



# ch4\_Vue\_Ecosystem

## Store

### 1. \$patch

`$patch` 是用來 **部分更新 store 的狀態**。它可以讓你一次性更新 store 中的一個或多個狀態屬性，而不需要逐一調用 mutation 或者直接修改每個屬性。`$patch` 允許你以更簡單和高效的方式更新 state。

用法：

- `$patch` 允許你傳遞一個對象，對應 store 中的狀態屬性，Pinia 會根據這個對象的內容更新狀態。
- 你也可以傳遞一個函數，這個函數會接受當前的 state 作為參數，並返回更新後的 state。

### 2. \$reset

`$reset` 是用來 **重置 store 的狀態**，將它恢復到在 state 定義時的初始狀態。這對於清除所有變更，並使 store 回到初始值非常有用。

`$patch` 用來 **部分更新** store 中的狀態。你可以傳遞一個對象或者函數來更新 store 的某些屬性。

`$reset` 用來 **重置** store 的狀態，將其恢復為初始定義時的狀態。

## Main.js

```
3   import { createApp } from 'vue'
4   import { createPinia } from 'pinia'
5
6   import App from './App.vue'
7   import router from './router'
8
9   const app = createApp(App)
10
11   app.use(createPinia())
12   app.use(router)
13
14   app.mount('#app')
```

是親戚但不是直系血親，要用use。

架構層放vue，其他重複地放components。

## Mitt

**1.Mitt** 是一個輕量級的 **Event Bus**庫，專為小型應用設計，通常用來在 JavaScript 或 Vue 應用程式中進行組件間的通信。Mitt 提供了一個簡單的 API 用來發送和接收事件，讓應用程式中的不同部分可以進行解耦的交互。

- 輕量級，沒有依賴。
- API 簡單且易於使用。
- 用於在不同模組或組件間進行事件觸發和監聽。

**2.Event Bus** 是一種通信模式，用來在應用程式的不同部分（通常是組件或模組）之間進行訊息或事件的傳遞。這種模式通常不需要直接引用或關聯到彼此的組件或模組，從而實現了鬆耦合。

- **發送端 (Publisher)**：觸發事件，通知其他部分有某些變化或動作。
- **接收端 (Subscriber)**：監聽事件並根據事件執行相應的處理邏輯。

### 3.語法

`emitter.emit(event, data)`：用來觸發某個事件，並傳遞資料（如果需要）。

`emitter.on(event, handler)`：用來註冊事件監聽器，當某個事件被觸發時，會執行對應的處理函數。

`emitter.off(event, handler)`：用來移除特定事件的監聽器。

#### 4. 沒有直接關係的組件要如何溝通？

1. 將會影響彼此的值，放在parent0

2. mitt(event bus)

透過共同的mitt(event bus)

一個組件發出自訂事件（屬性）：`bus.emit('自訂事件')`

另一組件聆聽這個事件（屬性）：`bus.on('自訂事件', callback)`

3. Pinia(狀態管理)

```
const bus = mitt() // 建立共同都認識的 mitt(event bus)
```

### 限制：

1. **無狀態管理功能**：Mitt 只是處理事件發送和接收，它本身不提供狀態管理功能。對於需要多狀態管理的應用，像 **Pinia** 或 **Vuex** 更合適。
2. **沒有事件隊列機制**：Mitt 觸發事件後不會有任何錯誤處理機制，所以如果沒有對應的監聽器來處理事件，這些事件就會丟失。

使用 Mitt 會使組件間的解耦變得更加簡單，但它不是一個完整的狀態管理解決方案。如果需要更複雜的狀態管理，則應考慮使用像 **Pinia** 或 **Vuex** 這樣的庫。

