建置工具Vite、環境安裝、屬性綁定、響應式、計算屬性

Vue3

2022

@Eason

目錄

01

建置工具 Vite

Vite 是一種新型前端構建工具，能夠顯著提升前端開發體驗。

- Module 發展史
- 開發環境 vs 生產環境

01. 建置工具 Vite

- Vite 提供開發者一個快速建置 Vue.js 架構，包含相關功能、套件安裝。
  只要簡單勾選欲安裝的選項，建置工具就會產生Vue完整架構及完成相關設定。

- Vite 在本地端開發階段時不做打包，而是使用原生瀏覽器支援的 es6 module 來載入相依性 (等到瀏覽器遇到 import 語句時才發出模組的請求)，使前端啟動開發伺服器時，能快速啟動。

Module 發展史

- CommonJS
- ESM (ES6 Modules)

CommonJS

- exporting

```
// doSomething.js
module.exports = function doSomething(n) {
  ...
}
```

- importing

```
// main.js
const doSomething = require('./doSomething.js');
```

CommonJS 經常在 node開發 中出現。

- CommonJS 使用同步方式引入模塊。
- 可以從node_modules或者本地目錄引入模塊。
  如：const someModule = require('./some/local/file');。
- CommonJS 引入模塊的一個複製文件。
- CommonJS 不能在瀏覽器裡工作。要在瀏覽器裡使用，則需要轉碼和打包 (如：webpack)。

ESM (ES6 Modules / JavaScript Module)

早期在瀏覽器並沒有原生的 module 機制，所以才會產生出各個標準，

但是這點在 ES6 的時候有了改變，因為 ES6 的規範裡終於有 module 了！

我們稱這個做 ES6 Modules，簡稱 ESM。

- export 語法

```
// profile.js
var firstName = 'Michael' ;
var lastName = 'Jackson' ;
var year = 1958 ;
export { firstName, lastName, year };
```

- import 語法

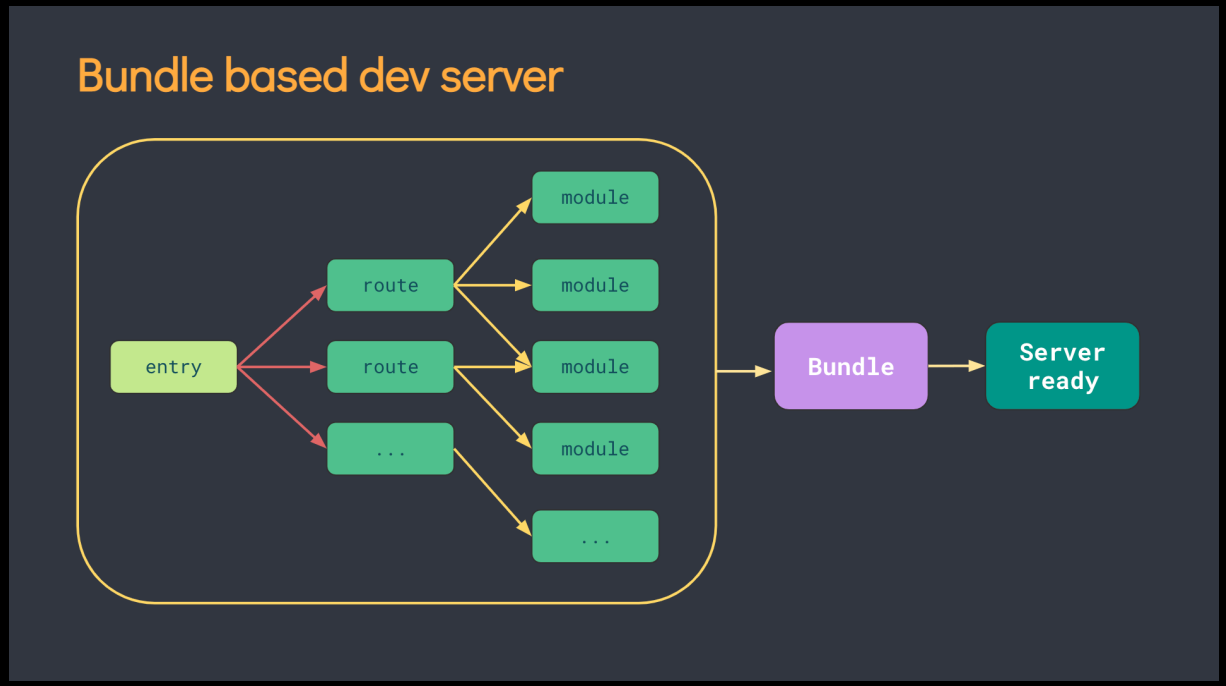
```
// main.js
import { firstName, lastName, year } from './profile.js' ;
function setName ( element ) {
   element. textContent = firstName + ' ' + lastName;
}
```

