第6章 前端工程化

学习Vue.js 3组件化时提到了SFC开发模式。默认情况下，不能直接使用单文件组件来编写组件，因为浏览器不认识SFC（.vue）文件。因此，需要构建一个支持SFC开发的Vue.js 3环境。我们可以使用webpack 或者 Vite 来构建。目前，webpack 被广泛使用，但使用 Vite 的人也越来越多了。无论使用webpack 还是 Vite 构建，都属于前端工程化。在介绍前端工程化前，先看看本章节源代码的管理方式，目录结构如下：

```
VueCode
├── ......
├── chapter06
|   └── 01_vuecli_demo # Vue CLI 构建的项目
|   └── 02_createvite_demo # create-app 构建的项目
```

6.1 前端快速发展史

无论是专业的开发者还是接触互联网的普通人，都能深刻感受到Web前端发展之快。而对于专业的开发者来说，体会更加深刻，例如。

- 从后端渲染的JSP、PHP，到原生JavaScript，再到jQuery，再到目前主流的Vue.js 3、React、Angular框架。
- 从原来ES5语法，到ES6、7、8、9、10，再到TypeScript，以及从简单的CSS，到预处理器Less、Scss等。

我们可以简单的概述一下前端发展的几个阶段。

- Web早期：也是互联网发展早期，前端只负责写静态页面，纯粹的展示功能，JavaScript的作用只是进行一些表单的验证和增加特效。当然，为了在页面中动态填充一些数据，也出现了JSP、ASP、PHP等开发模式。
- Web近期：随着AJAX技术的诞生，前端不仅仅只是展示页面，也可以管理数据以及和用户进行互动。随着用户交互、数据交互的需求增多，也让jQuery 这样优秀的前端工具库大放异彩。
- 现代Web：现代Web前端开发变得更加多样化和复杂化。比如多样化的前端支持开发PC Web页面、移动端Web页面、小程序、公众号和App，都属于前端开发范畴。当然在开发模式上，也面临一系列复杂性的问题。

现代Web前端开发目前面临一系列复杂性的问题，比如。

- 项目需要通过模块化的方式来开发。
- 项目需要使用一些高级特性来加快开发效率或安全性，比如通过ES6+、TypeScript开发脚本逻辑，通过Sass、Less等来编写CSS样式。
- 项目开发过程中需要本地服务，能实时监听文件变化并反映到浏览器上，提高开发效率。
- 项目打包部署时，需要对代码进行压缩、合并以及其他相关的优化。

对于大部分的Vue.js 3、React、Angular开发者来说，并不会遇到上述所描述的问题，因为大部分的人都是借助对应框架提供的脚手架（CLI）来创建工程化的项目。例如：Vue CLI、create-react-app、Angular CLI等脚手架，这些脚手架默认已经帮助我们解决了上述所描述的问题，它们底层也是基于webpack 构建工具来实现的。然而，这些通过脚手架创建的项目，通常称为前端工程化项目。

6.2 认识webpack

在Vue.js 3中，webpack 是一个非常重要的工具。根据官方定义，webpack是一个静态的模块化打包工具，用于打包现代的JavaScript应用程序。

> 官方的定义：webpack is a *static module bundler* for modern JavaScript applications.

上述解释可能有些晦涩难懂，下面我们来总结一下webpack的几个重要特点。

- 打包（bundler）：webpack是一个JS编写的打包工具，可将应用程序打包成可在浏览器中运行的JS文件。
- 静态的（static）：webpack可以将代码打包成静态资源，包括JS、CSS、图片等，如图6-1所示。
- 模块化（module）：webpack支持多种模块化开发方式，包括ES Module、CommonJS、AMD等。
- 现代的（modern）：webpack专门解决现代前端开发面临各种各样复杂的问题，比如：支持ES6+等。

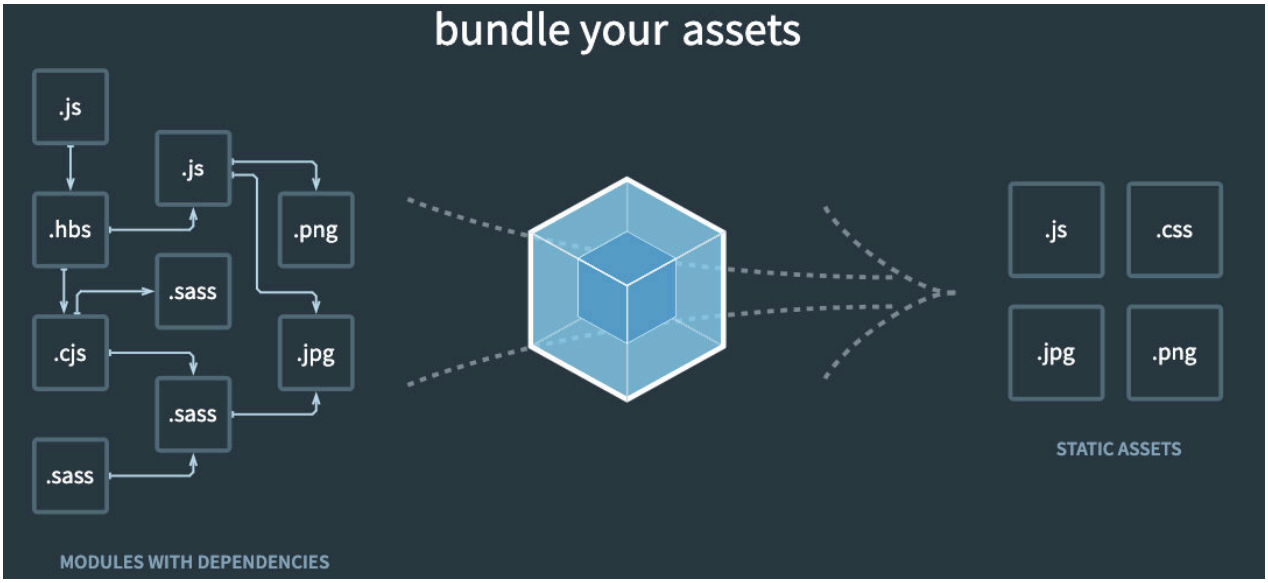

图6-1 webpack

认识了webpack之后，下面来看看webpack的使用，在Vue.js 3中使用webpack通常有两种方式。

1. 使用脚手架，例如Vue CLI脚手架，该脚手架是基于webpack来实现的。Vue CLI已经帮助我们编写好了各种配置，开箱即用。
2. 使用webpack从零搭建Vue.js 3开发环境，这个需要有webpack基础，需要自行编写各种webpack等配置。

初学者建议使用脚手架来创建项目，因为脚手架使用简单，零配置就可以搭建好Vue.js 3的工程化项目。并且在企业中，通常也是采用脚手架来创建项目的。下面将介绍使用Vue CLI搭建Vue.js 3项目。

6.3 Vue CLI脚手架

6.3.1 认识Vue CLI

脚手架实际上是建筑工程中的一个概念，如图6-2所示。在软件工程中，用于帮助搭建项目的工具也被称为脚手架，例如：Vue CLI、create-app等。

图6-2 建筑中的脚手架

在真实的开发中，我们通常会使用脚手架来创建项目。例如Vue.js 3项目会使用Vue CLI脚手架来创建。

- CLI全称是Command-Line Interface，翻译为命令行界面。
- 可以通过CLI选择项目的配置，创建Vue.js 3项目。
- Vue CLI已内置了webpack相关的配置，我们不需要从零开始编写webpack配置。

