# 單元 18：前端架構——Vue 3 與 Pinia 狀態管理

前面我們學習了如何使用和部署這些自動化系統。但如果你想要客製化系統的介面，或者理解它是如何運作的，就需要了解一些前端開發的知識。

這個單元將介紹簡報編修系統的前端架構，包括 Vue 3 框架和 Pinia 狀態管理。即使你不是程式設計師，理解這些概念也能幫助你更好地使用和維護這個系統。

## 什麼是前端？

在 Web 應用程式中，「前端」指的是使用者直接看到和操作的部分——就是你在瀏覽器中看到的介面。按鈕、表單、動畫、版面配置，這些都是前端的工作。

「後端」則是在伺服器上執行的程式，負責處理資料、執行商業邏輯、與資料庫溝通等。

簡報編修系統是一個「前後端分離」的架構：

- **前端**：Vue 3 應用程式，負責使用者介面
- **後端**：Flask 應用程式，負責 AI 處理和 API

這種架構的好處是前端和後端可以獨立開發和部署，也更容易維護。

## Vue 3 簡介

Vue（發音類似「view」）是一個用於建構使用者介面的 JavaScript 框架。它是目前最受歡迎的前端框架之一，以學習曲線平緩和開發體驗優良著稱。

### 為什麼選擇 Vue？

**易學易用**： Vue 的設計理念是「漸進式」——你可以從簡單開始，逐步引入更多功能。不需要一開始就學會所有東西。

**元件化開發：** Vue 使用「元件」（Component）的概念來組織程式碼。每個元件是一個獨立的、可重用的 UI 單元，包含自己的 HTML 結構、CSS 樣式和 JavaScript 邏輯。

**響應式系統：** Vue 的響應式系統會自動追蹤資料的變化，當資料改變時，相關的 UI 會自動更新。你不需要手動操作 DOM。

**豐富的生態系統：** Vue 有完整的生態系統，包括路由（Vue Router）、狀態管理（Pinia）、開發工具等，可以應對各種規模的專案。

### Vue 3 的新特性

Vue 3 是 Vue 的最新主要版本，帶來了許多改進：

**Composition API：** 一種新的組織程式碼的方式，讓邏輯可以更好地重用和組織。簡報編修系統使用的就是 Composition API。

**更好的 TypeScript 支援：** Vue 3 對 TypeScript 有原生的良好支援，可以獲得更好的類型檢查和開發體驗。

**更快的效能：** Vue 3 的虛擬 DOM 和響應式系統都經過重寫，效能比 Vue 2 提升了許多。

## 專案結構

讓我們看看簡報編修系統的前端專案結構：

## views（頁面組件）

這個資料夾存放「頁面級別」的元件。每個頁面對應一個 Vue 檔案：

- **SplashView.vue**：系統啟動時顯示的畫面，通常會顯示 Logo 和載入動畫
- **HomeView.vue**：首頁，使用者在這裡上傳 PDF 或圖片
- **SelectionView.vue**：顯示所有頁面的縮圖，讓使用者選擇要處理的頁面
- **ProcessView.vue**：顯示 AI 處理的進度
- **ResultView.vue**：顯示處理結果
- **EditView.vue**：讓使用者編輯文字區塊
- **SettingsView.vue**：系統設定頁面

## components（可重用元件）

這個資料夾存放可以在多個頁面重複使用的元件：

- **ErrorHelpModal.vue**：錯誤說明的彈窗元件
- **TextBlock.vue**：文字區塊元件
- **PropertyPanel.vue**：屬性面板元件

## stores（狀態管理）

這個資料夾存放 Pinia store，用於管理應用程式的狀態。後面會詳細介紹。

## router（路由配置）

這個資料夾存放 Vue Router 的配置，定義了 URL 和頁面的對應關係。

## App.vue（根元件）

這是整個應用程式的根元件，包含了共享的 UI 元素（如浮動按鈕、背景動畫）和路由視圖。

## 單檔案元件（SFC）

Vue 使用「單檔案元件」（Single File Component，簡稱 SFC）的格式來組織程式碼。一個 `.vue` 檔案包含三個部分：

```
<template>
  <!-- HTML 結構 -->
</template>

<script setup lang="ts">
// JavaScript/TypeScript 邏輯
</script>

<style scoped>
/* CSS 樣式 */
</style>
```