## Pinia

**Pinia** 是一個用於 **Vue 3** 的狀態管理庫。它是 Vue 3 推薦的狀態管理解決方案，取代了 Vuex（Vue 2 中的狀態管理庫）。

javascript

複製程式碼

```
import { ref, computed } from 'vue' // 從 Vue 庫中匯入 `ref` 和 `computed` 函數
import { defineStore } from 'pinia' // 從 Pinia 庫中匯入 `defineStore` 函數

// 定義一個名為 `counter` 的 store
export const useCounterStore = defineStore('counter', () => {
  const count = ref(0) // 定義一個響應式變數 `count`，初始值為 0
  const doubleCount = computed(() => count.value * 2) // 定義一個計算屬性 `doubleCount`，它
  function increment() { // 定義一個名為 `increment` 的函數
    count.value++ // 增加 `count` 的值
  }

  return { count, doubleCount, increment } // 返回這些變數和函數，讓它們可以在組件中使用
})
```

## Pinia 的特點：

1. **簡單易用**：Pinia 提供了簡單的 API，並且與 Vue 3 的 Composition API 完全兼容。它讓狀態管理變得更加直觀和簡單。
2. **響應式**：Pinia 基於 Vue 3 的響應式系統，這意味著在 Pinia 中管理的狀態會自動與組件的視圖保持同步。
3. **類型安全**：Pinia 具有很好的 TypeScript 支援，能夠提供類型檢查，讓開發者在開發過程中避免很多錯誤。
4. **模組化**：Pinia 支援建立多個 store，每個 store 都可以獨立處理不同的狀態和邏輯，這使得應用程式的狀態管理更加清晰和可維護。
5. **開發工具**：Pinia 有專門的開發者工具，能幫助你在開發過程中更輕鬆地調試和管理狀態。

## Pinia 和 Vuex 的區別：

- **簡化的 API**：Pinia 的 API 更加簡單，沒有像 Vuex 那樣需要使用 mutations 和 actions 來改變狀態。
- **更加現代化**：Pinia 是為 Vue 3 設計的，並完全基於 Composition API 和 Vue 3 的響應式系統，與 Vuex 不同，Vuex 是針對 Vue 2 設計的。
- **更好的 TypeScript 支援**：Pinia 原生支援 TypeScript，比 Vuex 更加靈活和強大。

在 **Pinia** 中，store 是一個用來管理 **全局狀態** 的容器。你可以把它想像成一個「資料中心」，它保存著應用程式中的狀態（比如變數、物件等），並提供修改這些狀態的方法。

## Pinia Store 的功能：

- **儲存狀態**：在 **store** 中，可以定義應用程式的狀態（如 **count**、**user** 等）。
- **提供方法**：可以在 **store** 中定義函數來修改狀態，這些方法稱為 **actions**。
- **計算屬性**：透過 **getters** 來創建基於狀態的計算值或衍生狀態。

**Pinia Store** 的狀態是全局的，可以在不同的組件中共享和修改。它提供了一個集中的方式來管理整個應用程式的狀態。

### 功能不同：

- **Pinia Store** 主要用於管理 **全局狀態**，也就是多個組件共享的資料或狀態。它解決的是跨組件的狀態管理問題。
- **Vue Component** 主要用於構建應用的 UI，每個組件有自己的 **局部狀態**，並處理組件範圍內的邏輯和渲染。