6.3.2 安装Node.js

在使用webpack和Vue CLI脚手架之前，必须先在计算机上安装Node.js环境。因此，我们需要先安装Node.js，并自动安装npm包管理器。下面是Node.js的安装步骤。

1. 访问Node.js官网（https://nodejs.org/）下载安装包，如图6-3所示。
    - 或下载与本书相同的Node.js版本：https://nodejs.org/dist/v16.13.1/
    - 本书用的Node.js版本是v16.13.1，npm版本是 8.1.2

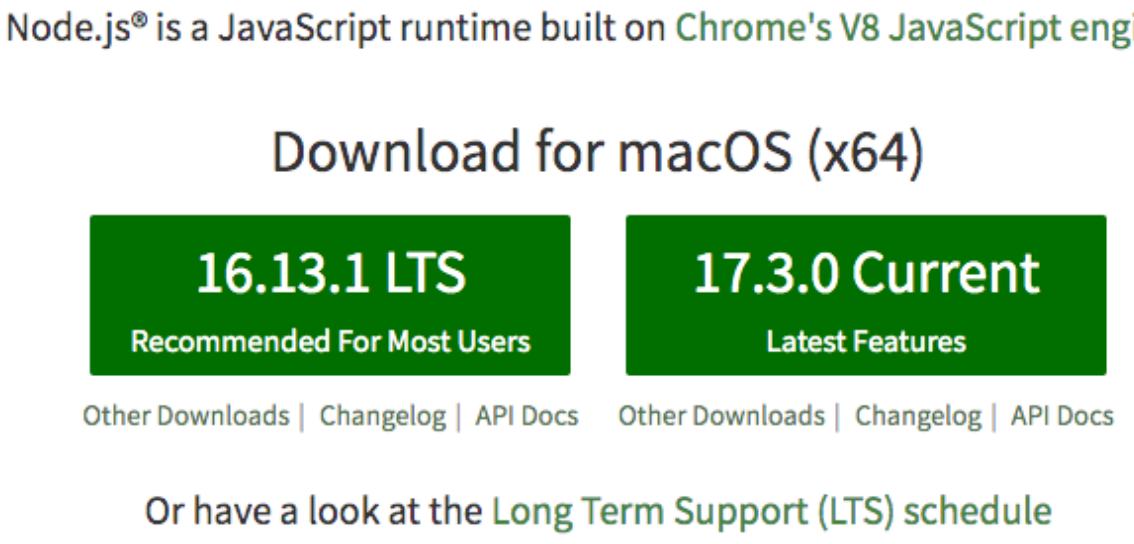

图6-3 下载Node.js

2. 在Windows系统下安装。

- 下载适用于 Windows 的 `node-v{version}-x86.msi` 程序(本书要求Node.js版本需大于16)。
- 下载后，运行安装程序 (`node-v{version}-x86.msi`)。默认单击下一步安装即可。

3. 在macOS系统下安装。
   - 下载适用于 macOS 的`node-v{version}.pkg`程序。
   - 下载后，双击`node-v{version}.pkg`程序安装。默认单击下一步安装即可。
4. 安装完成后，打开终端或命令行界面，输入以下命令检查Node.js和npm是否安装成功：

```
node -v # 查看Node.js版本
npm -v # 查看npm版本
```

Windows系统需打开`cmd`终端，macOS系统需打开`terminal`终端。如图6-4所示，是macOS系统打开的终端。

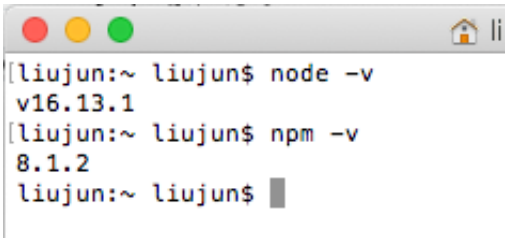

图6-4 下载Node.js

5. 现在，我们已经在计算机上安装好了Node.js环境，接下来可以开始安装Vue CLI脚手架了。

> 提示：Windows可用 nvm 工具管理Node.js版本；macOS可用 n 工具管理，这些命令行工具需自行安装。

6.3.3 安装Vue CLI

当安装好Node.js之后，我们就可以使用npm来安装Vue CLI。在终端中输入以下命令即可全局安装Vue CLI：`npm install -g @vue/cli`。这里的`-g`表示全局安装Vue CLI，这样以后在任意路径下打开终端，都可以使用`vue`命令来创建Vue.js 3项目（推荐大家安装与本书一样`5.0.8`版本）。

1.全局安装Vue CLI。

```
# 2)方式一(推荐)：在终端执行下面命令，来安装指定版本
npm install @vue/cli@5.0.8 -g

# 1)方式二：在终端执行下面命令，来安装最新版本
npm install @vue/cli -g
```

2.升级 Vue CLI 到最新版本（可选）。

```
npm update @vue/cli -g
```

3.通过`vue`的命令来创建项目。

```
vue create 项目的名称
```

4.安装完`Vue CLI`之后，可以在终端查看版本号，如图6-5所示。

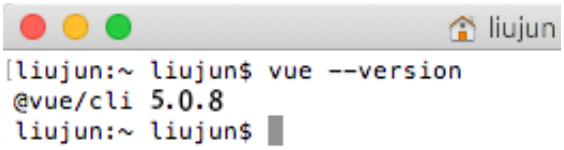

图6-5 查看Vue CLI版本号

6.3.4 Vue CLI新建项目

安装好 Node.js 和Vue CLI后，就可以使用Vue CLI脚手架来创建Vue.js 3项目。建议在创建Vue.js 3项目之前，先安装VS Code的Volar插件，为`.vue`文件提供语法高亮等支持（插件安装详见1.5.3节）。

注意：Vetur插件也可为.vue文件提供语法高亮等，不过它是Vue.js 2的产物。Vue.js 3官网推荐Volar插件。

安装好Volar插件后，下面我们开始创建一个Vue.js 3项目，具体步骤如下。

1.使用Vue CLI的`vue`命令新建一个`01_vuecli_demo`的Vue.js 3项目。

```
vue create 01_vuecli_demo
```

首先，使用 VS Code 打开 chapter06 文件夹。接着，在VS Code中单击顶部的 "Terminal" 选项，选择"New Terminal"来新建一个终端。

