

【案例解析】

为防止项目自带的部分 CSS 全局样式干扰案例的 CSS 样式，需要删除这些可能造成干扰的全局样式。在 App.vue 文件中，引入位于 components 目录下的 HelloWorld.vue 组件，并加以使用。项目的入口文件 main.js 负责启动 App.vue 文件，而 App.vue 文件又会读取并展示 HelloWorld.vue 组件。所以，最终呈现在页面上实际渲染的组件为 HelloWorld。

若想为“我爱中国”添加样式，可以在 HelloWorld.vue 文件内，在 <style> 标签中添加样式代码，对 p 标签内文字的字体颜色、粗细及大小进行设定，代码如下：

```
p {
    font-size: 22px;
    font-weight: 600;
    color: red;
}
```

随后在浏览器中即可正常显示，效果如图 2-2 所示。

素养园地：领悟多元价值

图 2-2　为“我爱中国”添加样式效果

多学一招

在项目创建与运行过程中，难免会遭遇各类错误（bug）。下面将针对这些错误进行深入剖析，详细阐述错误产生的原因，并给出切实可行的解决办法。

bug：项目在运行时控制台报错且无法正常生效，从浏览器打开或者在命令行窗口操作时均出现报错，具体报错信息如下：

```
C:\Users\admin\Desktop\vue\project\vite-project>npm run dev
> vue_manual_starter@0.0.0 dev
> vite
failed to load config from C:\Users\admin\Desktop\vue\project\vite-project\vite.config.js
error when starting dev server:
Error [ERR_MODULE_NOT_FOUND]: Cannot find package 'vite' imported from C:\Users\admin\
Desktop\vue\project\vite-project\vite.config.js.timestamp-1732273722973-03bc0e593362a.mjs
```

```
    at packageResolve (node:internal/modules/esm/resolve:838:9)
    at moduleResolve (node:internal/modules/esm/resolve:907:18)
    at defaultResolve (node:internal/modules/esm/resolve:1037:11)
    at ModuleLoader.defaultResolve (node:internal/modules/esm/loader:650:12)
    at #cachedDefaultResolve (node:internal/modules/esm/loader:599:25)
    at ModuleLoader.resolve (node:internal/modules/esm/loader:582:38)
    at ModuleLoader.getModuleJobForImport (node:internal/modules/esm/loader:241:38)
    at ModuleJob._link (node:internal/modules/esm/module_job:132:49)
```

上述报错信息表示，当前项目缺少 vite 依赖，为使项目正常运行，需安装该组件。可以通过 npm 重新安装对应的包。完成安装后，重新运行项目，就能顺利解决当前困扰。

```
C:\Users\admin\Desktop\vue\project\vite-project>npm install vite
added 30 packages in 4s
4 packages are looking for funding
    run npm fund for details
C:\Users\admin\Desktop\vue\project\vite-project>npm run dev
> vue_manual_starter@0.0.0 dev
> vite
    VITE v5.4.11  ready in 440 ms
        Local:   http://localhost:5173/
        Network: use --host to expose
        press h + enter to show help
```

2.2 模板语法与响应式系统

MVVM，即 Model-View-ViewModel，是一种专门用于简化用户界面（UI）与业务逻辑开发流程的软件架构模式。在构建应用程序时，它把程序的主要构成划分为三个关键部分：Model（模型）、View（视图）及 ViewModel（视图模型）。这种划分方式最大的优势在于实现了清晰的职责界定。Model 主要负责管理应用的数据与业务规则；View 专注于呈现用户界面，也就是用户能直接看到和交互的部分；ViewModel 则充当 View 和 Model 之间的桥梁，协调两者间的数据流动与交互逻辑。通过这种架构模式，开发人员能够更高效地开发和维护应用，让代码结构更清晰，协作更顺畅。Vue 作为一款基于 MVVM 架构的前端框架，以双向数据绑定为核心特性，为前端开发注入全新活力，有力推动开发流程朝着规范化、系统化方向迈进。