### template 區塊

這裡定義元件的 HTML 結構。Vue 的模板語法讓你可以在 HTML 中使用動態內容：

```
<template>
  <div class="container">
    <h1>{{ title }}</h1>
    <button @click="handleClick">點擊我</button>
    <ul>
      <li v-for="item in items" :key="item.id">
        {{ item.name }}
      </li>
    </ul>
  </div>
</template>
```

關鍵語法：

- `{{ }}` - 插值，顯示變數的值

- `@click` - 事件綁定，等同於 `v-on:click`
- `v-for` - 迴圈，重複渲染元素
- `:key` - 動態屬性綁定，等同於 `v-bind:key`

## script 區塊

這裡定義元件的邏輯。使用 `<script setup>` 語法可以更簡潔地撰寫：

```
<script setup lang="ts">
import { ref, computed } from 'vue'

// 響應式狀態
const count = ref(0)
const items = ref([
  { id: 1, name: '項目一' },
  { id: 2, name: '項目二' }
])

// 計算屬性
const doubleCount = computed(() => count.value * 2)

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

關鍵概念：

- `ref()` - 建立響應式的變數
- `computed()` - 建立計算屬性，會自動追蹤依賴並快取結果
- `lang="ts"` - 使用 TypeScript

## style 區塊

這裡定義元件的 CSS 樣式。`scoped` 屬性讓樣式只作用於當前元件，不會影響其他元件：

```
<style scoped>
.container {
  padding: 20px;
```

```
}

h1 {
  color: #333;
}
</style>
```

## Pinia 狀態管理

當應用程式變得複雜時，會有很多狀態（資料）需要在多個元件之間共享。如果每個元件都自己管理狀態，會變得很混亂。

Pinia 是 Vue 3 官方推薦的狀態管理庫。它提供一個集中的地方來管理應用程式的狀態，讓資料流更清晰、更可預測。

### 什麼是 Store？

在 Pinia 中，「Store」是一個存放狀態的容器。它包含：

- **State**（狀態）：應用程式的資料
- **Getters**（計算屬性）：從狀態衍生的資料
- **Actions**（方法）：修改狀態的函式

### 簡報編修系統的 Store

讓我們看看簡報編修系統的 `app.ts` store：

```
import { defineStore } from 'pinia'
import { ref, computed } from 'vue'

export const useAppStore = defineStore('app', () => {
  // 狀態
  const pendingItems = ref<PageItem[]>([])
  const results = ref<ProcessResult[]>([])
  const processing = ref(false)
  const currentPage = ref(0)
  const totalPages = ref(0)

  // 計算屬性
  const selectedItems = computed(() =>
    pendingItems.value.filter(item => item.selected)
  )
```

```
  const selectedCount = computed(() =>
    selectedItems.value.length
  )

  // 方法
  function addPendingItems(items: PageItem[]) {
    pendingItems.value.push(...items)
  }

  function reset() {
    pendingItems.value = []
    results.value = []
    processing.value = false
  }

  return {
    pendingItems,
    results,
    processing,
    selectedItems,
    selectedCount,
    addPendingItems,
    reset
  }
})
```

## Store 的結構解析

**定義 Store：**

```
export const useAppStore = defineStore('app', () => {
  // ...
})
```

`defineStore` 函式建立一個 store。第一個參數 `'app'` 是 store 的 ID，必須是唯一的。第二個參數是一個函式，回傳 store 的內容。

**狀態（State）：**

```
const pendingItems = ref<PageItem[]>([])
const processing = ref(false)
```

使用 `ref()` 定義響應式的狀態。`<PageItem[]>` 是 TypeScript 的類型註解，表示這是一個 PageItem 陣列。

**計算屬性（Getters）：**