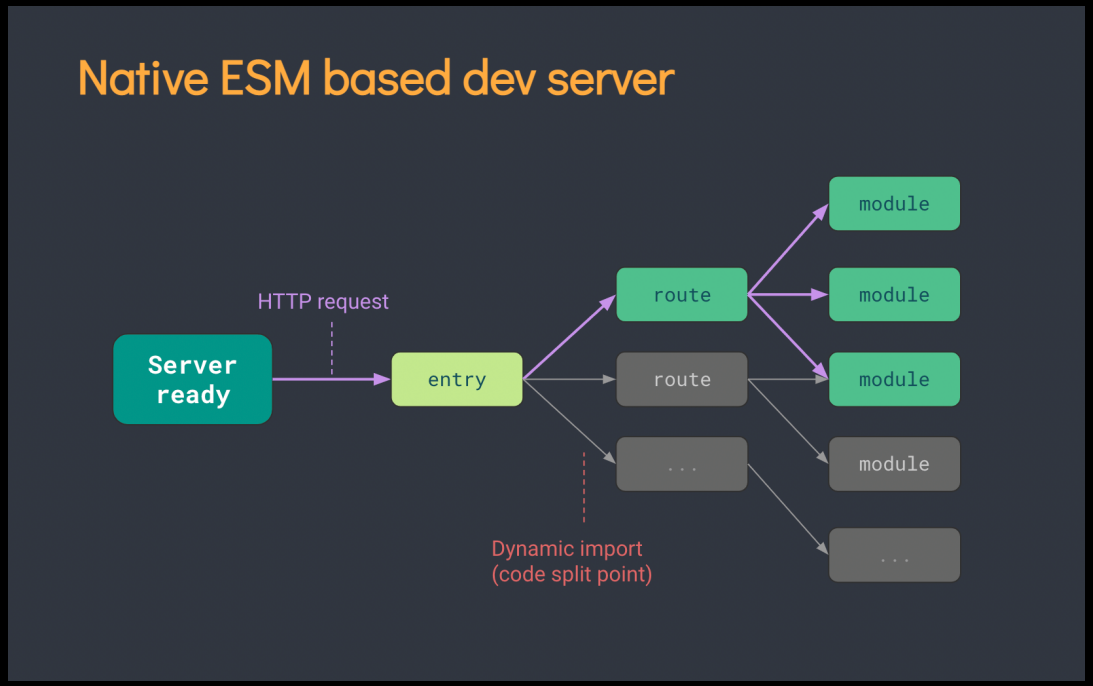
現在幾乎所有的主流瀏覽器都已經原生支援 ESM ！
以往在瀏覽器不支援的情況下，我們需要經過 webpack 或其他軟體打包檔案，
import 跟 export 在輸出時，可能已經被 babel 或 webpack 轉成 CommonJS 或其他形式。