设想在一个页面中，存在大量需要变更的数据，且这些数据散布于页面各处，若运用原生 JavaScript 来处理，代码编写会变得极为复杂，充斥着大量重复且烦琐的操作，不仅开发效率低下，而且后续维护和调试也困难重重。反观 Vue，借助其强大的数据绑定功能，开发者只需专注于数据的管理与更新，Vue 会自动根据数据变化，高效且精准地同步更新页面视图，反之亦然。这种便捷性极大地简化了开发过程，显著提升开发效率，让前端开发工作变得更加轻松高效。

2.2.1 数据绑定用法

1. 初识数据绑定

Vue 凭借独特的数据绑定机制，巧妙地实现了数据与页面内容的解耦，真正做到了以数据驱动视图更新。这一特性在外卖平台的开发中优势尽显。

以某外卖平台为例，平台需要展示海量且形式丰富的图文菜品信息，每道菜品又对应独立的点餐详情页。若开发者为每一个菜品详情页都逐一编写独立代码，不仅工作量巨大，开发效率低下，而且后期维护难度极高，显然并非明智之举。

在 Vue 的助力下，更为高效实用的做法是，开发者只需精心设计一个通用的详情页模板。在实际应用中，通过动态地变更页面所绑定的数据，就能轻松实现不同菜品详情的展示。例如，当用户单击“宫保鸡丁”菜品时，模板绑定的宫保鸡丁相关数据，如菜品图片、食材介绍、价格等，便会即刻驱动页面呈现出对应的菜品详情；切换至“麻婆豆腐”时，仅需更新绑定数据，页面视图便随之高效精准地更新，展示出麻婆豆腐的详细信息。这种数据绑定机制极大地简化了开发流程，显著提升了开发效率，让外卖平台的开发工作变得更加轻松且高效。

在 Vue 框架搭建的开发体系里，独特的开发模式展现为将页面所需数据从视图层精准抽离，集中放置于组件的逻辑层，实施高效管理。开发者在逻辑层编写各类逻辑代码，对数据进行灵活操控。一旦数据因业务逻辑执行发生任何变动，Vue 强大的自动侦测机制便会即刻捕捉到这些变化。基于数据绑定原理，Vue 会迅速响应，实时且精准地更新与之绑定的页面内容，确保页面状态始终与数据保持同步。这种开发方式，一方面极大地减少了开发者手动更新页面的烦琐操作，显著提升开发效率；另一方面，数据与视图的清晰分离，让代码结构更加清晰，极大地增强了代码的可维护性，为后续项目的持续迭代与优化提供了坚实保障。

2. 定义数据

在 Vue 开发过程中，组件状态数据的声明通过 data 选项实现。这个 data 选项并非普通数据结构，而是一个特殊函数，其功能是返回一个包含各类数据的对象，这些数据恰恰是组件正常运行所必需的。Vue 框架拥有强大的底层机制，能够将 data 选项返回对象里的所有数据转换为响应式数据。这意味着一旦这些数据因业务逻辑执行等原因发生任何变动，与之绑定的视图区域会自动触发更新流程，无须开发者手动干预。比如，在一个商品展示组件中，商品的名称、价格、库存等数据在 data 选项内声明，当商品库存数据因用户下单而减少时，Vue 会迅速捕捉这一变化，自动刷新页面上展示库存的区域，确保用户看到的始终是最新数据，极大地提升了开发便利性与用户体验。

data 选项的定义：存在这样一个函数，它的作用是返回一个囊括数据的对象。在 Vue2 或 Vue3 中，每个组件实例在初始化时，都会调用这个特定的 data() 函数。值得注意的是，每次调用 data() 函数，都会为对应的组件实例生成独立的数据任务，这些数据与该组件实例紧密关联，确保了不同组件实例间数据的隔离与独立，为组件的个性化功能实现和状态管理提供了有力支撑。示例代码如下：

```
    data() {
        return {
            itemName: '人工智能与科技强国应用研修班',
            itemDescription: '深入探究人工智能在科技强国战略中的核心应用，掌握前沿技术，为国家科技发展添砖加瓦！ ',
            cost: 299
        }
    }
```

