在线oj项目-vue基础

什么是vue

> 预备知识：
>
> 我们学习和使用vue3之前需要同学们具备html、css、JavaScript相关基础知识。

官网地址：https://cn.vuejs.org/

官方解释：Vue (发音为 /vju:/，类似 view) 是一款用于构建用户界面的 JavaScript 框架。它基于标准 HTML、CSS 和 JavaScript 构建，并提供了一套声明式的、组件化的编程模型，帮助你高效地开发用户界面。无论是简单还是复杂的界面，Vue 都可以胜任。

为什么要学习vue

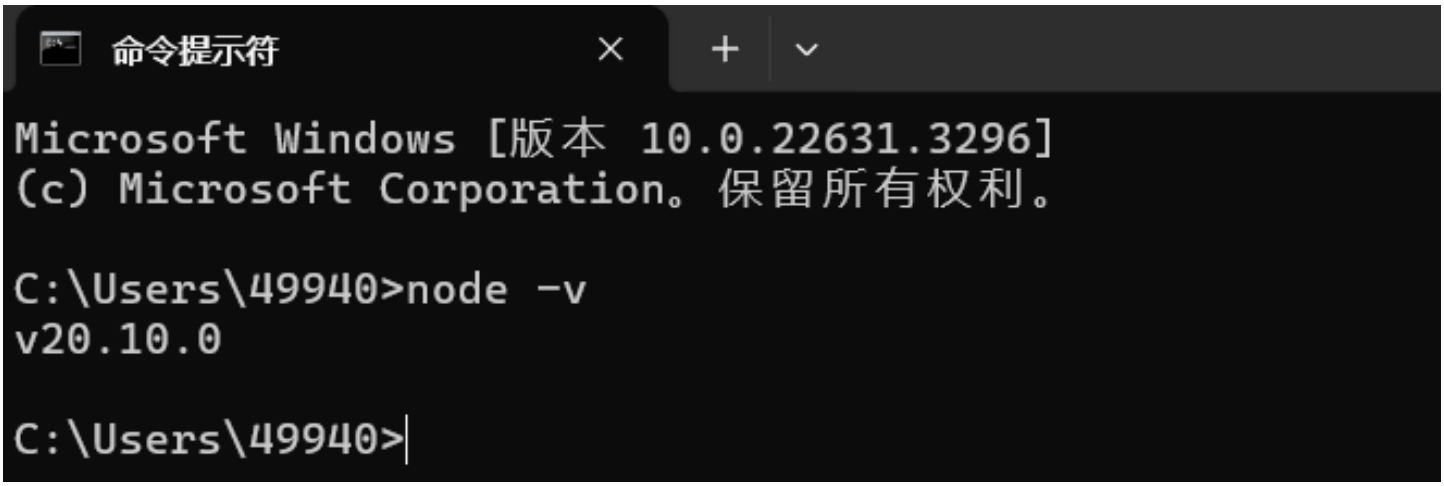
- **选择vue**
  - 易学易用
  - 性能出色
  - 灵活多变
  - 市场主流
- **选择vue3**
  - Vue 2 已于 2023 年 12 月 31 日停止维护
  - 性能更高
  - 体积更小
  - 使用更加灵活
  - 众多企业升级到vue3