当终端弹出后，然后输入命令"vue create 01_vuecli_demo"来创建一个新的"01_vuecli_demo"项目。

最后，按键盘向下键选择`"Manually select features"`手动选择项目所需的功能。 如图6-6所示。

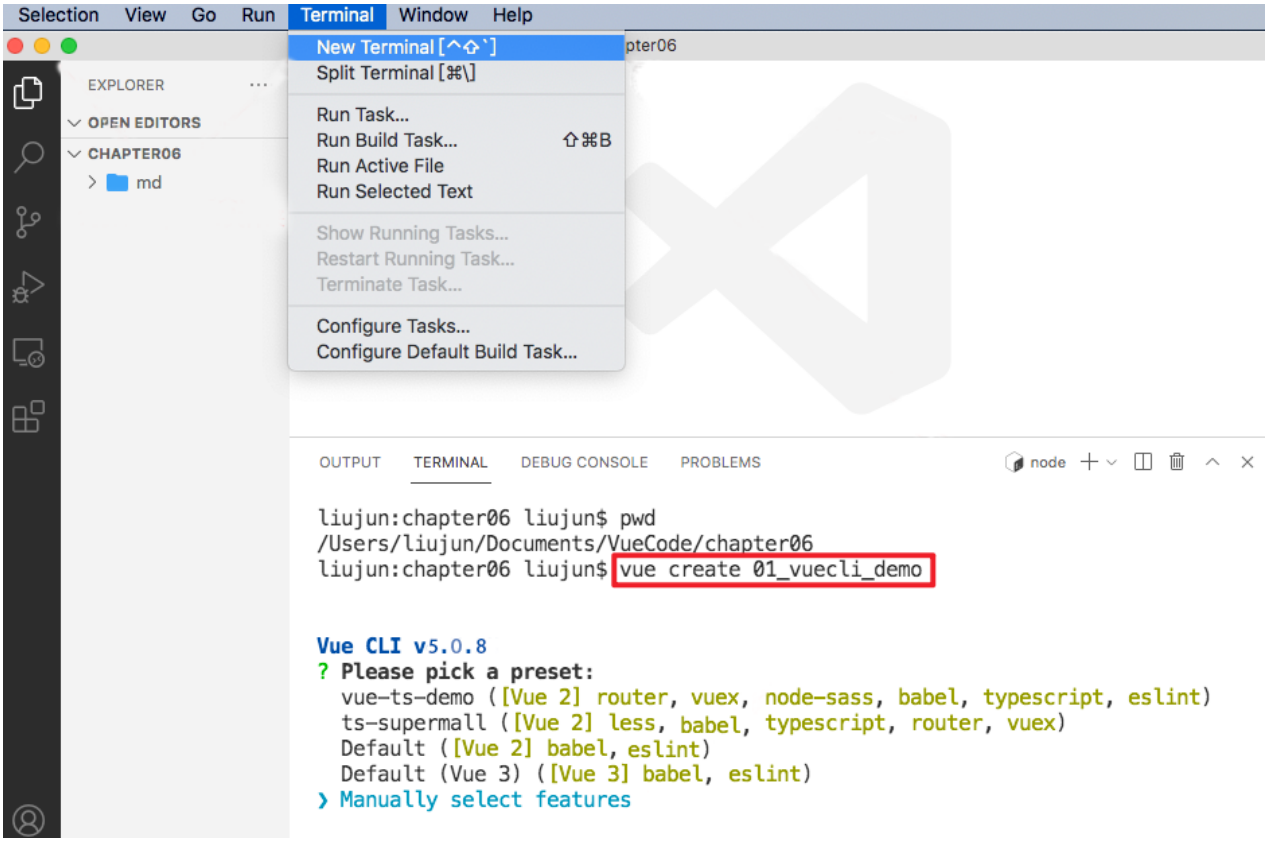

图6-6 Vue CLI新建Vue.js 3项目

Vue CLI 脚手架默认提供三个预设。

- Default (**Vue.js 2** babel, eslint)： 新建Vue.js 2默认的项目，项目集成 Babel，ESLint 插件。
- Default (**Vue.js 3** babel, eslint)： 新建Vue.js 3默认的项目，项目集成 Babel，ESLint 插件。
- Manually select features： 新建项目，手动选择项目所需的功能，如是否需要Babel，ESLint插件。

注意：上图 vue-ts-demo 和 ts-supermall 是作者之前自定义预设（有关于自定义预设见第5步）。

2.Manually select features 手动选择所需的功能。

为了让新建的项目更简单，这里只选择了Babel选项，如图6-7所示。其他选项暂时可以不用管，后面用到时再给大家讲解。

> 提示："选中"和"取消选中"是按空格键，上下移动是按上下键，确认是按Enter键。

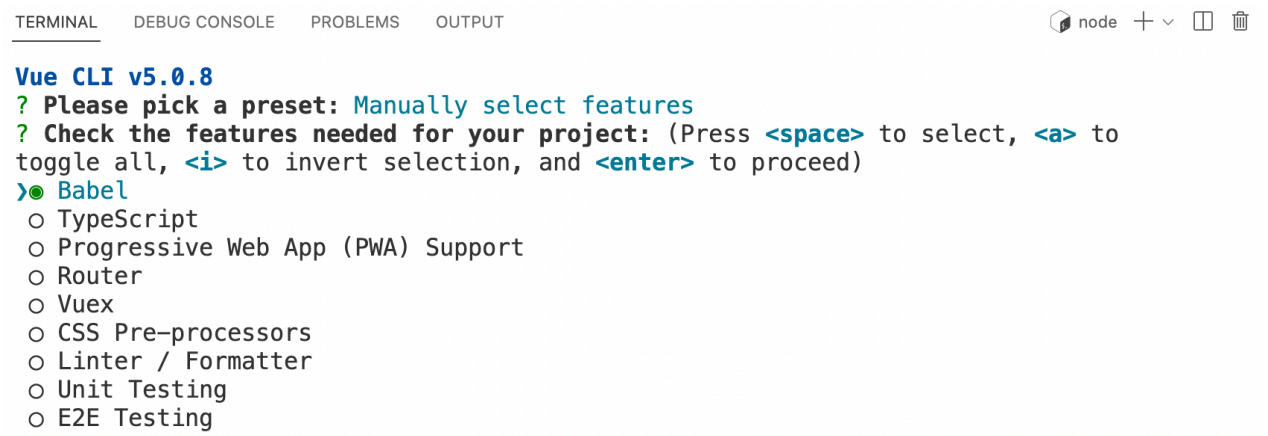

图6-7 选择项目所需功能

以下是 Vue CLI 新建项目时可供选择的功能。

- Babel：是否使用Babel作为JavaScript编译器，结合插件可将ES6、7、8、9、10等语法转换为ES5语法。
- TypeScript：是否使用TypeScript。
- Progressive Web App (PWA) Support：是否支持PWA，PWA是渐进式Web应用，一种无需安装的网页应用，具有与原生应用相同的用户体验优势。
- Router：是否默认集成路由，路由相关知识见第12章，用于处理页面的跳转。
- Vuex：是否默认集成Vuex状态管理，Vuex相关知识见第13章，用于在多个组件间共享数据。
- CSS Pre-processors：是否选用CSS预处理器，即常用的Less、Scss、Stylus预处理器。
- Linter / Formatter：是否选择ESlint对代码进行格式化限制。
- Unit Testing：是否添加单元测试。
- E2E Testing：是否添加E2E测试。

3.选择Vue.js的版本。

这里选择" 3.x "的版本，如图6-8所示。

图6-8 选择Vue.js的版本

4.选择配置存放的位置

这里选择`In dedicated config files`。意思是对于Babel、ESLint等配置信息，统一放到各自独立的配置文件中，而不是都放到 package.json 文件中。如图6-9所示。

图6-9 选择配置存放的位置

Vue CLI提供了两种方式来管理项目的配置信息。

- In dedicated config files ： 将配置信息放到各自独立的文件中。
- In package.json ： 将配置信息放到package.json文件中。

5.是否保存为自定义预设

在提示"Save this as a preset for future projects?"时，输入了"y"，当然也可输入"n"，即不保存自定义预设。如果保存了预设，下次新建项目时，在第 1 步选择预设时就可以看到我们保存过的预设，比如前面看到的"vue-ts-demo"预设。