```
const selectedItems = computed(() =>
  pendingItems.value.filter(item => item.selected)
)
```

使用 `computed()` 定義計算屬性。它會自動追蹤依賴（這裡是 `pendingItems`），當依賴改變時重新計算。

**方法（Actions）：**

```
function addPendingItems(items: PageItem[]) {
  pendingItems.value.push(...items)
}
```

普通的函式，用於修改狀態。

**匯出：**

```
return {
  pendingItems,
  results,
  selectedItems,
  addPendingItems,
  reset
}
```

把需要公開的狀態、計算屬性和方法回傳。

## 在元件中使用 Store

在元件中使用 store 非常簡單：

```
<script setup lang="ts">
import { useAppStore } from '@/stores/app'
```

```
const store = useAppStore()

// 存取狀態
console.log(store.pendingItems)

// 存取計算屬性
console.log(store.selectedCount)

// 呼叫方法
store.addPendingItems([...])
store.reset()
</script>

<template>
  <div>
    <p>已選擇 {{ store.selectedCount }} 頁</p>
    <button @click="store.selectAll()">全選</button>
  </div>
</template>
```

## 資料介面（Interface）

TypeScript 的介面（Interface）用於定義資料的結構。這讓程式碼更清晰，也能在編輯器中獲得更好的提示。

簡報編修系統定義了幾個重要的介面：

### PageItem（頁面項目）

```
export interface PageItem {
  id: string
  index: number
  type: 'pdf' | 'image'
  name: string
  thumb: string        // Base64 縮圖
  original: string     // Base64 原始圖
  selected: boolean
  pdfTextData?: {      // PDF 文字數據（可選）
    blocks: TextBlock[]
    pageWidth: number
    pageHeight: number
```

```
  }
}
```

這個介面描述了「待處理的頁面」的資料結構，包含：

- 基本資訊（ID、索引、類型、名稱）
- 圖片資料（縮圖、原始圖）
- 選擇狀態
- PDF 文字資料（如果來源是 PDF）

## TextBlock（文字區塊）

```
export interface TextBlock {
  id: string            // 唯一識別碼
  text: string          // 文字內容
  x: number             // X 座標（百分比）
  y: number             // Y 座標（百分比）
  width: number         // 寬度（百分比）
  height: number        // 高度（百分比）
  fontSize: number      // 字體大小
  fontName?: string     // 字體名稱（可選）
  color?: string        // 文字顏色（可選）
  fontWeight?: 'normal' | 'bold'
  fontStyle?: 'normal' | 'italic'
  textAlign?: 'left' | 'center' | 'right'
}
```

這個介面描述了「文字區塊」的資料結構，用於在簡報上定位和顯示文字。

注意座標使用「百分比」而不是「像素」——這樣不管簡報的實際尺寸是多少，文字都能正確定位。

## ProcessResult（處理結果）

```
export interface ProcessResult {
  pageIndex: number
  name: string
  original: string      // Base64 原始圖
  cleaned: string       // Base64 去字圖
  sourceType: 'pdf' | 'image'
```