创建vue项目

> 前置条件：
>
> - 熟悉命令行
> - 已安装 18.0 或更高版本的 Node.js

- **步骤1：node.js版本检查**

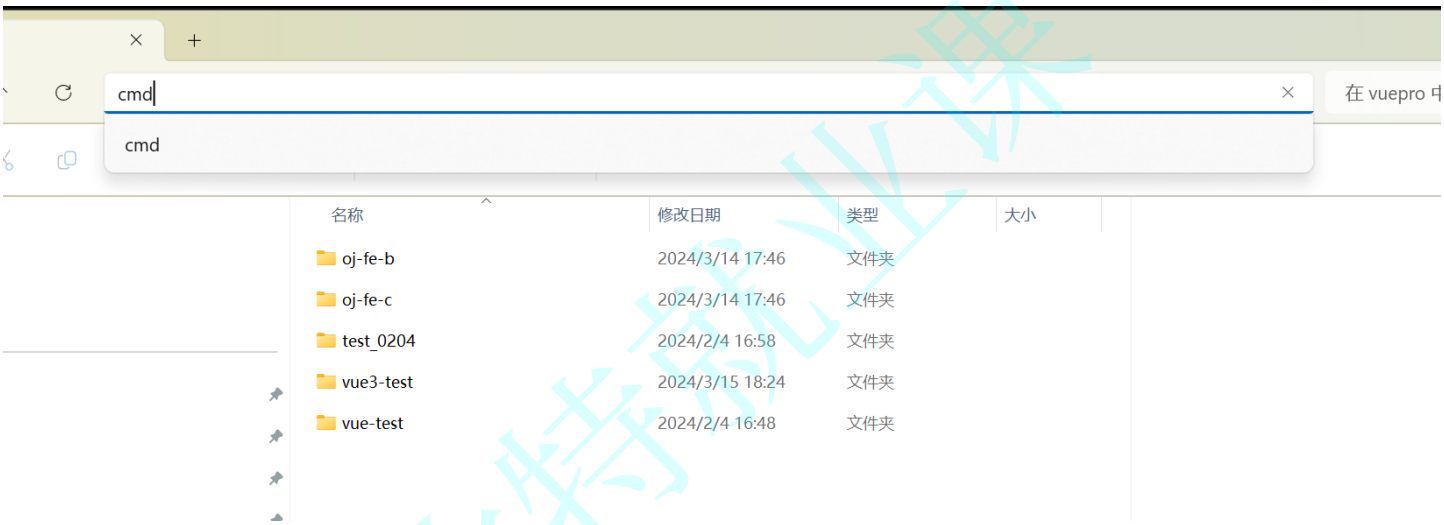
  执行node -v 查看node.js版本



- **步骤2：通过npm命令创建vue3项目**

  代码目录：E:\vuepro，大家可以改为自己想存放的目录。

  从命令行进入：进入到该目录下，在导航栏输入cmd按下回车，进入命令行模式。



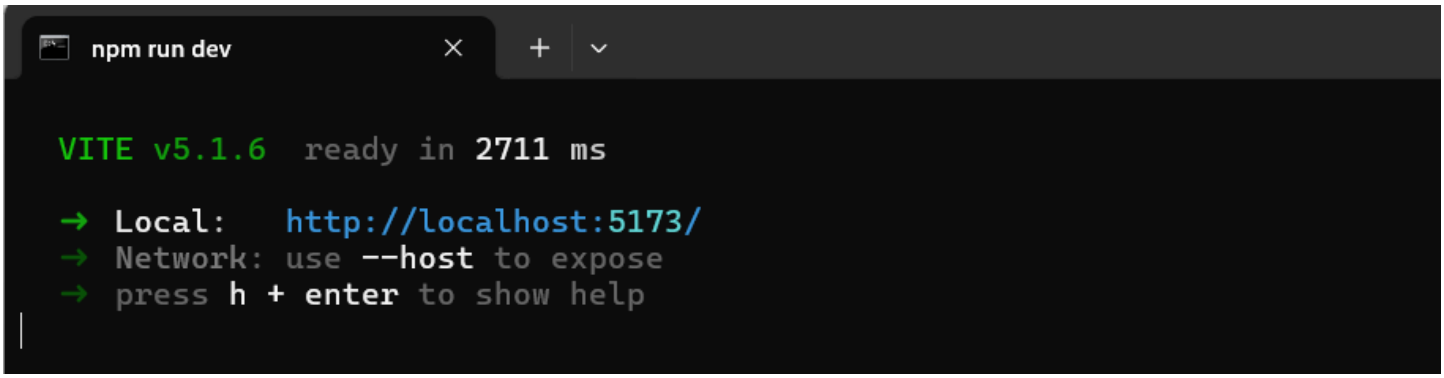
在命令行输入命令：npm create vue@latest

这一指令将会安装并执行 create-vue，它是 Vue 官方的项目脚手架工具。你将会看到一些诸如 TypeScript 和测试支持之类的可选功能提示（我们暂时都选否）：