这里我们选择了"yes"保存预设后，下一步需要给预设起一个名称，这里我给预设起名为"vue3-demo"。最后按"Enter"键即可，如图6-10所示。

图6-10 保存预设

6.新建成功的提示。

执行完上述步骤，如未出现错误，并提示"npm run serve"命令，则说明成功创建Vue.js 3项目，参见图6-11。

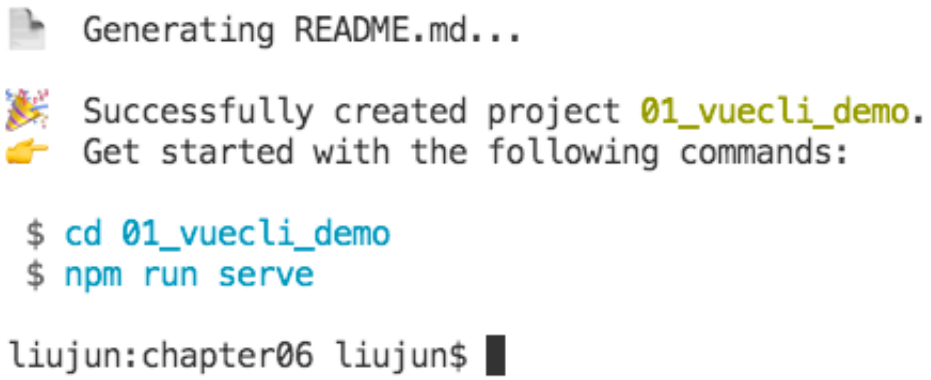

图6-11 项目新建成功

> 提示：不同版本的 Vue CLI 脚手架创建项目时，可能会有稍微区别，但是整体步骤还是一样的。

6.3.5 Vue.js 3项目目录结构

新建好Vue.js 3项目后，接着使用VS Code打开刚才新建的"01_vuecli_demo"项目。在VS Code中打开该项目后，界面如图6-12所示。

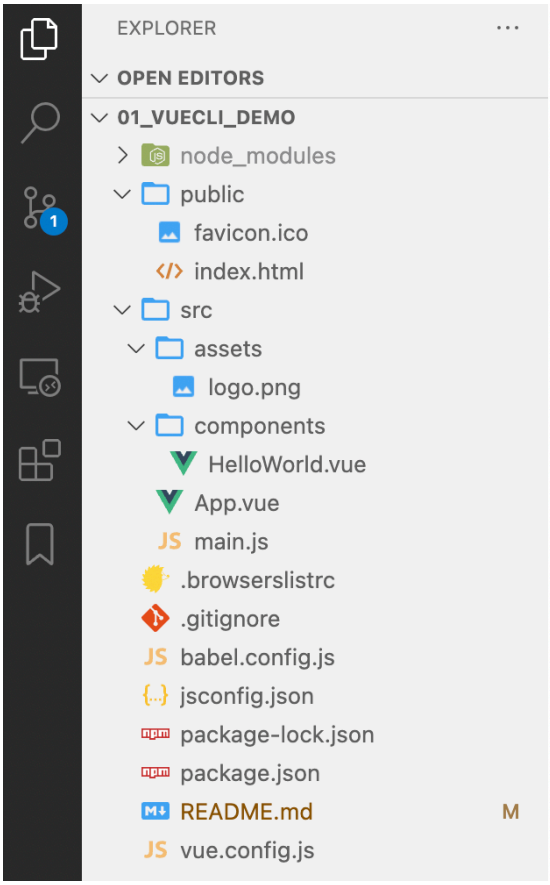

图6-12 Vue.js 3目录结构

我们可以看到这次工程化项目的目录和我们前面章节的目录完全不一样，多了很多的配置文件和目录结构。不用担心，下面我将为大家讲解该项目的目录结构。

```
01_vuecli_demo # 项目的名称( 命名建议：统一小写，多个单词使用下划线分割 )
├── node_modules  # 存放第三方依赖包(例如，执行 npm install 安装的依赖包)
├── public # 项目的一些资源(不会参与webpack打包)
|   ├── favicon.ico # 网站的 icon 图标
|   └── index.html # 网站的index.html
└── src # 项目所有代码存放的目录
    ├── assets # 存放项目的资源(例如：图片，字体，全局样式等)
    |   └── logo.png
    ├── components # 用于存放Vue.js 3组件的目录
    |   └── HelloWorld.vue # 3.名为 HelloWorld 的局部组件(即，SFC组件)
    ├── App.vue # 2.项目的App根组件
    └── main.js # 1.程序入口文件(也是打包的入口文件)
├── .browserslistrc # 兼容目标浏览器的配置文件( 可供Babel，PostCSS等插件使用)
├── .gitignore # 忽略哪些文件不需要提交到Git仓库中
├── .jsconfig.json # 此类文件，表明该目录是JavaScript项目的根目录，作用是可提供更好的代码智能提示
├── babel.config.js  # Babel插件的配置文件
├── package-lock.json # 锁定第三方安装的依赖包版本
├── package.json # 项目的描述文件(包含了项目名称，版本，开发和生成依赖，运行和打包项目的脚本等)
├── vue.config.js # Vue CLI脚手架的配置文件，比如配置：alias、devServer和configureWebpack等
```

```
└── README.md # 关于项目的说明文档
```

> 提示：Vue CLI 4.x 版本创建的项目是没有 .jsconfig.json 和 vue.config.js 文件。

6.3.6 项目的运行和打包

新建好项目后，接下来我们看看如何运行和打包项目。Vue CLI脚手架已经在 package.json 文件中提供了相应的命令，代码如下所示：

```
{
  ......
  "scripts": {
    "serve": "vue-cli-service serve",  # 1.开发环境，启动项目的脚本。
    "build": "vue-cli-service build" # 2.生产环境，打包项目的脚本。
  },
  ......
}
```

可以看到，在`scripts`属性中定义了两个脚本。

- `serve`：是运行项目的脚本，当在终端执行`npm run serve`时，便会执行`vue-cli-service serve`来启动一个本地服务在浏览器中运行代码。
- `build`：是打包项目的脚本，当在终端执行`npm run build`时，便会执行`vue-cli-service build`来打包该项目。

接下来将分别演示如何运行和打包项目。

1.运行Vue.js 3项目，执行npm run serve命令。

先使用VS Code打开"01_vuecli_demo"项目，接着在VS Code中打开终端。在终端中运行命令`npm run serve`，最后在浏览器中输入`localhost:8080`以在浏览器中运行代码，如图6-13所示。

