在 Web 前端開發三大框架中，為何要選擇學習 Vue.js 呢？首先，我們先來看 Vue.js 官方網站上的介紹的三大優點：

- **易用性**

  只要具備 HTML、CSS 及 Javascript 的基礎知識，可以透過簡單的教學文件快速上手。

- **高效性**

  前端網頁在傳統的開發上，常使用 Javascript 或 jQuery 進行 DOM 的操作。當網頁的元件或操作變為複雜時，容易有效能低落的情形。Vue.js 透過框架內的優化，提升網頁的執行效率。

- **豐富的生態體系**

  隨著使用 Vue.js 的人越來越多，在 Vue.js 有著豐富的生態體系，讓開發人員能夠容易找著需要的套件，加速開發的效率。

除此之外，Vue.js 結合了 React.js 與 Angular.js 框架中的優點，它不僅具備了 React.js 的單一資料流的畫面渲染機制，同時，也具備了 Angular.js 的資料雙向綁定的畫面宣染機制。因此，對於原本為 React.js 或 Angular.js 的開發者來說，若想擁有另一個框架好處時，進入 Vue.js 的世界，無疑又是一大誘因。談到這裡，讀者可能不明白何謂「單向資料流」及「資料雙向綁定」。這二大特性，我們將於後面詳細介紹。

1-2 程式架構模式

在 Web 應用程式中，包含使用者的介面開發，以及應用程式中的資料及流程處理。使用者介面的呈現可以交由視覺設計師進行網頁視覺的設計，系統的資料及流程處理由程式設計師進行程式的撰寫。當我們學習 Vue.js 之前，我們須先了解框架為我們處理了什麼事情。先前我

們討論到框架具有規範，幫助開發人員能有相同的共識，以提升開發的效率與可維護性。框架中所訂定的規範中，為了將前後端人員處理職責作清楚的劃分，常以 MVC 架構為主。然而，因著 MVC 架構中仍然有不足的部份，故近年來開始有了新的 MVVM、Flux 等概念的框架出現。在本節中，將向讀者介相關的程式架構模式。

1-2-1 MVC

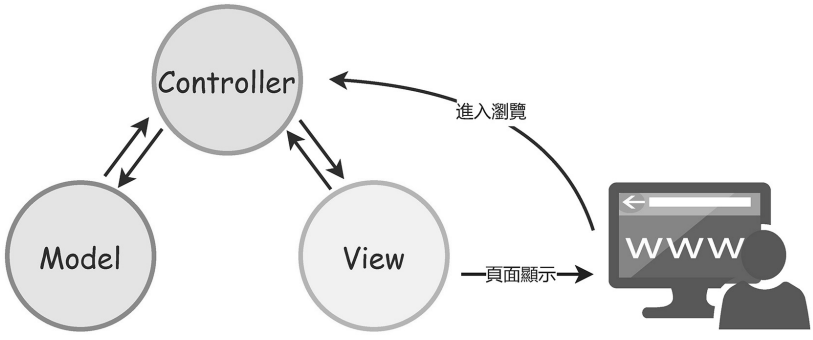

圖 1-5 MVC 架構

在 MVC 架構中，Web 應用程式可分為 Model（資料模型）、View（視圖）及 Controller（控制器），這三者的簡稱即為 MVC，下面我們來介紹這三者負責的部份。

- **Model（資料模型）**

  資料模型的職責在於資料的管理，將 Web 應用程式中會使用到的資料進行包裝、儲存，供系統內部程式使用，通常會搭配 MySQL、Microsoft SQL Server 等資料庫軟體使用。

- **View（視圖）**

  視圖主要專注於呈現給使用者的操作介面，在 Web 應用程式中，主要為 HTML 樣板。當 HTML 樣板需要呈現動態資訊時，會由框

架提供的模板引擎，透過取得資料模型，顯示取得的資料，供使用者瀏覽。

◉ Controller（控制器）

控制器主要處理 Web 應用程式裡各項作業的流程。當使用者操作處於不同階段時，控制器定義使用者應看見 View（視圖）的頁面，並提取所需的資料給 View（視圖）呈現給使用者觀看。

綜合以上三個部份，如圖所示，當使用者進入 Web 應用程式進行瀏覽時，Controller（控制器）判斷使用者應看見的頁面，從 Model（資料模型）取得頁面所需的後，提供給 View（視圖）中的 HTML 樣板，呈現頁面供使用者瀏覽。這樣子的架構下的優缺點如下：