## Router

- router不是vue的東西，不用回傳。
- children的路徑不用寫斜線，會被當成資料夾而不是.vue檔案。

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

```
import Home from '../components/Home.vue'
```

```

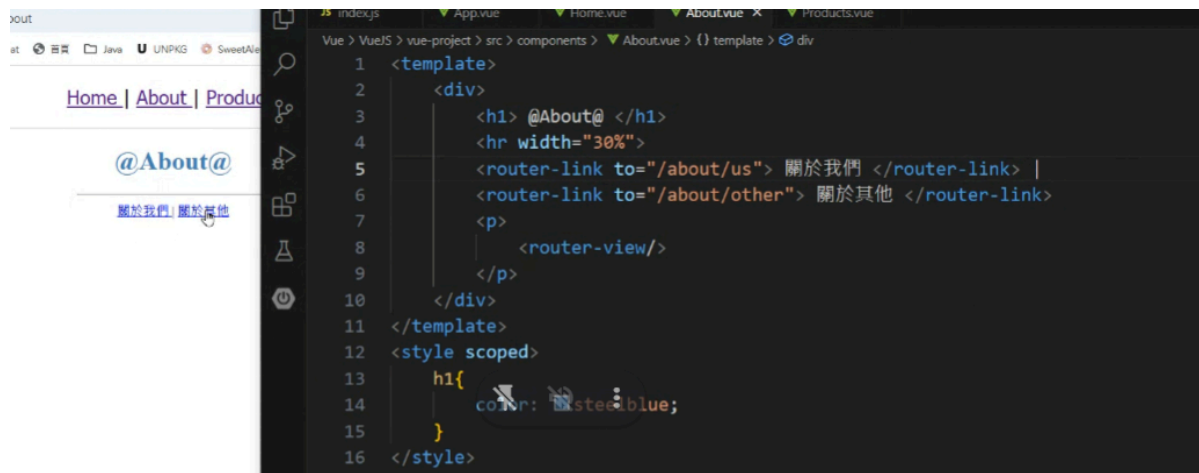
const routes = [    // 在哪個路徑下，render 哪個組件
  // Home 組件
  { path: '/', component: Home},

  // About 組件
  // { path: '/about', component: () => import('../components/About.vue') },
  {
    path: '/about',
    component: () => import('../components/About.vue'),
    children: [
      { path: '/us', component: () => import('../components/AboutUs.vue')},
      { path: '/other', component: () => import('../components/AboutOther.vue')},
    ],
  },

  // Products 組件
  // { path: '/products', component: () => import('../components/Products.vue')},
  {
    // path: '/products',
    // path: '/products/:sn',    // sn 是自訂名稱 (ex, id, item, sn, ...)
    path: '/products/:sn?',    // 加「?」代表後面的路徑可有可無
    component: () => import('../components/Products.vue'),
  },
]

```

- 單一網頁使用CSS的style 不想要影響到別人，記得加scope。



## 專案架構

開發在src，最後打包上線的是dist資料夾，在terminal做npm run dev。

### 開發環境 (Development Mode)

- 會顯示 **所有訊息**，包含錯誤（Error）、警告（Warning）、過時 API 提示（Deprecated）。
- 方便開發者修正問題。
- 例如在 JavaScript/Vue 可能會有：`console.warn("This API is deprecated.");`

## 正式環境（Production Mode）

- **只顯示錯誤（Errors）**，不會顯示警告或 Deprecated 訊息。
- 避免影響使用者體驗，不暴露內部細節。
- 例如，在 Vue / React / Node.js 這類框架，通常會設定 `NODE_ENV=production` 來過濾這些日誌。

## Storage


**Web Storage API**，它是瀏覽器提供的一種 API 用來在用戶的瀏覽器中存儲資料。Web Storage API 提供了兩種主要的儲存方式：

**localStorage**：用來儲存資料在瀏覽器的本地，資料將會持久保存，即使頁面刷新或瀏覽器關閉，資料仍然存在，直到手動清除。

**sessionStorage**：用來儲存資料在當前會話中，當瀏覽器或頁面被關閉時，資料會被刪除。

特性	localStorage	sessionStorage
作用範圍	資料持久化，直到手動刪除	僅在當前會話中有效，瀏覽器或頁面關閉後清除資料
存儲容量	約 5MB	約 5MB
存儲方式	以鍵值對的形式存儲	以鍵值對的形式存儲
跨頁面訪問	是	不是（只在同一窗口中有效）

javascript

 複製程式碼

```
localStorage.setItem('key', 'value'); // 儲存資料到 localStorage
sessionStorage.setItem('key', 'value'); // 儲存資料到 sessionStorage
```

很多components要一起變更屬性值，就可以使用Pinia。

**儲存資料 (setItem)：**用來將資料儲存到 localStorage 或 sessionStorage。

---