> 提示：在VS Code中打开终端的步骤为：单击VS Code顶部的 Terminal -> New Terminal。

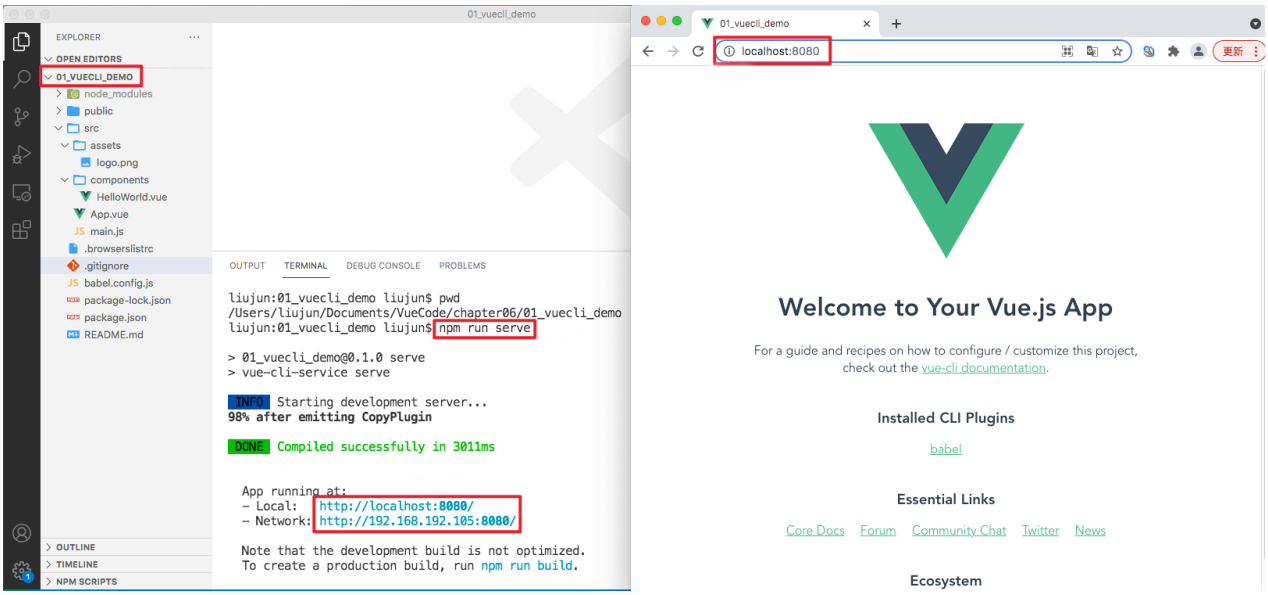

图6-13 运行Vue.js 3项目

这时，在App.vue或HelloWorld.vue组件的template中新增内容后，按Ctrl+S键保存代码，页面会自动刷新以显示新增内容。最后，关闭正在运行的Vue.js 3项目有以下几种方法。

1. 单击终端右上角的 x 按钮关闭。
2. 在Windows系统中，可以按 Ctrl + C 快捷键。
3. 在macOS系统中，可以按 Control + C 快捷键。

2.打包Vue.js 3项目，执行npm run build命令。

掌握了项目的运行和关闭后，下面来看看 Vue.js 3 项目的打包。打包生产环境的项目，也是在VS Code的终端中输入：“npm run build” 命令，打包成功后会在项目的根目录下生成一个“dist”文件夹，该文件夹就是以后线上部署的项目，如图6-14所示。

> 提示：关于项目如何部署见第18章。

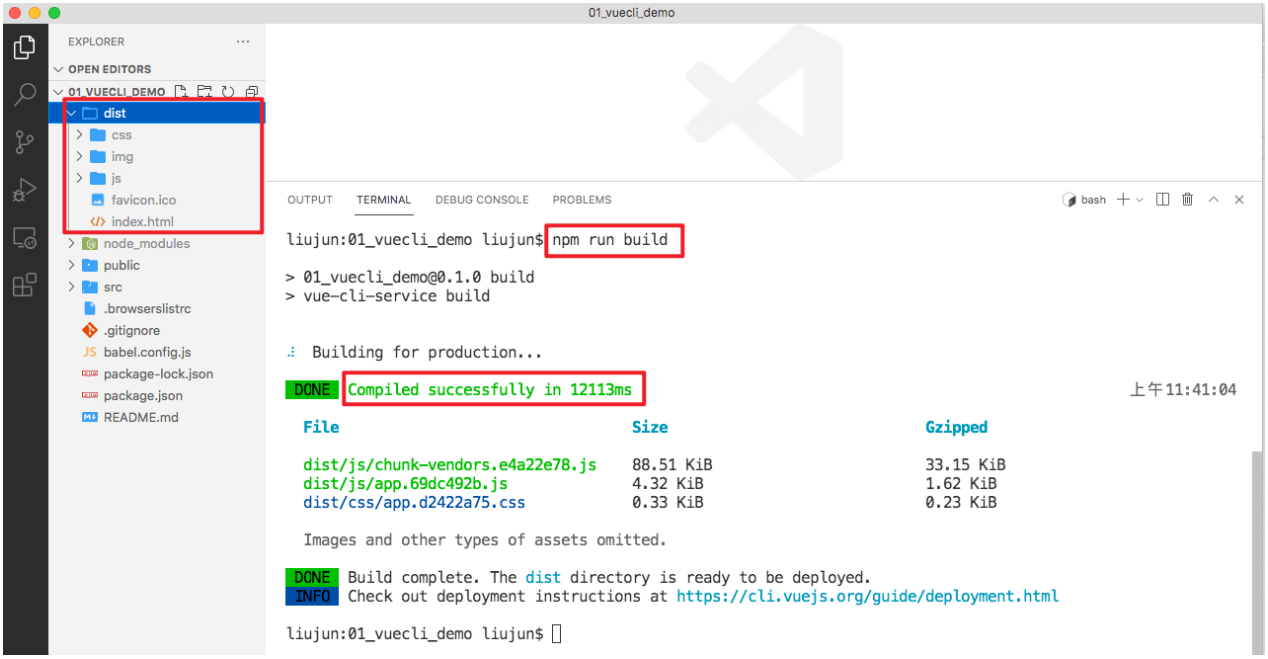

图6-14 打包Vue.js 3项目

6.3.7 vue.config.js文件解析

`VueCLI` 除了可以用来创建项目外，还提供了一些常用的配置方便我们对其进行扩展和自定义，比如：`outputDir`、`assetsDir`、`alias`、`devServer`和`chainWebpack`等。这些配置必须编写在项目根目录的 `vue.config.js` 文件中。

下面给大家介绍一些常用的配置。

1.outputDir：指定打包输出的目录名，默认是dist目录。

默认情况下，打包项目时会自动生成一个`dist`目录，如果想要修改该目录名，可以使用`outputDir`配置。

首先在`01_vuecli_demo`项目的根目录下新建`vue.config.js`文件，代码如下：

```
// vue.config.js 在Vue CLI 4.x需要手动新建该文件，Vue CLI 5.x默认已生成。
module.exports = {
  outputDir: 'build'
}
```

`outputDir` 属性指定了打包后生成目录的名为 `build`，而不是默认的 `dist`。请重新执行 `npm run build` 打包该项目。打包完成后，项目的根目录会生成一个 `build` 目录。

> 提示：outputDir属性值也支持 build/music 写法，代表将项目打包到 build/music 文件夹中。

另外需要注意的是，使用 Vue CLI 5.x 创建的项目，`vue.config.js`同样支持使用`defineConfig`宏函数，以获得更好的代码智能提示，代码如下所示：

```
// defineConfig 宏函数只支持 Vue CLI 5.x
const { defineConfig } = require("@vue/cli-service");
module.exports = defineConfig({
  // transpileDependencies: true,  // 如true，项目引用node_modules中的包也会用Babel来编译
  outputDir: "build",
});
```

可以看到，这里将`outputDir`配置写到了`defineConfig`宏函数中，效果和之前是一样的，但是这次编写配置会有智能提示功能。不过需要注意的是，该功能仅支持使用 Vue CLI 5.x 创建的项目。

2.assetsDir：指定静态资源存放目录。

我们可以通过 `assetsDir` 属性来指定项目静态资源（JS、CSS、Fonts、图片）存放的目录，该属性是相对于 `outputDir` 路径。在 `vue.config.js` 文件中进行设置，代码如下所示：

```
// 支持 Vue CLI 5.x  和 Vue CLI 4.x 创建的项目
module.exports = {
  outputDir: 'dist',
  assetsDir: 'static'
}
```

上面配置`assetsDir: "static"`，意思是指定打包后生成的静态资源统一放到`dist/static`目录下，默认是直接放到`dist`目录下。接着重新执行一下`npm run build`打包就可以发现静态资源统一存到`dist/static`目录中。

3.publicPath：指定引用资源的前缀。

默认情况下，Vue CLI 会假设你的应用是被部署在一个域名的根路径上，例如 https://www.my-app.com/。如果应用被部署在一个子路径上，你就需要使用`publicPath`选项指定这个子路径。

例如，如果你的应用被部署在 https://www.my-app.com/my-app/，则需要将`publicPath`设置为`/my-app/`。简单来说，`publicPath`可以为打包后的`index.html`文件中引用的资源路径添加前缀。