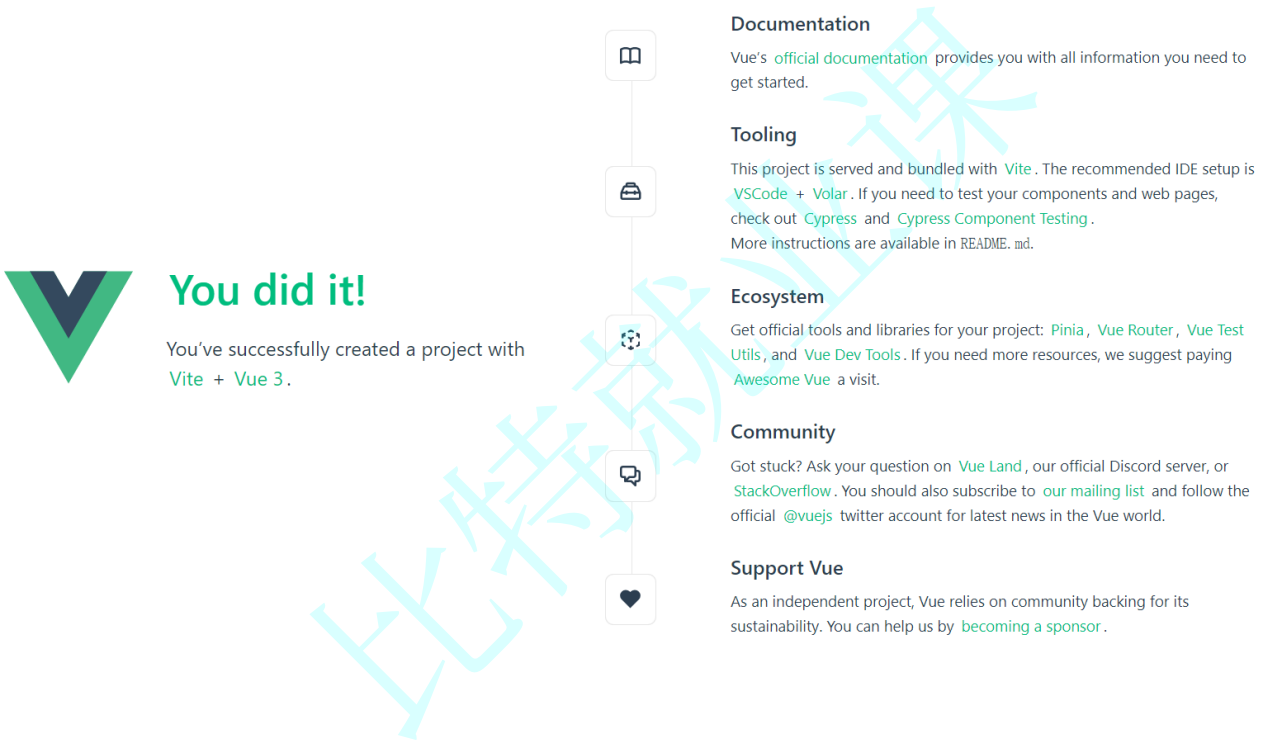
```
E:\vuepro>npm create vue@latest
Need to install the following packages:
create-vue@3.10.1
Ok to proceed? (y) y

Vue.js - The Progressive JavaScript Framework

√ 请输入项目名称：  ... vue-pro
√ 是否使用 TypeScript 语法？  ... 否 / 是
√ 是否启用 JSX 支持？  ... 否 / 是
√ 是否引入 Vue Router 进行单页面应用开发？  ... 否 / 是
√ 是否引入 Pinia 用于状态管理？  ... 否 / 是
√ 是否引入 Vitest 用于单元测试？  ... 否 / 是
√ 是否要引入一款端到端 (End to End) 测试工具？  » 不需要
√ 是否引入 ESLint 用于代码质量检测？  ... 否 / 是
√ Add Vue DevTools extension for debugging? (experimental) ... 否 / 是

正在初始化项目 E:\vuepro\vue-pro...

项目初始化完成，可执行以下命令：

  cd vue-pro
  npm install
  npm run dev


E:\vuepro>
```

安装依赖并启动服务器：创建项目成功后，我们可以看到提示：项目初始化完成，可以执行以下命令。我们则可以根据提示执行这3个命令：

```
E:\vuepro>cd vue-pro

E:\vuepro\vue-pro>cnpm install
√ Linked 27 latest versions fallback to E:\vuepro\vue-pro\node_modules\.store\node_modules
√ Linked 1 public hoist packages to E:\vuepro\vue-pro\node_modules
Recently updated (since 2024-03-09): 3 packages (detail see file E:\vuepro\vue-pro\node_modules\.recently_updates.txt)
√ Run 1 script(s) in 1s.
√ Installed 3 packages on E:\vuepro\vue-pro
√ All packages installed (27 packages installed from npm registry, used 6s(network 5s), speed 2.11MB/s, json 62(2.39MB),
 tarball 9.2MB, manifests cache hit 0, etag hit 0 / miss 1)


E:\vuepro\vue-pro>npm run dev

> vue-pro@0.0.0 dev
> vite
```



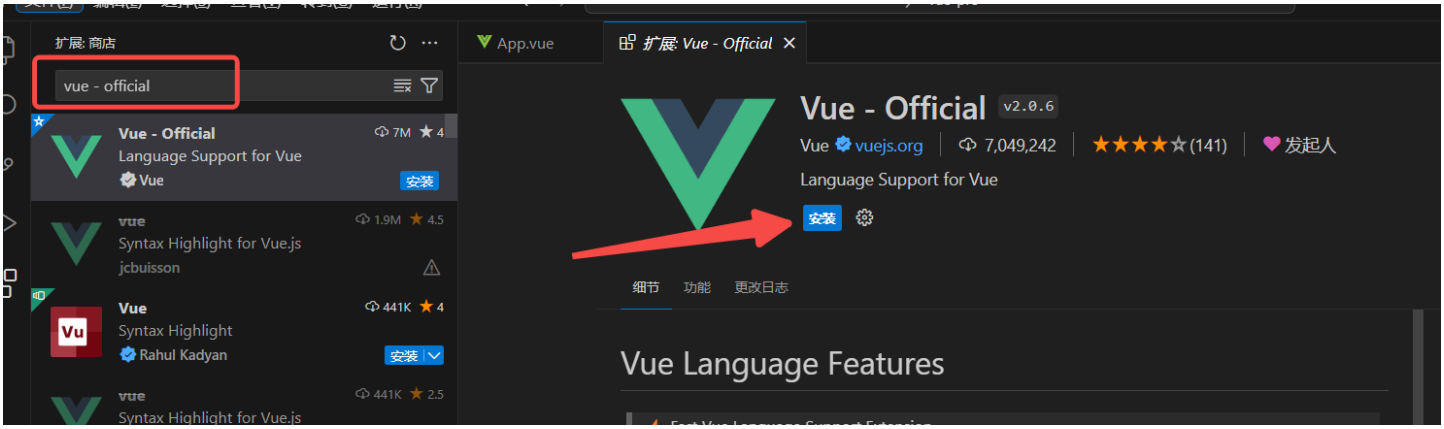
验证成功：第三个命令执行成功后，我们可以看到一个可访问地址：http://localhost:5173/在浏览器访问。看到如下页面则说明前端项目创建并启动成功。