☆【案例 2–2】Vue 的数据绑定机制堪称其核心亮点，它能巧妙地达成数据与视图的动态绑定。具体而言，开发者将数据与视图相关联后，Vue 会自动监控数据的变化。一旦数据发生变动，如将原本设定的普通文本数据更新为“科技引领发展，创新铸就强国，让我们携手推动科技进步，为实现科技强国目标而努力！”，Vue 便会依据其数据绑定规则，迅速响应这一变化，自动将更新后的数据实时呈现在对应的页面视图中（见图 2-3）。这种数据驱动视图更新的方式，生动地体现了 Vue 以数据为核心驱动力的特性，极大简化了前端开发中数据与视图同步更新的复杂流程，让页面展示的内容能随着数据的动态变化而及时、精准地调整，为用户带来流畅且实时交互的体验。

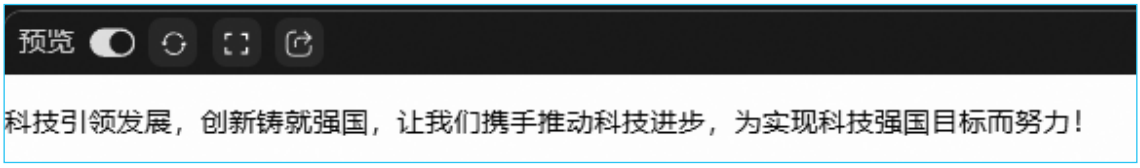

图 2-3　数据绑定示例运行效果

【案例实现】

在前端开发里，数据与页面是相互分离的架构模式。这就意味着，在数据得以在页面上渲染展示之前，必须在特定标签中预先定义好组件所需的数据。如此一来，才能确保组件在运行时，能够准确获取并处理这些数据，进而完成页面的渲染。修改项目 src 目录下的 App.vue 文件，将文件内容替换为指定的代码，代码如下：

```
    <template>
        <div>
            <p>{{ title }}</p>
        </div>
    </template>
    <script>
    export default {
        data() {
            return {
                title : '科技引领发展，创新铸就强国，让我们携手推动科技进步，为实现科技强国目标而
努力！ '
            }
        }
    }
    </script>
```

【案例解析】

data() 函数是选项式 API 写法，而在最新 Vue3 开发体系里，组合式 API 与响应式数据机制堪称核心亮点。其中，setup() 函数作为组合式 API 的关键入口，肩负着定义数据与逻辑的重任。需要注意的是，在 setup() 函数中直接声明的普通变量（如示例中的 title），默认并不具备响应式特性。若期望数据发生变化时，页面视图不仅能自动更新，还可以借助 ref() 和 reactive() 函数，将普通数据转换为响应式数据，从而实现数据与视图的高效联动，极大提升开发效率与用户体验。将上述代码修改为组合式 API 中 setup() 函数定义普通数据，代码如下：

```
    <script>
    export default {
```

```
        setup() {
            return {
                title : '科技引领发展，创新铸就强国，让我们携手推动科技进步，为实现科技强国目标而努力！'
            }
        }
    }
    </script>
```