webpack 這些打包工作之所以慢的原因，在於打包及編譯的過程，
需要分析過所有檔案以及套件的相依性，再根據這些資訊把東西包在一起。

而 Vite 就是避開了 bundling，採用 Native ESM，利用瀏覽器處理複雜的相依性。

Bundle vs Native ESM





開發環境 vs 生產環境

vite 在 生產環境 產生 build 時 採用與webpack一樣的方式，
將所有套件以及依賴都打包進去。

而 vite 使用的是 rollup 工具，走傳統的打包策略，跟 webpack 沒兩樣。

開發環境 與 生產環境 使用不同編譯方式：

- 開發環境：使用 Native ESM 是因為所有套件依賴其實都在本機產生的 Local server 中，
  瀏覽器可以很快速的找到依賴。

- 生產環境：瀏覽器就要等到這些套件全部都下載完成以後才能開始執行JavaScript，
  而且瀏覽器會有同時下載量的限制，執行速度反而會更慢。

02 建置Vue應用

- 應用實例
- 根組件
- 掛載應用
- 多個應用實例

創建Vue應用

```
npm init vue@3.4.0
```

這一指令將會安裝並執行 create-vue，它是 Vue 官方的項目腳手架工具。

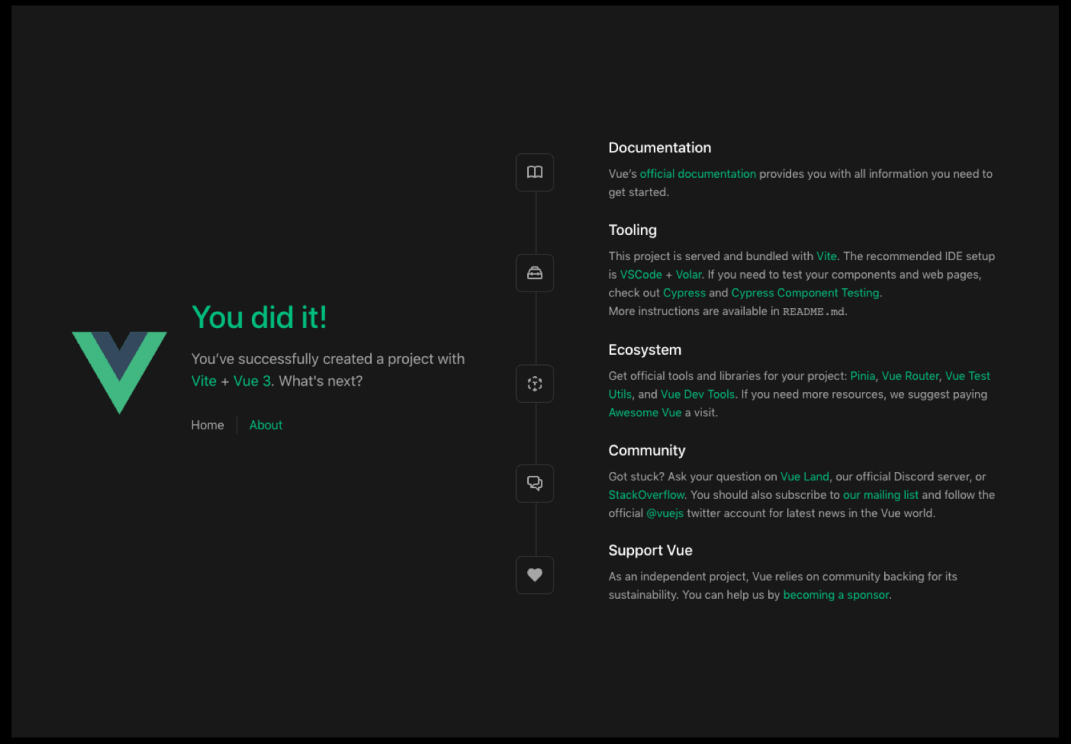
將會看到一些諸如 TypeScript 和測試支持之類的可選功能提示：

```
   Project name: … <your-project-name>
   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? › - Use arrow-keys. Return to submit.
❯   No
    Cypress
    Playwright
   Add ESLint for code quality? … No / Yes
Scaffolding project in ./<your-project-name>...
Done.
```