使用编译器

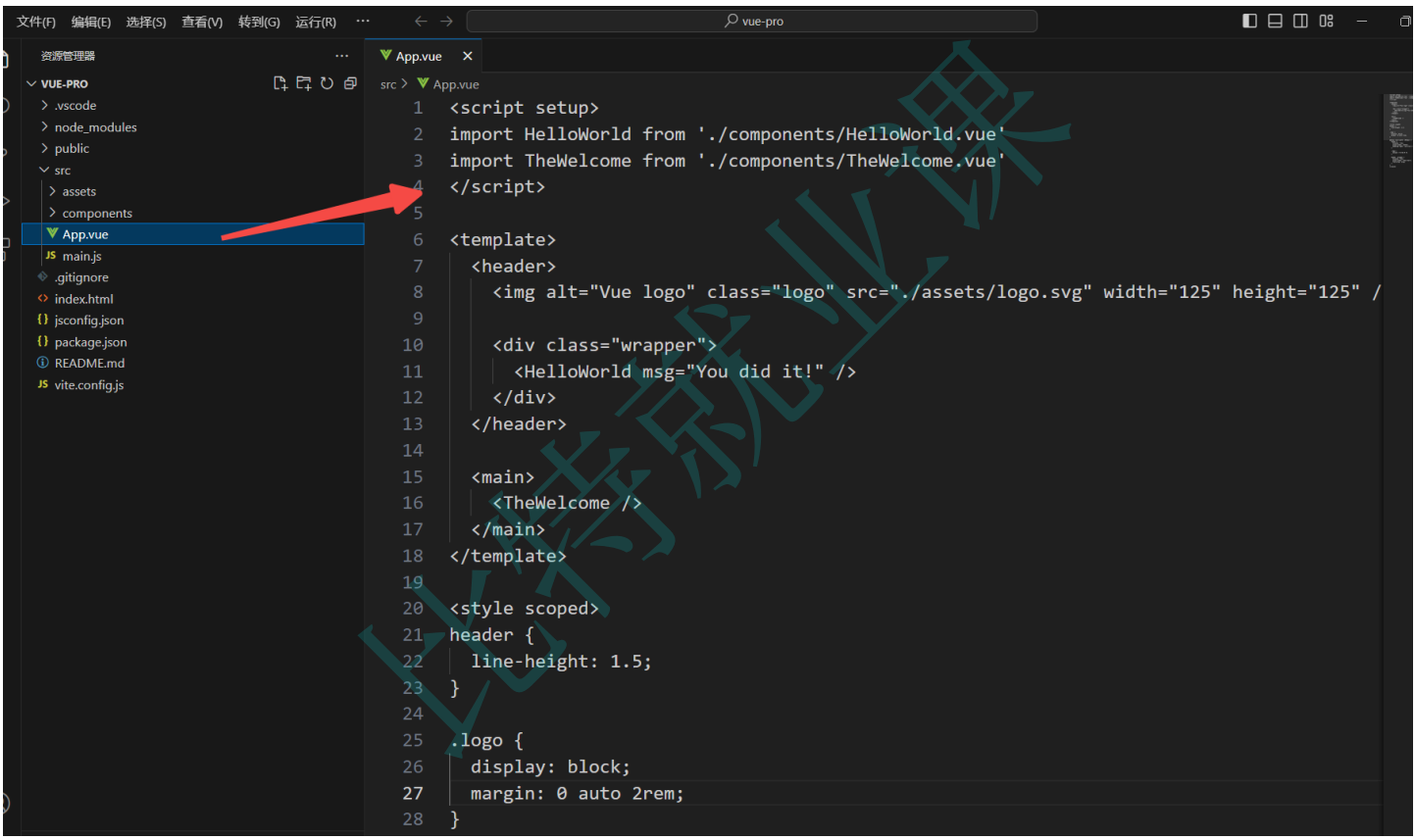
官方推荐IDE配置：vs-code + vue - official 扩展。

vue - official插件安装：

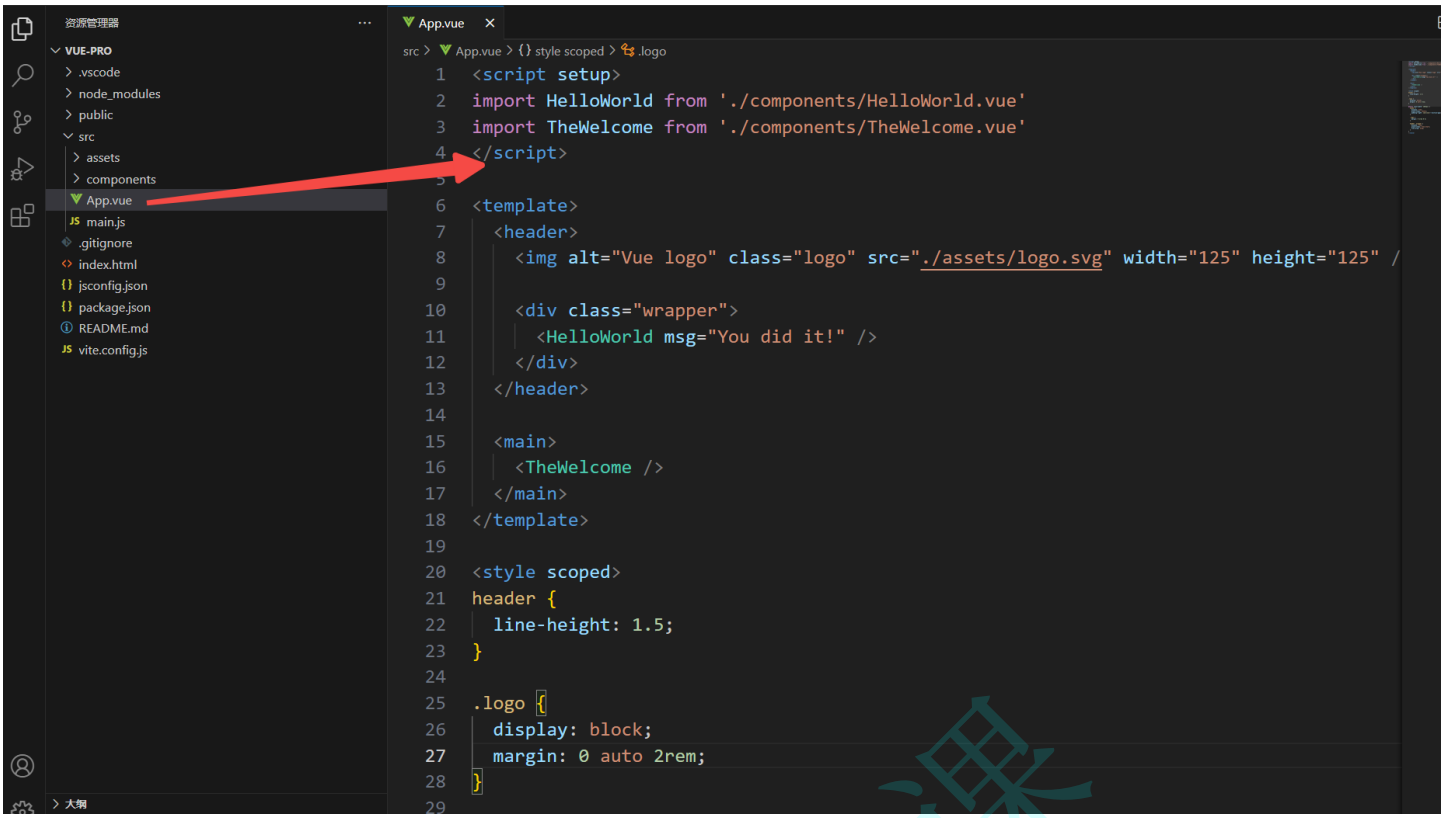


- **安装前：**

  通过vs-code打开刚刚创建好的项目，我们可以看到，文件完全没有高亮，不便于我们进行开发。



- **安装后：**



项目结构

核心目录

- **node_modules**：支持项目运行的依赖文件。（后期开发中我们会引入多个依赖包）
- **public**：存放静态资源和公共资源，如favicon.ico网站图标。
- **src**：项目开发主要文件夹。
- **index.html**：入口的html文件。
- **package.json**：项目的描述文件。
- **vite.config.js**：项目的配置文件。

单文件组件

在大多数启用了构建工具的 Vue 项目中，我们可以使用一种类似 HTML 格式的文件来书写 Vue 组件，它被称为单文件组件 (也被称为 *.vue 文件，英文 Single-File Components，缩写为 SFC)。顾名思义，Vue 的单文件组件会将一个组件的逻辑 (JavaScript)，模板 (HTML) 和样式 (CSS) 封装在同一个文件里。

每一个 *.vue 文件都由三种顶层语言块构成：<template>、<script> 和 <style>，以及一些其他的自定义块。

模板<template>

- 使用html语法，描述整个组件的结构和布局。