多学一招

在 Vue3 的开发环境里，为助力开发者编写更为简洁的代码，提升整体开发效率，专门引入了 setup 语法糖这一实用功能。若想运用 setup 语法糖，只需在特定标签中添加 setup 属性即可。以下为使用 setup 语法糖的代码格式示例，通过示例能更直观地了解其使用方式。

```
<template>
    <div>
        <p>{{ title }}</p> <!-- 模板语法 -->
    </div>
</template>
<script setup>
const title = '科技引领发展，创新铸就强国，让我们携手推动科技进步，为实现科技强国目标而努力！'
</script>
```

setup 语法糖示例运行效果如图 2-4 所示。

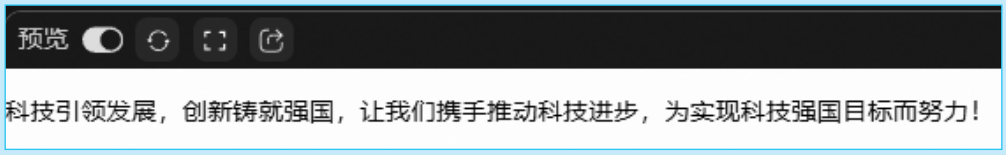

图 2-4　setup 语法糖示例运行效果

2.2.2　响应式渲染函数

Vue3 的响应式渲染函数，是开发者手中的得力工具，它赋予开发者以编程方式精准掌控组件结构与行为的强大能力。该渲染函数依托 Proxy 对象构建起深度响应式系统，能敏锐捕捉数据的任何变动。一旦数据发生改变，系统会自动触发视图更新流程，确保用户界面始终与最新数据保持同步。在执行过程中，渲染函数会返回虚拟节点（VNode），这些虚拟节点可以转化成真实的 DOM 元素。Vue 凭借其高效的转换机制，将 VNode 快速且精准地转化为真实的 DOM 元素，最终实现动态内容在页面上的流畅渲染，为用户带来优质的交互体验。

☆【案例 2–3】在 Vue3 的学习与实践中，借助实际案例能深入理解数据响应式原理。通过构建具体案例，观察数据变化时视图的更新状况，开发者能清晰分辨普通变量与响应式数据的差异。例如，当普通变量值改变时，视图不会自动更新；而响应式数据一旦变动，视图便会实时刷新。同时，通过案例实操，开发者能够熟练掌握 JavaScript 语句的使用，将数据转换为响应式数据，实现数据变化驱动视图自动更新的效果，切实提升 Vue3 开发技能。响应式渲染示例运行效果如图 2-5 所示。

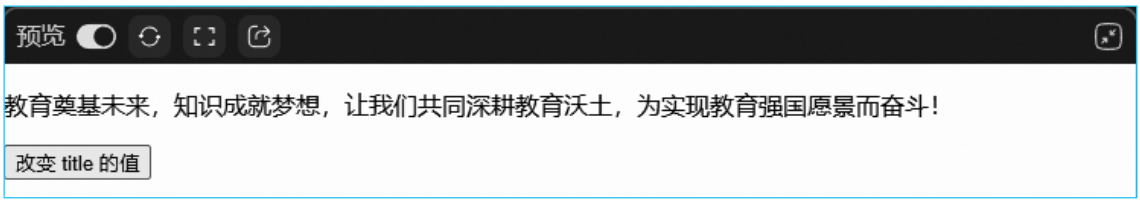

图 2-5 响应式渲染示例运行效果

【案例实现】

在案例 2-2 中，已完成了数据的定义，并成功将其渲染至页面，但尚未对数据进行修改操作。接下来，将借助实际案例展开深入测试，通过编写 methods 中的方法，实现对数据的精准控制与动态改变，以此进一步探究 Vue3 的强大功能，具体代码如下：

```
<template>
    <div>
        <p>{{ title }}</p> <!-- 模板语法 -->
        <button @click="buttonClick"> 改变 title 的值 </button>
    </div>
</template>
<script>
export default {
    data() {
        return {
            title: '科技引领发展，创新铸就强国，让我们携手推动科技进步，为实现科技强国目标而努力！ '
        };
    },
    methods: {
        buttonClick() {
            this.title = '教育奠基未来，知识成就梦想，让我们共同深耕教育沃土，为实现教育强国愿景而奋斗！ ';
        }
    }
};
</script>
```