運行Dev

- 切換至 專案資料夾
- 安裝相關套件
- 運行

```
cd <your-project-name>
npm install
npm run dev
```



應用實例

每個Vue應用都是通過 createApp 函數創建一個新的 **應用實例**：

```
import { createApp } from 'vue';

const app = createApp({
  /* 根組件選項 */
})
```

根組件

每個應用都需要一個 根組件，其他組件將作為其子組件。

```
import { createApp } from 'vue';

// 從一個單文件組建中導入根組件
import App from './App.vue'

const app = createApp(App)
```

掛載應用

應用實例必須在調用了 .mount() 方法後才會渲染出來。

該方法接收一個“**容器**”參數，可以是一個實際的DOM 元素或是一個CSS 選擇器字符串：

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

```
app.mount('#app');
```

多個應用實例

應用實例並不只限於一個。

createApp API 可以在同一個頁面中創建多個共存的Vue 應用，

而且每個應用都擁有自己的用於配置和全局資源的作用域。

```
const app1 = createApp({
  /* ... */
})
app1.mount('#container-1')
```

```
const app2 = createApp({
  /* ... */
})
app2.mount('#container-2')
```

03 SFC (單文件組件)

相應語言塊

SFC (單文件組件)

Vue 的單文件組件(*.vue 文件，英文 Single-File Component，簡稱 **SFC**)，是一種類似HTML語法的自定義文件格式。

每一個 *.vue 文件都由三種頂層語言塊構成：

模板<template>、

邏輯<script> 和

樣式<style>，以及其他自定義塊。

SFC (單文件組件)

邏輯
<script>

```
<script setup>
 ...
</script>
```

模板
<template>

```
<template>
 <p class="greeting">{{ greeting }}</p>
 <custom>
  This could be e.g. documentation for the component.
 </custom>
</template>
```

樣式
<style>

```
<style>
 .greeting {
  ...
 }
</style>
```

相應語言塊

<template>

- 每個 *.vue 文件最多可以包含一個頂層 <template> 塊。
- 語塊包裹的內容將會被提取、傳遞給 @vue/compiler-dom，預編譯為JavaScript 渲染函數，並附在導出的組件上作為其 render 選項。

<script>

- 每個 *.vue 文件最多可以包含一個 <script> 塊 (使用 <script setup> 的情況除外)。
- 這個腳本代碼塊將作為 ES 模塊執行。
- 默認導出應該是Vue 的組件選項物件，可以是一個物件字面量或是defineComponent函數的返回值。

<script setup>

- 每個 *.vue 文件最多可以包含一個 <script setup>。(不包括一般的<script>)
- 這個腳本塊將被預處理為組件的 setup() 函數，這意味著它將為每一個組件實例都執行。<script setup> 中的頂層綁定都將自動暴露給模板。

<style>

- 每個 *.vue 文件可以包含多個 <style> 標籤。
- 一個 <style> 標籤可以使用 scoped 或 module 屬性，來幫助封裝當前組件的樣式。使用了不同封裝模式的多個 <style> 標籤可以被混合入同一個組件。

自定義塊

- 在一個 *.vue 文件中可以為任何項目特定需求使用額外的自定義塊。
- 舉例：<custom />。

04 文本插值

最基本的數據綁定形式是文本插值，它使用的是 Mustache 語法(雙大括號)：

```
let msg = '最新公告';
```

```
<span>Message: {{ msg }}</span>
```

雙大括號標籤會被替換為相應組件實例中 msg 屬性的值。

同時每次 msg 屬性更改時它也會同步更新。

使用 JavaScript表達式

這些表達式都會以組件為作用域解析執行。

```
{{ number + 1 }}

{{ ok ? 'YES' : 'NO' }}

{{ message.split('').reverse().join('') }}
```

```
<div :id="`list-${id}`"></div>
```