- 在模板中可以使用vue的模板语法进行绑定数据、处理事件以及定义组件的DOM结构等。
- 每个 *.vue 文件最多可以包含一个顶层 <template> 块。

脚本 <script>

- 这部分包含组件的JavaScript或TypeScript代码。定义了组件的行为逻辑。
- 每个 *.vue 文件最多可以包含一个 <script> 块。(使用 <script setup> 的情况除外)

<script setup>

- 专为组合式api设计的一种特殊的语法糖，它允许你在一个更简洁、更直观的方式下编写组件逻辑。
- 每个 *.vue 文件最多可以包含一个 <script setup>。(不包括一般的 <script>)

样式 <style>

- 通常使用CSS或CSS预处理器编写样式。定义了组件的样式和布局，用于控制组件的外观和样式。
- 每个 *.vue 文件可以包含多个 <style> 标签。

API风格

Vue 的组件可以按两种不同的风格书写：选项式 API 和组合式 API。

选项式API

选项式 API 是传统的组件开发方式。在这种方式下，组件的选项（如 data、methods、computed、watch 等）被组织在单个对象中，每个选项负责不同的功能。这种方式使得组件的结构清晰，但随着组件的复杂度增加，可能会出现代码难以管理和维护的问题。

- **data:** 定义组件的响应式数据。
- **methods:** 定义组件的方法，它们可以访问组件的data和其他methods。
- **computed:** 用于定义依赖于其他响应式数据的计算值。
- **watch:** 用于观察响应式数据的变化，并执行异步或者代价较大的操作。

.....