当用户单击按钮后，页面上的 title 未出现预期变化，查看控制台，也并未发现报错信息。尽管从程序运行角度看，该变量的值已经改变，但 Vue 无法自动感知这种变化并同步更新页面。为了验证变量值确实发生了改变，可在标签中运用打印函数来加以确认，代码如下：

```
<script>
export default {
    data() {
        return {
            title: '科技引领发展，创新铸就强国，让我们携手推动科技进步，为实现科技强国目标而努力！ '
        };
    },
    methods: {
```

```
            buttonClick() {
                this.title = '教育奠基未来，知识成就梦想，让我们共同深耕教育沃土，为实现教育强国愿景而奋斗！ ';
                console.log(this.title);
            }
        }
    };
    </script>
```

【案例解析】

运行程序后能够清晰地观察到控制台成功输出了 title 更新后的值，但页面上通过模板语法展示 title 的区域却未发生任何变化。这一现象直观表明，当 title 的值在程序中发生改变时，页面并不能自动进行同步更新渲染。这背后反映出 Vue3 数据更新机制中，若未正确设置数据的响应式，数据变动与页面视图更新之间就会出现脱节情况。

若期望达成页面与数据的同步更新，关键在于实施响应式数据绑定，即把数据妥善定义为响应式数据（ref）。在 Vue 的技术框架中，为助力开发者轻松实现这一目标，精心提供了 ref() 函数、reactive() 函数、toRef() 函数及 toRefs() 函数，如表 2-1 所示。借助这些函数，能够将普通数据巧妙转换为响应式数据，从而确保数据一旦发生变动，页面视图便能自动、及时地更新。接下来，将针对这几个函数，逐一展开详细解析，介绍响应式数据的定义与运用技巧。

表 2-1　响应式渲染函数

函数	描述
ref()	用来创建响应式的基本数据类型或对象（访问对象时需通过 .value），适用于需要响应式的数据类型场景
reactive()	将普通对象转变为响应式对象，该对象的所有属性都会具备响应式特性
toRef()	能够把响应式对象中的某个属性转换为 ref，便于单独处理该属性的响应状态
toRefs()	可以将响应式对象的所有属性都转换为 ref，常用于解构赋值操作，同时还能维持属性的响应式

1. ref() 函数

在 Vue3 开发环境里，ref() 函数是实现数据响应式转换的得力工具。它的核心作用是将传入的普通数据整体转换为响应式数据。具体来说，该函数接收普通数据作为参数，经过内部处理后，会返回转换完成的响应式数据。一旦数据被 ref() 函数成功转换为响应式，数据的任何变动都能被 Vue3 的响应式系统敏锐捕捉，进而驱动页面视图自动更新，有效解决数据变动与页面显示不同步的问题。使用 ref() 函数定义响应式数据的语法格式如下：

```
响应式数据 = ref(' 数据 ')
```

如果需要更改响应式数据的值，可以使用如下语法格式进行修改：

```
响应式数据 .value = 新值
```

提示

JavaScript 具有自动分号插入（Automatic Semicolon Insertion，ASI）的机制。它会在代码解析过程中自动插入分号，以分隔语句。这意味着即使没有显式地写分号，JavaScript 引擎也会自动添加。

☆【案例 2–4】使用 ref() 函数定义响应式数据，从而实现数据变化时视图的自动更新，效果如图 2-6 所示。

图 2-6　ref() 函数示例运行效果

【案例实现】

修改项目 src 目录下 App.vue 文件内容，将文件内容替换为指定的代码，键入代码：

```
<template>
    <div>
        <h1>{{ message }}</h1>
        <button @click="changeMessage"> 点击开启科技新征程 </button>
    </div>
</template>
<script setup>
import { ref } from 'vue';
// 创建科技主题的 ref 对象
const message = ref(' 科技强国 创新先行 ');
// 定义科技主题的交互方法
const changeMessage = () => {
    message.value = ' 以科技赋能发展 让创新引领未来 ';
    console.log(' 科技强国征程已开启！ ');
};
</script>
```