我们看一下默认打包时，`dist/index.html` 文件中的资源路径，代码如下所示：

```
<!DOCTYPE html>
<html lang="">
 <head>
  ......
  <link rel="icon" href="/favicon.ico" />
  <title>01_vuecli_demo</title>
  <link href="/css/app.d2422a75.css" rel="preload" as="style" />
  <link href="/js/app.69dc492b.js" rel="preload" as="script" />
  <link href="/js/chunk-vendors.e4a22e78.js" rel="preload" as="script" />
  <link href="/css/app.d2422a75.css" rel="stylesheet" />
 </head>
 <body>
  ......
  <div id="app"></div>
  <script src="/js/chunk-vendors.e4a22e78.js"></script>
  <script src="/js/app.69dc492b.js"></script>
 </body>
</html>
```

接着，在`vue.config.js`文件配置 publicPath属性并赋值 ，代码如下所示：

```
module.exports = {
    // 这里判断是开发环境还是生成环境
    publicPath: process.env.NODE_ENV === 'production'
    ? '/my-app/'
    : '/'
  }
```

上面配置意思是：如果打包的是生产环境，publicPath赋值为`/my-app/`，如果是开发环境运行项目，赋默认值`/`。

接着，重新执行一下`npm run build`打包，然后查看dist目录下的index.html文件，代码如下所示：

```
<!DOCTYPE html>
<html lang="">
 <head>
  ......
  <link rel="icon" href="/my-app/favicon.ico" />
  <title>01_vuecli_demo</title>
  <link href="/my-app/css/app.d2422a75.css" rel="preload" as="style" />
  <link href="/my-app/js/app.095ad9ac.js" rel="preload" as="script" />
  <link href="/my-app/js/chunk-vendors.e4a22e78.js" rel="preload" as="script"
/>
  <link href="/my-app/css/app.d2422a75.css" rel="stylesheet" />
 </head>
 <body>
```

```
 ......
  <div id="app"></div>
  <script src="/my-app/js/chunk-vendors.e4a22e78.js"></script>
  <script src="/my-app/js/app.095ad9ac.js"></script>
 </body>
</html>
```

可以看到，这次的 `index.html` 文件中引用的资源路径都添加了`/my-app`的前缀。

4.alias：配置导包路径的别名。

webpack有一个非常好用的功能是配置别名alias，例如：当项目的目录结构比较深时或一个文件的路径可能需要使用 "../../../" 这种路径片段来导包时，这种导包对于代码的阅读性和维护性都不友好。

这时，我们可以为常见的路径起一个别名，简化路径的同时还提高了代码的阅读性和可维护性。

在Vue.js3项目中，配置 alias 可以在 vue.config.js 文件中的 chainWebpack 属性上进行置。chainWebpack 是一个函数，该函数会接收一个基于webpack-chain的 config 实例，允许对webpack配置进行更细粒度的修改。

因此，我们可以借助 "chainWebpack" 来为路径起别名。

我们来看一下`01_vuecli_demo`项目中src目录下的App.vue文件，代码如下所示：

```
<script>
// 1.使用ES6语法导入了一个 SFC 格式的 HelloWorld 组件
import HelloWorld from './components/HelloWorld.vue'
export default {
  name: 'App',
  components: {
    // 2.注册局部组件
    HelloWorld
  }
}
</script>
```

从上面代码可以看到，从`./components/HelloWorld.vue`路径中导入了`HelloWord`组件。如果我们想要改成从`@/components/HelloWorld.vue`路径中导入该组件，需要配置一下`@`路径别名，其中`@`可以代表一个路径。

接着，在`vue.config.js`文件的`chainWebpack`函数中配置`@`路径别名即可，代码如下所示：

```
const path = require('path')
function resolve(dir) {
  return path.join(__dirname, dir) // __dirname 拿到当前文件所在的路径
}
module.exports = {
    chainWebpack: (config) => {
      // 1.路径起别名
      config.resolve.alias
```

```
        // 2. @别名代表的路径是：指向当前项目的src目录
        .set('@', resolve('src'))
         // 3.components别名代表的路径是：指向当前项目src目录下的components目录
        .set('components', resolve('src/components'))
    }
}
```

可以看到，在`chainWebpack`函数中使用了`config.resolve.alias`来为路径起别名。首先，为项目`src`目录起了一个`@`路径别名。接着，为`src/components`目录起了一个`components`别名。

以后在使用`import`导包时可以直接使用该别名。例如，在`01_vuecli_demo`项目中的`src/App.vue`文件中，可以使用三种方式（选其一即可）来导入`HelloWorld`组件，代码如下所示：

```
<script>
// 1.使用相对路径，导入HelloWorld组件。
import HelloWorld from './components/HelloWorld.vue'
// 2.使用@别名，导入HelloWorld组件。
import HelloWorld from '@/components/HelloWorld.vue'
// 3.使用components别名，导入HelloWorld组件。
import HelloWorld from 'components/HelloWorld.vue'
export default {
  name: 'App',
  components: {
    HelloWorld
  }
}
</script>
```

接着，重新执行`npm run serve`来在浏览器中运行代码。发现`01_vuecli_demo`项目依然可以正常运行。

除此之外，`Vue CLI`脚手架还有许多功能。这次我们先介绍到这里，后面用到其他功能时再详细讲解。如果你想了解更多有关`Vue CLI`的配置，请查看官方网站：https://cli.vuejs.org/config/

接下来，让我们继续介绍一个非常火热的构建工具：`Vite`。

- 它以极速启动、快如闪电的热重载吸引了许多开发者。
- 据`Vite`官网介绍，`Vite`使用`Go`编写的`esbuild`进行预构建依赖，比使用`JavaScript`编写的打包器预构建依赖快10-100倍。
- 那么，它到底是什么呢？下面让我们来一探究竟。

6.4 认识Vite

官方对Vite的定义：**下一代的前端工具链，为开发提供极速响应**，如图6-15所示。

Vite（法语发音 /vit/，是快速的意思）是前端构建工具，专门解决现代前端开发面临的各种复杂问题。Vite拥有以下特点。

- 极速的服务启动：使用原生ESM文件，无需打包。

- 轻量快速的热重载：无论应用程序大小如何，都始终极快的模块热替换（HMR）。
- 丰富的功能：对TypeScript、JSX、CSS等支持开箱即用。
- 优化的构建：可选“多页应用”或“库”模式的预配置Rollup构建。
- 通用的插件：在开发和构建之间共享Rollup-superset插件接口。
- 完全类型化的API：灵活的API和完整的TypeScript类型。

目前，Vite在前端开发领域非常火热，Vue.js 3也对其未来充满期待。它极速启动服务、快如闪电的热重载确实令人惊叹。但是，Vite也存在一定的缺点，例如相比于webpack，Vite整个社区插件还在快速完善。还有Vite更新迭代速度较快，在开发时需注意使用的版本。另外在生产环境中，Vite需要额外依赖Rollup等工具来打包。

图6-15 Vite 下一代的前端工具链

认识了Vite之后，接下来我们来看看如何使用Vite。在Vue.js 3中，通常有两种使用Vite的方式。

1. 使用脚手架，官方推荐使用基于Vite实现的create-vue脚手架。该脚手架已经为我们预设好了各种Vite和Vue.js 3相关的配置，方便快速上手。
2. 从零开始使用Vite搭建Vue.js 3开发环境。这种方式需要一定的Vite基础知识，并需要手动编写各种Vite等配置。