官方示例：

```
<script>
export default {
  // data() 返回的属性将会成为响应式的状态
  // 并且暴露在 `this` 上
  data() {
    return {
```

```
      count: 0
    }
  },

  // methods 是一些用来更改状态与触发更新的函数
  // 它们可以在模板中作为事件处理器绑定
  methods: {
    increment() {
      this.count++
    }
  },

  // 生命周期钩子会在组件生命周期的各个不同阶段被调用
  // 例如这个函数就会在组件挂载完成后被调用
  mounted() {
    console.log(`The initial count is ${this.count}.`)
  }
}
</script>

<template>
  <button @click="increment">Count is: {{ count }}</button>
</template>
```

组合式API

通过组合式 API，我们可以使用导入的 API 函数来描述组件逻辑。在单文件组件中，组合式 API 通常会与 <script setup> 搭配使用。这个 setup attribute 是一个标识，告诉 Vue 需要在编译时进行一些处理，让我们可以更简洁地使用组合式 API。

总结：组合式API可以使我们更清晰地组织和编写组件的逻辑，这种风格使得代码更加易于阅读、测试和维护，尤其适用于大型、复杂的Vue应用程序。

官方示例：

```
<script setup>
import { ref, onMounted } from 'vue'

// 响应式状态
const count = ref(0)

// 用来修改状态、触发更新的函数
function increment() {
  count.value++
}

```

```
// 生命周期钩子
onMounted(() => {
  console.log(`The initial count is ${count.value}.`)
})
</script>

<template>
  <button @click="increment">Count is: {{ count }}</button>
</template>
```

<script setup>

<script setup> 是在单文件组件 (SFC) 中使用组合式 API 的编译时语法糖。当同时使用 SFC 与组合式 API 时该语法是默认推荐。相比于普通的 <script> 语法，它具有更多优势：

- 更少的样板内容，更简洁的代码。
- 能够使用纯 TypeScript 声明 props 和自定义事件。
- 更好的运行时性能 (其模板会被编译成同一作用域内的渲染函数，避免了渲染上下文代理对象)。
- 更好的 IDE 类型推导性能 (减少了语言服务器从代码中抽取类型的工作)。

如何选择

两种 API 风格都能够覆盖大部分的应用场景。它们只是同一个底层系统所提供的两套不同的接口。实际上，选项式 API 是在组合式 API 的基础上实现的！关于 Vue 的基础概念和知识在它们之间都是通用的。

选项式api结构清晰，初学者友好。

组合式api更加灵活、自由、高效、更好的复用。

综上：

我们的项目中将选择使用组合式API。

响应式数据

响应式数据是指当数据发生变化时，能够自动更新和通知与之相关的视图或组件，实现视图的实时响应。在Web开发中，响应式数据是实现动态交互和实时更新的关键，它简化了开发过程，提高了用户体验。

在vue3中可以通过ref或是reactive定义响应式数据。

ref

- **接收基本类型或者对象类型的数据传入并返回一个响应式的对象。**

```
//接收对象
<template>
    <button @click="increment">Count is: {{ counter.count }}</button>
</template>

<script setup>
import { ref } from 'vue'

const counter = ref({
  count: 0
})

function increment() {
  counter.value.count++
  console.log('count: ' , counter.value.count)
}
</script>

//接收简单类型

<template>
    <button @click="increment">Count is: {{ counter }}</button>
</template>

<script setup>
import { ref } from 'vue'

const counter = ref(0)
console.log('counter: ' , counter)

function increment() {
  counter.value++
  console.log('count: ' , counter.value)
}
</script>
```

reactive

- **reactive仅支持对象类型**。接受对象类型数据的参数传入并返回一个响应式的对象。
- **示例：**

```
<template>
    <button @click="increment">Count is: {{ counter.count }}</button>
</template>

```

```vue
<script setup>
import { reactive } from 'vue'

const counter = reactive({
  count: 0
})

function increment() {
  counter.count++
  console.log('count: ' , counter.count)
}
</script>
```

Vue Router

官网：https://router.vuejs.org/zh/

什么是路由

路由

路由是一个网络层的概念。路由是指路由器从一个接口上收到数据包，根据数据包的目的地址进行定向并转发到另一个接口的过程。

在Web开发中，路由这个概念也被借鉴，用来描述一个URL到其处理程序的映射关系。简单来说，路由就是根据URL找到对应的处理程序或组件的过程。

服务端路由

服务端路由通常发生在Web服务器层面。当用户请求一个URL时，服务器会根据URL的路径决定执行哪个后端代码并返回生成的HTML页面给客户端。这种路由方式意味着每次用户导航到一个新的页面或刷新页面时，都会从服务器请求一个新的完整的HTML页面。

