创建Vue3项目

* node version > 16.0
* npm create vue@latest
  + npm install
  + npm run dev

Vue3项目结构

* `vite.config.js`：项目的配置文件，基于vite的配置
* `package.json`：项目包文件
* `index.html`：单页入口，提供挂载点
* `main.js`：入口文件，createApp()函数创建应用实例
* `App.vue`：根组件，<script> - <template> - <style>

├── ...

├── src

│   ├── main.js

│   ├── App.vue

│   ├── components

│   └── assets

├── vite.config.js

├── package.json

├── index.html

└── ...

组合式API：setup选项

* beforeCreate钩子之前，自动执行
* this不再指向组件实例，指向undefined

|  |  |
| --- | --- |
| 标准写法 | setup语法糖 |
| <script>  export default {  setup() {  const message = "Hello World"  const logMessage = () => console.log(message)  return {  message, logMessage  }  }  }  </script> | <script setup>  const message = "Hello World"  const logMessage = () => console.log(message)  </script> |

组合式API：reactive()、ref()函数

* reactive()函数
  + 接受对象类型数据的参数传入，并返回一个响应式对象

<script setup>

// 导入reactive

import { reactive } from 'vue'

const state = reactive({

count: 0

})

const increaseOne = () => state.count++

</script>

<template>

<button @click="increaseOne">{{ state.count }}</button>

</template>

* ref()函数
  + 接受简单类型或对象类型的参数传入，并返回一个响应式对象

<script setup>

// 导入ref

import { ref } from 'vue';

const count = ref(0)

const increaseOne = () => count.value++

</script>

<template>

<button @click="increaseOne">{{ count }}</button>

</template>

组合式API：computed()计算属性函数

* 计算属性不应该有“副作用”，比如异步请求/修改DOM
* 避免直接修改计算属性的值，计算属性应该是只读的

<script setup>

import { ref } from 'vue'

// 导入computed

import { computed } from 'vue'

const list = ref([1, 2, 3, 4, 5, 6, 7, 8])

// 计算属性

const computedList = computed(() => list.value.filter(i => i > 2))