【案例解析】

此代码通过 Vue3 的 <script setup> 语法编写。借助 ref() 函数创建了一个名为 message 的响应式数据，初始值是“科技强国 创新先行”。当单击按钮时，changeMessage 方法会被触发，从而改变 message 的值，页面内容也会随之更新。

2. reactive () 函数

在 Vue 中，reactive() 函数是构建响应式数据的得力工具，它能够将普通的对象或者数组转换为响应式数据。只需把想要转换的普通对象或数组作为参数传递给 reactive() 函数，便能轻松达成这一效果。使用 reactive() 函数定义响应式数据的格式如下：

```
响应式数据对象或数据 = reactive( 普通的对象或数组 )
```

☆【案例 2–5】使用 reactive() 函数定义响应式数据，当数据发生变动时，无须手动操作，视图便会自动

随之更新，运行效果如图 2-7 所示。

图 2-7　使用 reactive() 函数定义响应式数据示例运行效果

【案例实现】

修改项目 src 目录下 App.vue 文件内容，将文件内容替换为指定的代码，代码如下：

```
<template>
    <div>
        <h2>{{ message.content }}</h2>
        <button @click="updateMessage"> 切换科技标语 </button>
    </div>
</template>
<script setup>
import { reactive } from 'vue';
const message = reactive({
    content: ' 科技强国，创新引领发展新征程 ',
    isSwitched: false
});
const updateMessage = () => {
    if (message.isSwitched) {
        message.content = ' 科技强国，创新引领发展新征程 ';
    } else {
        message.content = ' 科技赋能未来，建设世界科技强国 ';
    }
    message.isSwitched = !message.isSwitched;
    console.log(' 科技标语切换成功 !');
    console.log(' 当前标语切换状态 :' + message.isSwitched.toString());
};
</script>
```

完成代码修改并保存后，启动项目。单击页面上的按钮，会发现页面中渲染的值已经发生变化。同时，还可以在浏览器的控制台查看更新后的值（在页面空白处右击，执行“检查”→“控制台”命令）。

【案例解析】

这段代码通过 reactive() 函数创建了一个响应式对象 message，它包含 content 和 isSwitched 两个属性。content 用于存储要显示的文本内容，isSwitched 用于记录内容是否已经切换过。同时，在 <template> 中，使用“{{ message.content }}”来显示当前的文本内容，并绑定了一个按钮的点击事件“@click="updateMessage"”。在 updateMessage() 函数中，根据 message.isSwitched 的值来决定 message.content 的内容。如果 isSwitched 为 true，则将 content 设置为初始内容，否则设置为切换后的内容。然后切换 isSwitched

的值，并在控制台打印提示信息。这样，每次单击按钮，文本内容就会在初始内容和切换后的内容之间切换，并且控制台会输出切换成功的提示和当前的切换状态。

☆【案例 2–6】借助 reactive() 函数来创建响应式数组。当这个数组里的数据发生改变时，与之关联的视图能够自动进行更新，无需手动干预，运行效果如图 2-8 所示。

图 2-8　使用 reactive() 函数创建响应式数组示例运行效果

【案例实现】

修改项目 src 目录下 App.vue 文件内容，将文件内容替换为指定的代码，代码如下：

```
<template>
    <div>
        <p>{{ title }}</p>  <!-- 模板语法 -->
        <button @click="buttonClick"> 修改标语 </button>
    </div>
</template>
<script setup>
import { reactive } from 'vue'
const title = reactive([' 乡村振兴 ',' 和谐社会 ',' 振兴中华 '])
const buttonClick = ()=> {
    title[0] = ' 科技强国 '  // 数组默认索引值从 0 开始
    console.log(' 修改成功 !')
}
</script>
```

完成代码编辑并保存后，启动程序运行项目。单击页面上的按钮，页面中渲染的值会立即改变。同时，在浏览器的开发者控制台中，能够清晰地看到更新后的值，直观呈现代码逻辑运行效果。