客户端路由

要理解什么是客户端路由，我们需要先搞清楚什么是单页面应用。

单页面应用（Single Page Application，SPA）

同学们，你们平时上网浏览网页的时候，点击一个链接，浏览器就会加载一个新的页面，这种每次点击链接都加载全新页面的应用，我们称为多页面应用。

在单页面应用中，无论你点击哪个链接或者进行什么操作，浏览器其实只加载了一个页面，也就是我们的主页面。但是，这个主页面里面有很多不同的部分或者组件，当我们点击链接或者进行其他操作时，这些部分或组件会根据我们的需求动态地改变，而不需要重新加载整个页面。这种技术通常通过使用前端框架（如Vue.js）和客户端路由来实现。

什么是客户端路由

主要应用在单页面应用（SPA）中。当用户通过客户端访问不同的路径时，路由的映射函数会利用诸如History API事件这样的浏览器API来管理应用当前应该渲染的视图。这种方式的优点在于，它可以在不重新加载整个页面的情况下，根据用户的请求动态地加载和渲染新的数据或组件，从而带来更为顺滑的用户体验。

Vue Router 是 Vue 官方的客户端路由解决方案。

优点

- **用户体验更加流畅**：SPA的最大特点是页面无需重新加载即可更新视图，这为用户提供了更快速、更流畅的导航体验。用户无需等待整个页面刷新，只需等待相关组件或数据的更新，大大减少了页面跳转时的加载时间。
- **更好的交互性和响应性**：SPA允许前端应用更加精细地控制用户界面的变化。通过动态更新页面的部分内容，SPA可以实现更丰富的交互效果和更快速的响应，从而提升用户的使用体验。
- **更好的前后端分离**：SPA通常与RESTful API等后端技术结合使用，实现前后端的完全分离。这种架构使得前端和后端可以独立开发、测试和部署，提高了开发效率和可维护性。

缺点：

- **初次加载时间长**：SPA通常需要加载较多的JavaScript和CSS资源，导致初次加载页面时可能需要较长时间。这可能会给用户带来不便，尤其是在网络较慢的情况下。
- **SEO挑战**：传统的SPA对搜索引擎来说不够友好，因为搜索引擎爬虫可能无法很好地解析JavaScript动态生成的内容。这可能导致SPA在搜索引擎结果中的排名受到影响。不过，通过服务器端渲染（SSR）或预渲染等技术可以部分解决这个问题。

使用

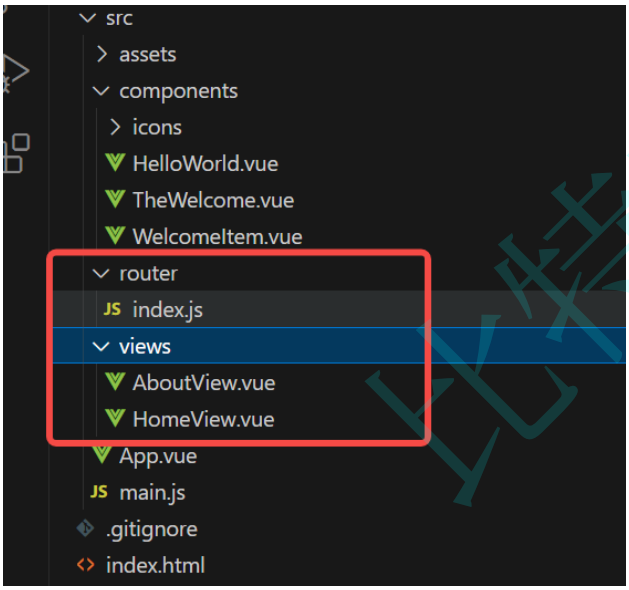
- **创建项目：**

```
C:\Users\49940> npm create vue@latest

Vue.js - The Progressive JavaScript Framework

√ 请输入项目名称：  ... vue-router-test2
√ 是否使用 TypeScript 语法？  ... 否 / 是
√ 是否启用 JSX 支持？  ... 否 / 是
√ 是否引入 Vue Router 进行单页面应用开发？  ... 否 / 是
√ 是否引入 Pinia 用于状态管理？  ... 否 / 是
√ 是否引入 Vitest 用于单元测试？  ... 否 / 是
√ 是否要引入一款端到端（End to End）测试工具？  » 不需要
√ 是否引入 ESLint 用于代码质量检测？  ... 否 / 是
? 是否引入 Vue DevTools 7 扩展用于调试？(试验阶段) » 否 / 是
```

- **package.json**

```json
{
  "name": "vue-router-test",
  "version": "0.0.0",
  "private": true,
  "type": "module",
  ▷ 调试
  "scripts": {
    "dev": "vite",
    "build": "vite build",
    "preview": "vite preview"
  },
  "dependencies": {
    "vue": "^3.4.21",
    "vue-router": "^4.3.0"
  },
  "devDependencies": {
    "@vitejs/plugin-vue": "^5.0.4",
    "vite": "^5.2.8"
  }
}
```