```
  pdfTextData?: PageItem['pdfTextData']
  ocrBlocks?: any[]     // OCR 結果
  error?: string        // 錯誤訊息（如果有）
}
```

這個介面描述了 AI 處理後的結果，包含原始圖、處理後的圖、文字資料，以及可能的錯誤訊息。

## 應用程式流程

理解了這些概念後，讓我們看看整個應用程式的流程是如何運作的：

### 1. 啟動

使用者開啟 http://localhost:5173

1. Vue Router 根據 URL（`/`）顯示 SplashView
2. SplashView 顯示啟動畫面幾秒鐘
3. 自動跳轉到 HomeView

### 2. 上傳檔案

使用者在 HomeView 上傳 PDF 或圖片

1. 前端使用 pdfjs-dist 解析 PDF，將每頁轉換成圖片
2. 呼叫 `store.addPendingItems()` 把頁面資料加入 store
3. 跳轉到 SelectionView

### 3. 選擇頁面

使用者在 SelectionView 選擇要處理的頁面

1. 從 `store.pendingItems` 讀取所有頁面
2. 顯示縮圖，讓使用者勾選
3. 使用者點擊「開始處理」
4. 呼叫 `store.startProcessing()`
5. 跳轉到 ProcessView

### 4. AI 處理

ProcessView 顯示處理進度

1. 從 `store.selectedItems` 取得選中的頁面
2. 逐頁送到後端 API 處理
3. 呼叫 `store.updateProgress()` 更新進度
4. 呼叫 `store.addResult()` 儲存處理結果
5. 所有頁面處理完成後，跳轉到 ResultView

### 5. 查看結果

使用者在 ResultView 查看處理結果

1. 從 `store.results` 讀取處理結果
2. 顯示成功和失敗的頁面
3. 使用者可以選擇「編輯」或「匯出」

### 6. 編輯（可選）

使用者在 EditView 編輯文字區塊

1. 從 `store.getPageData()` 取得頁面資料
2. 顯示背景圖和文字區塊
3. 使用者可以移動、調整、刪除、新增文字區塊
4. 呼叫 `store.updateTextBlock()` 等方法更新狀態

### 7. 匯出

使用者點擊「匯出 PPTX」

1. 從 `store.allPagesInOrder` 取得所有頁面資料
2. 使用 pptxgenjs 生成 PowerPoint 檔案
3. 下載到使用者的電腦

## 響應式系統的運作

Vue 的響應式系統是它的核心特色。讓我們理解它是如何運作的：

### 追蹤依賴

當你在模板中使用一個響應式變數時，Vue 會自動追蹤這個「依賴關係」：

```
<template>
  <p>已選擇 {{ store.selectedCount }} 頁</p>
</template>
```

Vue 知道這個 `<p>` 元素依賴於 `store.selectedCount`。

### 觸發更新

當 `store.selectedCount` 的值改變時（例如使用者選擇了更多頁面），Vue 會自動重新渲染這個 `<p>` 元素，顯示新的數字。

你不需要手動操作 DOM（例如 `document.querySelector('p').textContent = newValue`），Vue 會處理好這一切。

### 計算屬性的快取

計算屬性會被快取。只有當它的依賴改變時，才會重新計算：

```
const selectedCount = computed(() => selectedItems.value.length)
```

如果 `selectedItems` 沒有改變，多次存取 `selectedCount` 會直接回傳快取的值，不會重複計算。

## 如何客製化

如果你想要客製化系統的介面，可以從以下幾個方向著手：

### 修改樣式

最簡單的客製化是修改 CSS。每個元件的 `<style>` 區塊定義了它的樣式。

例如，修改按鈕的顏色：

```
<style scoped>
.primary-btn {
  background: #your-brand-color;
```

```
}
</style>
```

### 修改文字

介面上的文字都在 `<template>` 區塊中。找到對應的元件，修改其中的文字即可。

### 新增功能

如果要新增功能，通常需要：

1. 在 store 中新增狀態和方法
2. 在元件中使用這些狀態和方法
3. 在模板中新增對應的 UI

### 新增頁面

如果要新增新的頁面：

1. 在 `views/` 資料夾建立新的 `.vue` 檔案
2. 在 router 配置中新增路由
3. 如果需要，在 store 中新增相關的狀態

## 本章小結

本章介紹了簡報編修系統的前端架構：

**Vue 3 框架**：使用元件化開發和響應式系統，讓建構使用者介面更加高效

**單檔案元件**：將 HTML、JavaScript、CSS 組織在同一個檔案中，結構清晰

**Pinia 狀態管理**：集中管理應用程式的狀態，讓資料流更清晰、更可預測

**資料介面**：使用 TypeScript 介面定義資料結構，提高程式碼的可讀性和可維護性

**應用程式流程**：從啟動到匯出的完整流程，以及各個元件如何協作

理解這些概念可以幫助你：

- 更好地理解系統是如何運作的

- 在需要時進行客製化
- 排除前端相關的問題

下一個單元，我們會介紹後端架構，包括 Flask 框架和 Gemini API 的整合。