僅支援表達式，**不合法** 範例如下：

範例	說明
`{{ var a = 1 }}`	不支援，因為 這是一個語句
`{{ if (ok) { return message } }}`	條件控制不支援，請使用三元運算

05 v-bind (屬性綁定)

- 布林值Attribute
- 動態綁定多個值

v-bind (屬性綁定)

雙大括號不能在 HTML attributes 中使用。
想要響應式地綁定一個attribute，應該使用 v-bind 指令：

```
<div v-bind:id="dynamicId"></div>
```

v-bind 指令，
指示 Vue 將元素的 id attribute 與組件的 dynamicId 屬性保持一致。

如果綁定的值是 null 或 undefined，則 attribute，會從渲染的元素上移除。

可簡寫『：』

```
<div :id="dynamicId"></div>
```

布林值Attribute

布林值attribute 依據 true / false 值來決定 attribute 是否應該存在於該元素上。
例：disabled 就是最常見的例子之一。

```
<button :disabled="isButtonDisabled">Button</button>
```

當 isButtonDisabled 為 真值(Truthy)或一個空字符串(即 <button disabled= “” >) 時，
元素會包含這個 disabled attribute。

而當其為其他 假值(Falsy) 時，attribute 將被忽略。

動態綁定多個值

一個包含多個 attribute 的JavaScript 物件：

```
const objectOfAttrs = {
  id: 'container’  ,
  class: 'wrapper'
}
```

通過不帶參數的 v-bind (不能簡寫)，可以將它們綁定到單個元素上：

```
<div v-bind="objectOfAttrs"></div>
```

06 v-html (原始HTML)

v-html (原始HTML)

v-html (原始HTML)

雙大括號會將數據解譯為純文字，而不是HTML。

若想插入HTML，需要使用 v-html 指令：

```
let rawHtml = '<span style="color: red">這應該是紅色的。</span>'
```

```
<p>使用文本插值：{{ rawHtml }}</p>
<p>使用 v-html 指令：<span v-html="rawHtml"></span></p>
```

使用文本插值：這應該是紅色的。

使用 v-html 指令：這應該是紅色的。。

07 條件渲染

- v-if、v-else-if、v-else
- v-show
- v-if vs v-show

v-if、v-else-if、v-else

只會在指令的表達式返回 真值(Truthy) 時才被渲染。

可使用於 <template> 元素上。

```
<h1 v-if="awesome">Vue is awesome!</h1>
```

```
<div v-if="type === 'A' " >
 A
</div>
<div v-else-if="type === 'B' " >
 B
</div>
<div v-else>
 Not A/B
</div>
```

v-show

按條件顯示元素。

```
<h1 v-show="ok">Hello!</h1>
```

- 不同之處在於 v-show 會在DOM 渲染中保留該元素；
- v-show 僅切換了該元素上 display 的CSS 屬性。
- v-show 不支持在 <template> 元素上使用，也不能和 v-else 搭配使用。

v-if vs v-show

按條件顯示元素。

```
<h1 v-show="ok">Hello!</h1>
```

- v-if 是惰性的，如果在初次渲染時條件值為 假值(Falsy) ，則不會做任何事。
  條件區塊只有當條件首次變為 真值(Truthy) 時才被渲染。
- v-show 簡單許多，元素無論初始條件如何，始終會被渲染，
  只有CSS display 屬性會被切換。

- v-if 有更高的切換開銷，而 v-show 有更高的初始渲染開銷。
  因此，如果需要頻繁切換，則使用 v-show 較好；
  如果在運行時綁定條件很少改變，則 v-if 會更合適。