- **目录结构：**



- **main.js：**

```js
import './assets/main.css'

import { createApp } from 'vue'
import App from './App.vue'
import router from './router'

const app = createApp(App)

app.use(router)
```

```
10
11 app.mount('#app')
12
```

- **App.vue**

RouterLink组件：的主要作用是生成可点击的链接。这些链接通常用于导航菜单或页面内的跳转。RouterLink通过其to属性指定链接的目标地址。当用户点击这些链接时，路由会自动切换到对应的页面。

RouterView组件：则用于根据当前路由状态动态渲染匹配的组件。在单页应用中，当URL发生变化时，RouterView会根据当前的路由状态自动渲染对应的组件。这意味着，无论用户导航到哪里，RouterView都会显示与当前路由相匹配的组件内容。

- **router下的index.js**

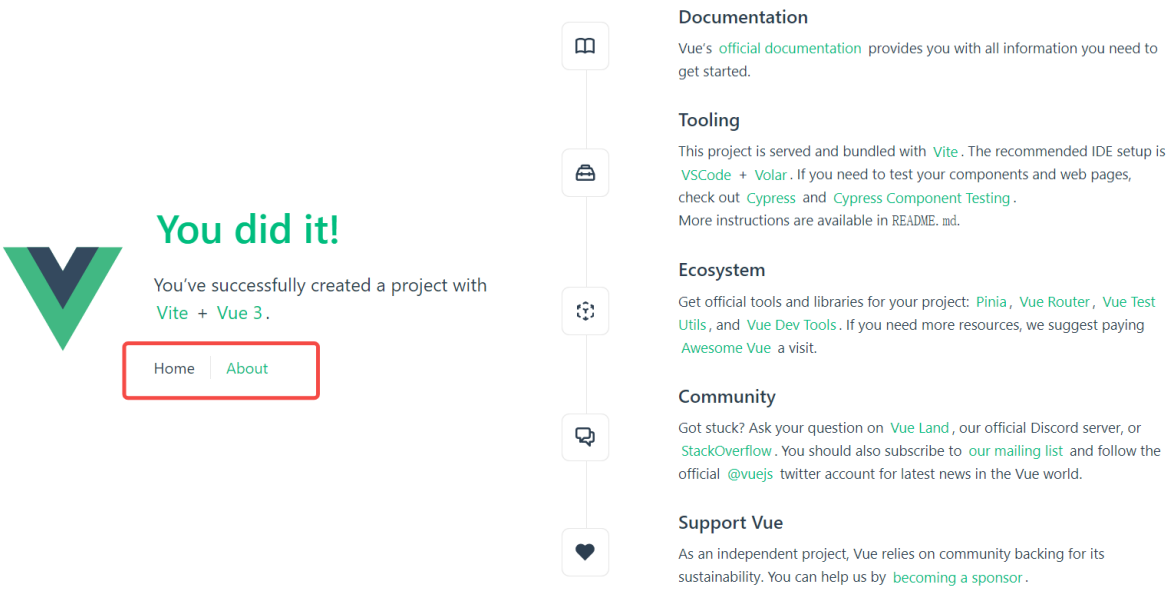
```
1 import { createRouter, createWebHistory } from 'vue-router'
2 import HomeView from '../views/HomeView.vue'
3
4 const router = createRouter({
5   history: createWebHistory(import.meta.env.BASE_URL),
6   routes: [
7     {
```

```
8       path: '/',
9       name: 'home',
10      component: HomeView
11    },
12    {
13      path: '/about',
14      name: 'about',
15      // route level code-splitting
16      // this generates a separate chunk (About.[hash].js) for this route
17      // which is lazy-loaded when the route is visited.
18      component: () => import('../views/AboutView.vue')
19    }
20  ]
21 })
22
23 export default router
```

代码详解：

1. 使用 `createRouter` 创建一个新的 Vue Router 实例。
2. `history` 选项用于定义路由的模式。这里使用了 `createWebHistory`，它基于 HTML5 的 History API。它允许开发者以编程方式操作浏览器的历史记录。
3. `routes` 数组定义了应用中所有的路由。
   a. 每个路由对象都有一个 `path`（路由的路径）、`name`（路由的名称，用于编程式导航）和 `component`（与路径关联的组件）。
   b. 第一个路由对象表示当 URL 为 `/` 时，将渲染 `HomeView` 组件。
   c. 第二个路由对象表示当 URL 为 `/about` 时，将渲染 `AboutView` 组件。这里使用了异步组件的方式，通过函数来动态地导入 `AboutView`。这允许代码分割，使得 `AboutView` 组件的代码只有在用户访问 `/about` 路径时才会被加载，从而提高了应用的初始加载速度。
4. `export default router`：最后，这个 Router 实例被导出，以便在其他地方（比如 Vue 应用的主文件）被引用和使用。

- **页面效果：**



- **切换至About**