无论是使用webpack还是Vite，对于初学者来说，我们建议使用脚手架来创建项目。下面，我们就使用create-vue脚手架来创建Vue.js 3项目。

6.5 create-vue 脚手架

6.5.1 认识create-vue

`create-vue`类似于`Vue CLI`脚手架，可用来快速创建Vue.js 3项目。`Vue CLI`基于`webpack`，而`create-vue`基于`Vite`。`Vite`也是支持`Vue CLI`中的大多数配置，并且Vite极速启动服务、快如闪电的热重载，提供了更好的开发体验。

`create-vue`脚手架与`Vue CLI`不同的是，它会根据你选择的功能创建一个预配置的项目，然后将其余部分委托给`Vite`。另外需要注意的是，`create-vue`脚手架要求`Node.js`版本大于16。

6.5.2 create-vue新建项目

了解了create-vue 之后，接下来我们将使用create-vue来脚手架来创建Vue.js 3项目。常见有以下两种方式。

1.全局安装create-app脚手架，来新建Vue.js 3项目。

（1）全局安装create-vue脚手架。

在终端中直接输入“npm install -g create-vue”，来全局安装create-vue。

```
# 在终端执行下面命令，来安装create-vue脚手架的最新版本
npm install create-vue@latest -g
```

（2）通过`create-vue`命令来新建Vue.js 3项目。

在 VS Code中打开 `chapter06` 文件夹，接着打开 VS Code终端，然后在终端输入 `create-vue` 命令来新建 ”02-createvite-demo“项目，如图6-16所示。

图6-16 create-vue新建项目

在使用 create-vue 脚手架创建新项目时，可选择以下功能：

- Add TypeScript? ... No / Yes：项目是否集成 TypeScript。
- Add JSX Support? ... No / Yes：项目是否支持 JSX 语法。
- Add Vue Router for Single Page Application development? ... No / Yes：项目是否集成 Router 路由。
- Add Pinia for state management? ... No / Yes：项目是否集成 Pinia 状态管理库。
- Add Vitest for Unit Testing? ... No / Yes：项目是否集成 Vitest 单元测试框架。
- Add an End-to-End Testing Solution? › No：项目是否集成 端到端测试。
- Add ESLint for code quality? ... No / Yes：项目是否集成 ESLint 来检查代码规范。

2.局部安装create-vue脚手架，来创建Vue.js 3项目

在终端中直接输入“npm install -g create-vue”，来全局安装create-vue。

（1）在终端中直接输入"npm init vue@latest"命令来安装create-vue，并创建Vue.js 3项目。

```
npm init vue@latest
```

这里，执行”npm init vue@lates“的意思是：

- 先临时安装 `create-vue@latest` 脚手架（注意：不是全局安装）

- 安装完后立即执行 create-vue 命令来创建项目。

下面我们在 VS Code中打开 `chapter06` 文件夹，接着打开 VS Code 的终端，然后在终端输入 `npm init vue@latest` 命令来创建"02-createapp-demo"项目，如图6-17所示。

```
TERMINAL  DEBUG CONSOLE  PROBLEMS  OUTPUT                                  bash
liujun:chapter06 liujun$ npm init vue@latest   1.第一步
Need to install the following packages:
  create-vue@latest
Ok to proceed? (y) y   2.第二步

Vue.js - The Progressive JavaScript Framework

✔ Project name: … 02-createapp-demo   3.第三步，下面选项暂时都选择了NO
✔ Add TypeScript? … No / Yes
✔ Add JSX Support? … No / Yes
✔ Add Vue Router for Single Page Application development? … No / Yes
✔ Add Pinia for state management? … No / Yes
✔ Add Vitest for Unit Testing? … No / Yes
✔ Add an End-to-End Testing Solution? › No
✔ Add ESLint for code quality? … No / Yes

Scaffolding project in /Users/liujun/Documents/VueCode/chapter06/02-createapp-demo...

Done. Now run:

  cd 02-createapp-demo
  npm install
  npm run dev
```

图6-17 create-vue新建项目

6.5.3 Vue.js 3项目目录结构

用 VSCode 打开`chapter06`目录下已新建好的`02_createvite_demo`项目，如图6-20所示。

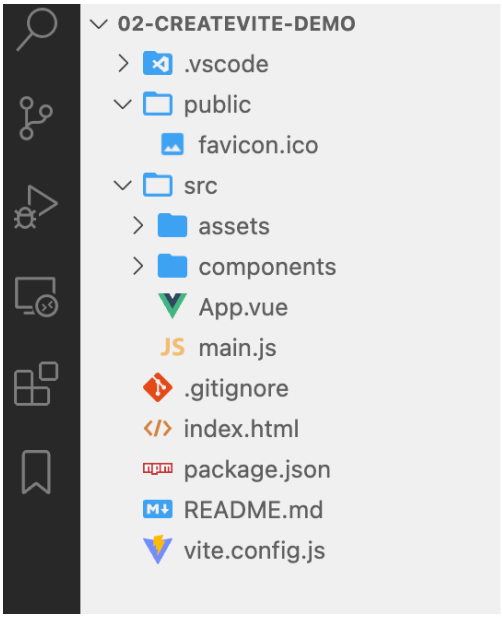

图6-20 Vue.js 3项目目录结构

可以看到，在创建Vue.js 3项目时，无论是使用create-vue还是Vue CLI，项目的目录结构基本相同。接下来，让我们来详细了解一下该项目的目录结构：

```
02_createvite_demo # 项目的名称( 命名规范：建议统一小写，多个单词使用下划线分割 )
├── .vscode # 存放 VS Code 扩展的配置文件
```

```
|   └── extensions.json  # 存储当前工作区或文件夹中所需的扩展列表
├── public  # 项目的一些资源(不会参与webpack打包)
|   └── favicon.ico  # 网站的 icon 图标
├── src  # 项目的所有代码存放的目录
|   ├── assets  # 存放项目的资源
|   |   ├── ..... # 其他资源，例如：图片，字体，全局样式等
|   |   └── logo.svg
|   ├── components # 用于存放Vue.js 3组件的目录
|   |   ├── ......  # 其他组件
|   |   └── HelloWorld.vue # 3.名为 HelloWorld 的局部组件(即SFC组件)
|   ├── App.vue # 2.项目App根组件
|   └── main.js # 1.程序入口文件(也是打包的入口文件)
├── .gitignore # 忽略哪些文件不需要提交到 Git 仓库中
├── index.html # 网站的index.html
├── package.json # 项目的描述文件(包含了项目名称，版本，开发和生成依赖，脚本等等)
├── README.md # 关于项目的说明文档
└── vite.config.js # vite构建工具的配置文件
```

6.5.4 项目的运行和打包

与Vue CLI创建的项目一样，用`create-vite`脚手架创建的项目在`package.json`文件中也提供了对应运行和打包项目的脚本，代码如下所示：

```
{
  "scripts": {
    "dev": "vite",             # 1.开发环境，启动开发服务器。Vite是vite dev或vite serve的别名。
    "build": "vite build",     # 2.生产环境，打包项目的脚本。
    "preview": "vite preview" # 3.本地阅览项目的脚本。
  }
}
```

可以看到，在`scripts`属性中定义了三个脚本。

- `dev`：是启动项目的脚本，当执行`npm run dev`会执行`vite`来启动一个本地服务来运行项目。
- `build`：是打包项目的脚本，当执行`npm run build`会执行`vite build`来打包项目。
- `preview`：是阅览项目的脚本，当执行`npm run preview`会执行`vite preview`来阅览项目。