</script>

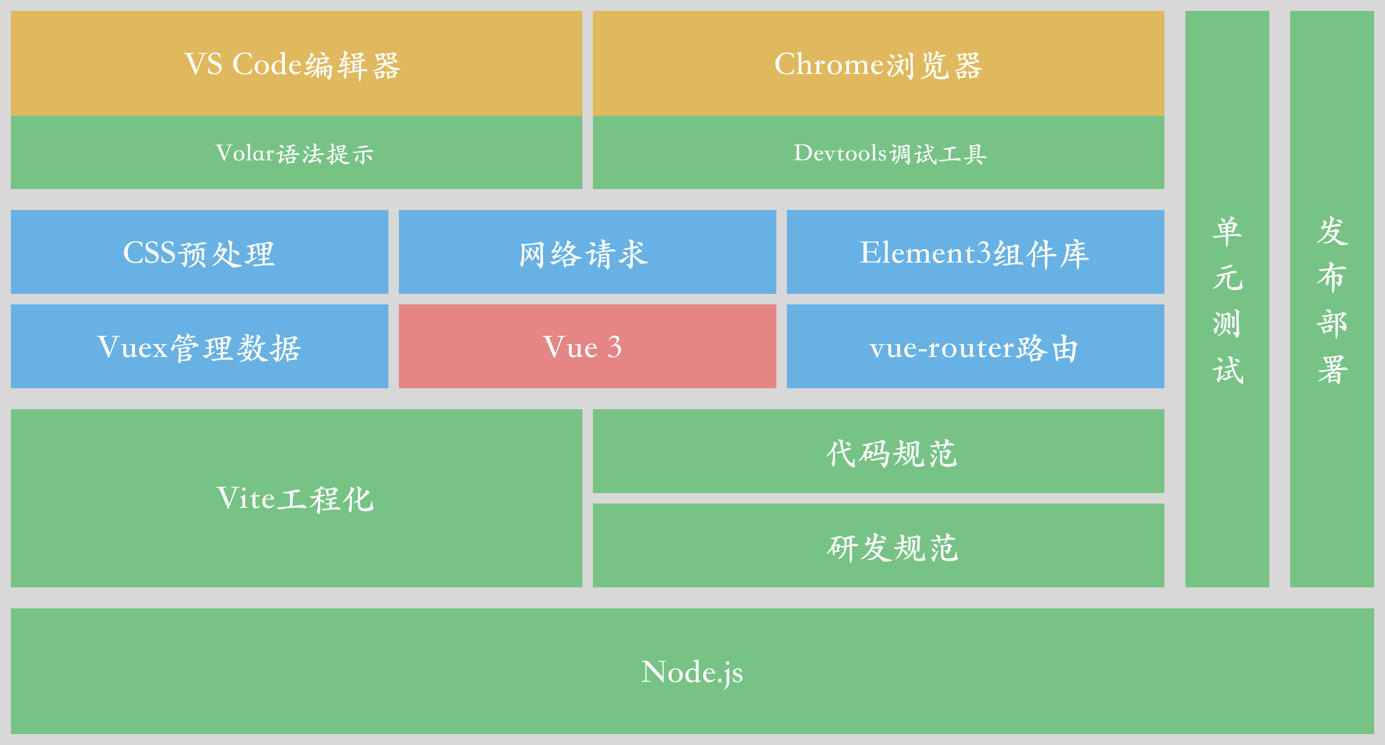
<template>

<div>原始数组: {{ list }}</div>

<div>计算后数组: {{ computedList }}</div>

</template>

组合式API：watch()监听对象



node.js

vs code + Volar

Chrome + Vue3 Devtools

npm init vue@latest

npm install

npm run dev

├── index.html // 单页入口

├── package-lock.json

├── package.json // 包配置文件

├── vite.config.json // vite工程化配置文件

├── public // 资源文件

│   └── favicon.ico

└── src // 源码

├── App.vue // 单文件组件

├── main.js // 入口文件

├── api // 数据请求

├── assets // 静态资源

├── components // 组件

├── pages // 页面

├── router // 路由配置

├── store // vuex数据

└── utils // 工具函数

npm install vue-router@next vuex@next

// src/router/index.js

import {createRouter, createWebHistory} from 'vue-router'

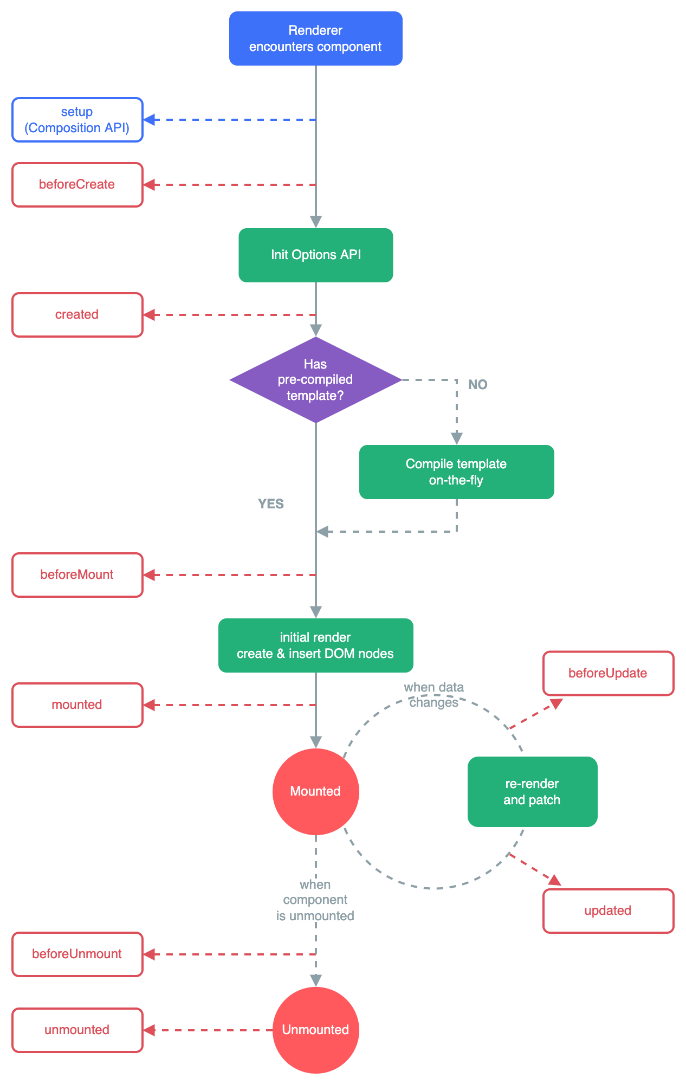
// src/main.js

createApp(App).use(router).mount('#app')

<router-link to=””></router-link>

<router-view></router-view>

生命周期

mounted，updated，unmounted

所有生命周期钩子函数的 this 上下文都会自动指向当前调用它的组件实例。注意：避免用箭头函数来定义生命周期钩子，因为如果这样的话你将无法在函数中通过 this 获取组件实例。

组合式API

setup()

<script setup>

</script>

声明响应式状态

ref(), reactive()函数：创建一个响应式对象或数组

响应式对象其实是 JavaScript Proxy

import {ref, reactive} from ‘vue’

要在组件模板中使用响应式状态，需要在 setup() 函数中定义并返回。

单文件组件（SFC）时，我们可以使用 <script setup> 来大幅度地简化代码。