08 v-for

v-for

v-for 指令的值需要使用 item in items 形式的特殊語法。

- item：迭代項的別名
- items：源數據的陣列

```
const items = [
  { message: 'Foo' },
  { message: 'Bar' }
]
```

```
<li v-for="item in items">
  {{ item.message }}
</li>
```

- Foo
- Bar

09 v-on

v-on

用於監聽DOM事件，並於事件觸發時執行對應的JavaScript。

```
function say(message) {
  alert(message);
}
```

```
<button v-on:click="say('hello')">Say hello</button>
<button v-on:click="say('bye')">Say bye</button>
```

可簡寫『@』

```
<button @click="say('hello')">Say hello</button>
```

10 事件修飾符

事件修飾符

Vue 為 v-on 提供了 **事件修飾符**。
修飾符是用『.』表示的指令後綴，包含以下這些：

修飾符	說明	範例
.stop	單擊事件將停止傳遞	`<a @click.stop="doThis"></a>`
.prevent	停止html預設行為	`<form @submit.prevent="onSubmit"></form>`
.self	僅當 event.target是元素本身，才觸發	`<div @click.self="doThat">...</div>`
.capture	事件使用捕獲模式	`<div @click.capture="doThis">...</div>`
.once	點擊事件最多只會觸發一次	`<a @click.once="doThis"></a>`
.passive	完全忽略 event.preventDefault()， 意指 .prevent 會被忽略，不建議一起使用	`<div @scroll.passive="onScroll">...</div>`

11 按鍵修飾符

- 按鍵別名
- 系統按鍵修飾符
- 滑鼠按鍵修飾符

按鍵別名

Vue 為一些常用的按鍵提供了別名：

- .enter
- .tab
- .delete (將捕獲 "Delete" 和 "Backspace" 按鍵)
- .esc
- .space
- .up
- .down
- .left
- .right

```
<!-- 僅在 `key` 為 `Enter` 時調用 `submit` -->
<input @keyup.enter="submit" />
```

系統按鍵修飾符

- .ctrl
- .alt
- .shift
- .meta：在Mac鍵盤上，是Command鍵(⌘)，在Windows鍵盤上，是Windows鍵(田)

```
<!-- Alt + Enter -->
<input @keyup.alt.enter="clear" />

<!-- Ctrl + 點擊 -->
<div @click.ctrl="doSomething">Do something</div>
```

.exact 修飾符

僅當按下明確按鈕，且未按任何其他鍵時才會觸發

```
<!-- 當按下 Ctrl 時，即使同時按下 Alt 或 Shift 也會觸發 -->
<button @click.ctrl="onClick">A</button>

<!-- 僅當按下 Ctrl 且未按任何其他鍵時才會觸發 -->
<button @click.ctrl.exact="onCtrlClick">A</button>

<!-- 僅當沒有按下任何系統按鍵時觸發 -->
<button @click.exact="onClick">A</button>
```

滑鼠按鍵修飾符

- .left：左鍵
- .right：右鍵
- .middle：中鍵

```
<!-- 當按下 左鍵 時 觸發 -->
<button @click.left="onClickLeft">Left</button>

<!-- 當按下 右鍵 時 觸發 -->
<button @click.right="onClickRight">Right</button>

<!-- 當按下 中鍵 時 觸發 -->
<button @click.middle="onClickMiddle">Middle</button>
```

12 表單輸入綁定 (v-model)

- 修飾符
- 組件上的 v-model

12.表單輸入綁定 (v-model)

當需要將表單輸入的內容同步綁定時，需處理手動連接值和事件監聽綁定：

```
<input
  :value="text"
  @input="event => text = event.target.value">
```

v-model 指令可簡化這一步驟：

```
<input v-model="text">
```

v-model 可用於各種不同類型的輸入，<input>、<select>等元素。

它會根據使用的元素自動使用對應的DOM屬性和事件組合：

元素	對應機制
<input>、<textarea>	會綁定 value屬性，並偵聽 input事件。
<input type="checkbox">、 <input type="radio">	會綁定 checked屬性，並偵聽 change事件。
<select>	會綁定 value屬性，並偵聽 change事件。