下面介绍一下这些脚本的使用。

1.运行Vue.js 3项目，可执行npm run dev命令。

在 VS Code 中打开 ”02_createvite_demo“项目，并在 VS Code 终端中执行`npm install`安装项目所需依赖。

安装完成后，执行`npm run dev`，即可在浏览器中输入`localhost:5173`查看项目效果，如图6-21所示。

图6-21 运行Vue.js 3项目

这时，如果在App.vue或HelloWorld.vue等组件的template中新增一些内容。然后按 Ctrl+S 键保存代码后，页面便会自动刷新显示新增的内容。而关闭正在运行项目的方式和前面 Vue CLI 讲的一样。

2.打包Vue.js 3项目，可执行npm run build命令。

打包项目和前面Vue CLI讲的一样，在 VS Code 终端中输入"npm run build" 命令打包，打包成功后会在项目的根目录下生成一个 `dist` 文件夹。

6.5.5 vite.config.js文件解析

与Vue CLI相同，create-vue脚手架也提供了一些常用的配置，方便我们进行扩展和自定义。这些配置包括outDir、assetsDir、alias、base和server等。

需要注意的是，这些配置必须编写在项目根目录的 vite.config.js 文件中，该文件默认配置代码如下所示：

```
import { fileURLToPath, URL } from 'node:url'
import { defineConfig } from 'vite'
import vue from '@vitejs/plugin-vue'
// https://vitejs.dev/config/
export default defineConfig({
  plugins: [vue()], // 提供Vue.js单文件组件支持。
  resolve: {
    alias: {
      '@': fileURLToPath(new URL('./src', import.meta.url)) // 给src路径起个@别名
    }
  }
})
```

接下来将给大家详细介绍这些常用的配置。

1.outDir：指定打包输出的目录名，默认是dist。

`outDir` 选项用于指定打包输出的目录名称，默认为 `dist`。与上述提到的 `vue.config.js` 中的 `outputDir` 选项功能相同，代码如下所示：

```
export default defineConfig({
  ......
  build:{
    outDir: 'build', // 1.指定打包输出的目录名，默认是dist（相对于项目根目录）
  }
})
```

上面代码中，`outDir: 'build'` 指定了打包后生成目录的名称为 **build**，而不是默认的 **dist**。

重新执行 `npm run build` 命令进行打包后，将会在项目的根目录下自动生成一个名为 **build** 的目录。

2.assetsDir：指定静态资源存放目录。

`assetsDir` 用于指定静态资源存放的目录，与上面 `vue.config.js` 中的 `assetsDir` 功能相同，代码如下所示：

```
export default defineConfig({
  ......
  build:{
    outDir: 'build',
    assetsDir: 'static', // 2.指定静态资源存放目录（相对于build.outDir），默认是
assets
  }
})
```

上面代码中，`assetsDir: 'static'` 指定了静态资源存放的目录为 static 。重新执行一下`npm run build`再次打包该项目，打包完成后，静态资源将被存放在`static`目录中。

3.base：指定引用资源的前缀。

`base` 用于指定开发或生产环境服务的公共基础路径，和上面`vue.config.js`中的`publicPath`功能相同，代码如下所示：

```
export default defineConfig({
  base:'/my-app/', // 3.开发或生产环境引入资源的前缀
})
```

上面代码中，`base: '/my-app/'` 指定开发或生产环境服务的公共基础路径为`/my-app/` 。

重新执行一下`npm run build`打包项目，生成的 `index.html` 文件中引用的资源路径都添加了`/my-app`的前缀。

这里的 base 无论是开发环境还是生成环境都赋了`/my-app/`，如果执行`npm run dev`运行开发环境时，本地启动的服务器也会多个`/my-app/`的基础路径。

另外需要注意的是，base属性还支持。

- 绝对URL路径，如 `/my-app/`
- 完整URL，如`http://hy.com/`

- 用于 dev 环境的空字符串或 `./` 。

4.alias：配置导包路径的别名。

`alias`属性用来给导包的路径起个别名，与上面`vue.config.js`中的`alias`功能相同，代码如下所示：

```
export default defineConfig({
  // 4. 给导包的路径起别名
  resolve: {
    alias: {
      "@": fileURLToPath(new URL("./src", import.meta.url)),
      components: fileURLToPath(new URL("./src/components", import.meta.url)),
    },
  },
  ......
})
```

接着，在App.vue根组件中分别使用 @ 和 components 别名来导入`HelloWorld`组件，代码如下所示：

```
<script setup>
// import HelloWorld from './components/HelloWorld.vue' // ok
import HelloWorld from '@/components/HelloWorld.vue' // ok
// import HelloWorld from 'components/HelloWorld.vue' // ok
</script>

<template>
  <HelloWorld msg="Hello Vue 3 + Vite" />
  ......
</template>
```

然后，重新执行`npm run dev`运行”02_createvite_demo“项目，项目依然正常能运行。

其实 create-vue 脚手架还有很多功能，这里先给大家介绍到这里。更多的功能可查阅如下官网文档。

- https://vitejs.dev/config/

6.6 webpack和Vite的区别

Vue CLI 和 create-vue 这两款脚手架都支持开箱即用，都可以快速创建 Vue.js 工程化项目。Vue CLI 是基于 webpack 实现的，而 create-vue 是基于 Vite 实现的。下面我们来总结一下 webpack 和 Vite 的区别。

- webpack 在打包应用程序时，会生成一个依赖关系图，该依赖关系图会包含应用程序所需的所有模块，然后遍历图结构，编译一个个模块，当某个模块有变时，相关依赖模块需全部编译一次。当项目越复杂、模块越多时，打包速度会越来越慢。
- Vite 利用了 ES Module 会自动发起请求的特性，它在打包应用程序时，不需要分析模块的依赖、不需要编译。当浏览器请求某个模块时，再按需对模块内容进行编译，这种按需动态编译的方式，

极大地缩减了编译时间。因此，Vite 的启动速度非常快，比以 JavaScript 编写的打包器预构建依赖快 10-100 倍。当项目越复杂、模块越多时，Vite 打包的优势越明显。

- Vite 天然支持打包 TypeScript、JSX、CSS 等文件；而 webpack 需要安装对应的 Loader 专门处理。
- webpack 支持开发和生产环境打包；Vite 在打包生产环境时需要使用 Rollup 打包，Vite 主要优势在开发阶段。
- webpack 无论是自身优势还是生态都是非常强大，使用者非常多；而 Vite 整个社区生态也正在快速完善。

其实无论是 webpack 还是 Vite，它们本身都是非常优秀的构建工具，非常值得我们去学习。它们也有各自的优点和缺点，在学习阶段使用 Vue CLI 或 create-vue 脚手架都是可以的，但是在企业的生产环境中，建议大家优先选择 Vue CLI 脚手架，因为 webpack 经过这么多年的发展，已经非常稳定，社区生态也非常广。

当然，选择 Vite 我们需要拥抱 Vite 的快速变化，而 webpack 目前表现的更稳定。因此，本书后面章节新建项目将会采用更稳定 Vue CLI 脚手架。

6.7 本章小结

本章内容如下。

- 前端工程化：在一定规模化的前端项目下，利用标准化的工具，比如：webpack、Babel、Less、ESLint、Vue Loader等构建工程化项目，可以提升前端开发效率和质量，降低研发成本，更好地服务于前端研发。
- Vue CLI 脚手架：已经内置了许多webpack及Vue.js相关的配置，它开箱即用，可快速创建Vue.js工程化项目，无需从零开始构建Vue.js工程化项目。
- create-vue 脚手架：是基于Vite，同样也是开箱即用，可快速创建Vue.js工程化项目。Vite也支持Vue CLI项目中的大多数配置，并且Vite极快的启动和模块热替换速度，提供了更好的开发体验。