【案例解析】

此段代码充分展示了 Vue3 的先进特性。在 Vue3 里，借助 <script setup> 语法与组合式 API，利用 reactive() 函数成功创建了响应式数组。在 <template> 部分，运用插值表达式将数组的首个元素呈现出来。页面上有一个按钮，单击会触发 buttonClick 事件处理器，对数组的第一个元素的值进行更新。得益于 Vue3 强大的响应式系统，一旦数组元素的值发生改变，视图会自动更新，实时呈现出变化后的结果。这整个过程很好地体现了 Vue3 对响应式数组良好的支持及高效的模板渲染机制。

3. toRef() 函数

toRef() 函数的作用是把一个响应式对象中的单个属性转换为响应式数据。使用 toRef() 函数来定义响应式数据的代码格式如下：

```
响应式数据 = toRef( 响应式对象 , ' 属性名 ');
```

☆【案例 2–7】运用 toRef() 函数将响应式对象的某个属性单独提取出来，转换为独立的响应式数据。这样一来，当这个属性的数据发生变化时，与之关联的视图会自动更新，无须额外的手动操作，运行效果如图 2-9 所示。

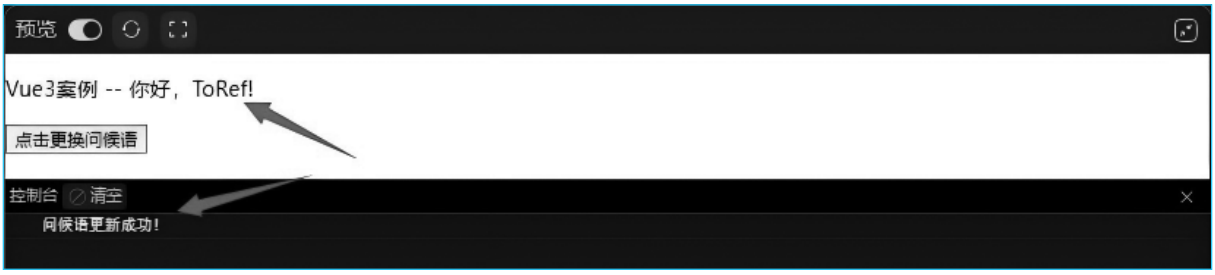

图 2-9　toRef() 函数示例运行效果

【案例实现】

修改项目 src 目录下 App.vue 文件内容，将文件内容替换为指定的代码，代码如下：

```
<template>
    <div>
        <p>{{ obj.title }} -- {{ msg }}</p>  <!-- 显示响应式对象属性和 toRef 引用值 -->
        <button @click="updateMessage"> 点击更换问候语 </button>
    </div>
</template>
<script setup>
import { reactive, toRef } from 'vue'
// 创建响应式对象
const obj = reactive({
    title: 'Vue3 案例 ',
    content: ' 你好，Vue!'
})
// 使用 toRef 创建对 obj.content 属性的引用
const msg = toRef(obj, 'content')
// 按钮点击事件处理函数
const updateMessage = () => {
    msg.value = ' 你好，ToRef!'  // 通过引用修改原始对象属性
    console.log(' 问候语更新成功 !')
}
</script>
```

【案例解析】

在上述代码中，使用 reactive() 函数创建包含 title 和 content 属性的响应式对象 obj，并通过“toRef(obj,'content')”创建对 obj.content 的独立引用 msg，保持响应式连接；在 <template> 标签中，“{{obj.title}}”直接显示响应式对象属性，“{{msg}}”显示通过 toRef() 函数创建的引用值（内部会自动代理 value 属性）。单击按钮时，调用 updateMessage 方法，通过 msg.value 修改引用值，会同步更新原始对象 obj.content，模板会自动重新渲染显示新值。

4. toRefs () 函数

toRefs() 函数的用途是把响应式对象里的所有属性都转换为响应式数据，最后以对象的形式返回这些转