修飾符

- .lazy

  默認情況下，v-model會在每次input事件後更新數據。

  若想改為在每次 change 事件觸發時才更新數據(unFocus)，

  可加上 .lazy 修飾符：

```
<input v-model.lazy="msg" />
```

- .number

  如果想讓輸入自動轉換為數字，

  可以在 v-model 後添加 .number 修飾符來管理輸入：

```
<input v-model.number="age" />
```

  - 如果該值無法被 parseFloat() 處理，那麼將返回原始值。
  - 當輸入框有 type=“number” 時，number 修飾符會自動啟用。

- .trim

  默認自動去除輸入內容中兩端的空格，

  可以在 `v-model` 後添加 `.trim` 修飾符：

```
<input v-model.trim="msg" />
```

組件上的 v-model

默認情況下，v-model在組件上都是使用 modelValue 作為 prop，

並以 update:modelValue 作為對應的事件。

```
<CustomInput v-model="searchText" />
```

```
<!-- CustomInput.vue -->
<script setup>
 defineProps(['modelValue   '])
 defineEmits(['update:modelValue'])
</script>

<template>
<input
 :value="modelValue"
 @input="$emit('update:modelValue', $event.target.value)"
/>
</template>
```

也可以自定義參數名：

```
<CustomInput v-model:title="searchText" />
```

```
<!-- CustomInput.vue -->
<script setup>
 defineProps([ ‘title'])
 defineEmits(['update:title'])
</script>

<template>
<input
 :value=” title"
 @input="$emit('update:title', $event.target.value)"
/>
</template>
```

13

響應式 (reactive、 ref)

Vue3 Composition API 是透過 ref 或是 reactive 來定義資料

- reactive
- 用 ref() 定義響應式變量

reactive

將響應的資料以物件格式，放到 reactive() 中。

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

 const state = reactive({ count: 0 })

 function increment() {
  state.count++
 }
</script>

<template>
 <button @click="increment">
  {{ state.count }}
 </button>
</template>
```

- reactive() 的局限性
  - 僅對物件類型有效（物件、陣列和 Map 、 Set 這樣的集合類型），
    而對 string、number 和 boolean 這樣的 原始類型 無效。
  - 因為 Vue 的響應式系統是通過屬性訪問進行追踪的，
    因此必須始終保持對該響應式物件的相同引用。
    意味著 不可以隨意地替換 一個響應式物件，
    因為這將導致對初始引用的響應性連接丟失：

```
let state = reactive({ count: 0 })

// 上面的引用 ({ count: 0 }) 將不再被追蹤 (響應性連接已丟失！)
state = reactive({ count: 1 })
```

用 ref() 定義響應式變量

reactive() 的種種限制歸根結底是因為JavaScript 沒有可以作用於所有值類型的 引用 機制。

為此，Vue 提供了一個 ref() 方法來允許我們創建可以使用任何值類型的響應式ref：

```
import { ref } from 'vue';

const count = ref(0);
```

ref() 將傳入參數的值包裝為一個帶有 .value 屬性的 ref物件：

```
const count = ref(0);

console.log(count);     // { value: 0 }
console.log(count.value);  // 0

count.value++;
console.log(count.value);  // 1
```

和響應式物件的屬性類似，ref 的 .value 屬性也是響應式的。

同時，當值為物件類型時，會用 reactive() 自動轉換它的 .value。

- ref 在模板中的自動解包

  當ref 在模板中引用時，會自動解包，在模板中不需要使用 .value 。

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

 const count = ref(0);

 function increment() {
  count.value++;
 }
</script>

<template>
 <button @click="increment">
  {{ count }} <!-- 無需 .value -->
 </button>
</template>
```

14 計算屬性 computed

- 計算屬性 vs 方法
- 可寫入的計算屬性

如果在模板中寫太多邏輯，會讓模板變得臃腫，難以維護。

建議使用 計算屬性 來描述依賴響應式狀態的複雜邏輯。

```
<script setup>
 import { reactive, computed } from 'vue'
 const author = reactive({
  name: 'John Doe',
  books: [
   'Vue 2 - Advanced Guide',
   'Vue 3 - Basic Guide',
   'Vue 4 - The Mystery'
  ]
 })
 const publishedBooksMessage = computed(() => {
  return author.books.length > 0 ? 'Yes' : 'No'
 })
</script>

<template>
 <p>Has published books:</p>
 <span>{{ publishedBooksMessage }}</span>
</template>
```

計算屬性 vs 方法

計算屬性 Computed

- 基於響應式依賴，會將結果緩存。
- 計算屬性僅會在其響應式依賴更新時，才重新計算。

```
<p>{{ calculateBooksMessage }}</p>
```

```
const calculateBooksMessage = computed(() => {
  return author.books.length > 0 ? 'Yes' : 'No'
})
```

方法 Methods 在某些情境能達到相同效果。

- 每次調用，都會在重新渲染時再次執行函數。

```
<p>{{ calculateBooksMessage() }}</p>
```

```
function calculateBooksMessage() {
  return author.books.length > 0 ? 'Yes' : 'No'
}
```

可寫入的計算屬性

計算屬性默認是唯讀的。

欲修改一個唯讀的計算屬性時，會收到一個運行時警告。

若需要 **可寫**，可以通過提供 getter 和 setter 來設定：

```
import { ref, computed } from 'vue'
const firstName = ref('John')
const lastName = ref('Doe'  )
const fullName = computed({
 // getter
 get() {
  return firstName.value + ' ' + lastName.value
 },
 // setter
 set(newValue) {
  // 解構賦值語法
  [firstName.value, lastName.value] = newValue.split(' ')
 }
})
```

> 當運行 fullName.value = 'John Doe' 時，setter 會被調用，而 firstName 和 lastName 會隨之更新。

15

Class、CSS 綁定

Vue 專門為 **class** 和 **style** 的 **v-bind** 用法提供了特殊的功能增強。

除了字符串外，表達式的值也可以是物件或陣列。

- 綁定HTML class
- 綁定內聯樣式

綁定HTML Class

- 綁定物件
- 綁定陣列

綁定物件

- 直接賦值

```
<div :class="{ active: isActive }"></div>
```

- 傳遞物件

```
const classObject = reactive({
  active: true,
  'text-danger': false
})
```

```
<div :class="classObject"></div>
```

綁定陣列

- 渲染多個 Class

```
const activeClass = ref('active')
const errorClass = ref('text-danger')
```

```
<div :class="[activeClass, errorClass]"></div>
```

- 陣列中，使用物件

```
<div :class="[{ active: isActive }, errorClass]"></div>
```

綁定內聯樣式

- 綁定物件
- 綁定陣列

綁定物件

:style 支持綁定 JavaScript 物件值，對應的是 HTML元素的 style 屬性：

- 直接賦值

```
const activeColor = ref('red')
const fontSize = ref(30)
```

```
<div :style="{ color: activeColor, fontSize: fontSize + 'px' }"></div>
```

- 小駝峰 amelCase

```
<div :style="{ fontSize: fontSize + 'px' }"><div>
```

- 烤肉串 kebab-case

```
<div :style="{ 'font-size': fontSize + 'px' }"></div>
```

- 傳遞物件

```
const styleObject = reactive({
  color: 'red   ',
  fontSize: '13px'
})
```

```
<div :style="styleObject"></div>
```

綁定陣列

可以給 :style 綁定一個包含多個樣式物件的數組。

這些物件會被合併後渲染到同一元素上：

```
<div :style="[baseStyles, overridingStyles]"></div>
```

Thank you

2022

@Eason