◉ 優點

- **職責分離**

  協助開發人員進行職責分離，不論開發人員負責 Model、View 或 Controller 的，都可專注在自己負責的部份。

- **可重用性**

  MVC 各自的部份透過職責分離，可將不同的部份拆解成數個元件進行開發，使各元件可重複利用。

- **提升擴充性及維護性**

  由於 MVC 各自獨立，開發人員可在不影響整體架構下，針對新需求進行擴充或修改。

◉ 缺點

- **效能降底**

  由於 MVC 的職責區分下，原本可直接連接資料庫的程式，變得須透過 Model 的程式進行連結，程式變得肥大，導致程式執行效能下降。

- **容易關聯複雜**

  Web 應用程式在 MVC 架構下，開發越來越龐大時，若沒有合適的規劃，會導致系統中各元件關聯及複雜度增加，導致系統在修復或追蹤問題時變得更為不易。

1-2-2 Flux – 單向資料流

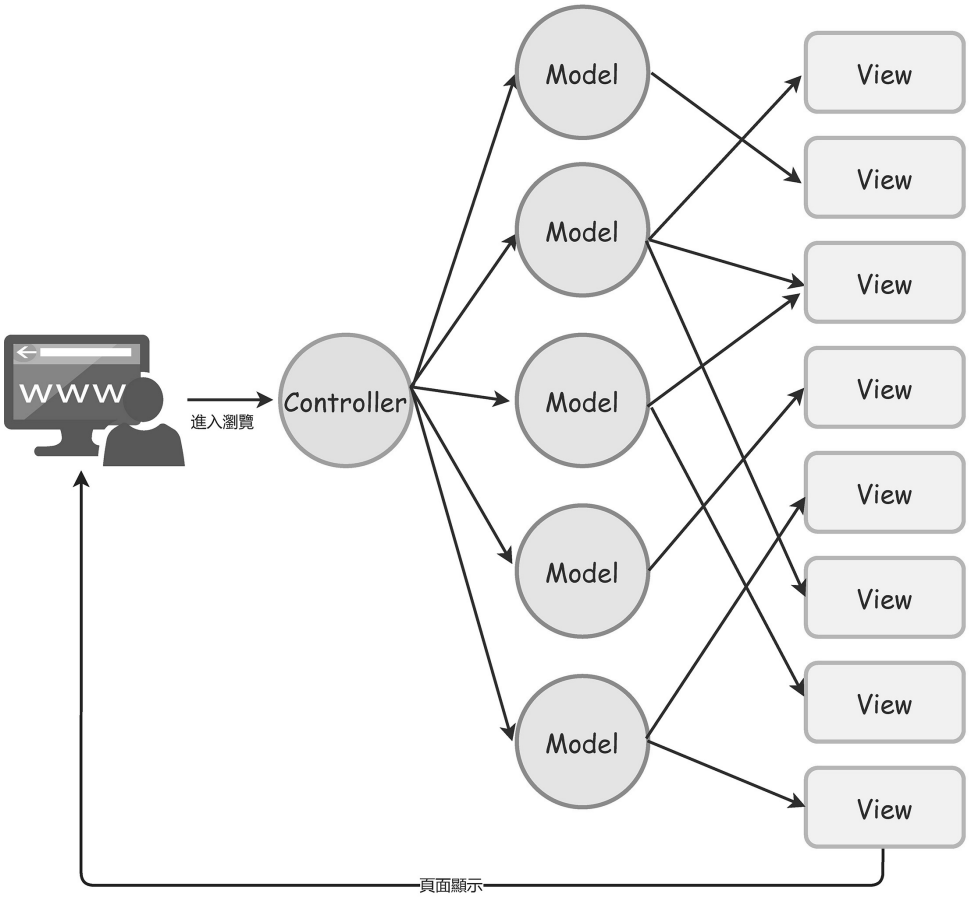

圖 1-6　MVC 框架問題

傳統的 Web 應用程式開發中，我們常以 MVC 架構進行開發。然而，在 MVC 架構下的專案，當專案變得越來越龐大時，由於未定義資料該如何流動進行畫面的渲染，如上圖所示，由於多個 Model 與 View 產生對應、多個 View 與 Model 產生對應，導致資料流向複雜，開發人員產生混亂。為了解決這樣開發的問題，Facebook 於 2014 年在一場大會中提出 Flux 的概念，以解決資料流凌亂的問題。

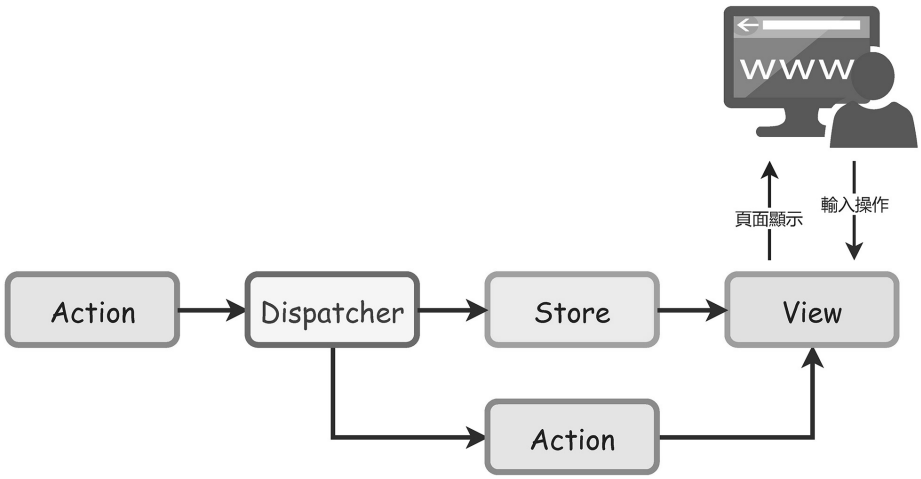

圖 1-7　Flux 單向資料流

如上圖所示，Flux 具有 View、Action、Dispatcher、Store 4 個部份，以下將作詳細的介紹：

- View

  View 的職責在於使用者介面及在介面中監聽使用者的各項操作

- Action

  Action 定義使用者於介面上操作動作的內容，及相關資料的操作

06

單頁式應用程式（SPA）

6-1 Vue.js SPA 架構

開發大型 Web 應用程式系統時，資料、頁面元件會越來越多，使得程式會變得更加複雜。這時候使用 SPA 架構將會協助我們更有效地管理 Vue.js 的程式碼。SPA 為 Single Page Application 的縮寫，意即單頁式應用程式。

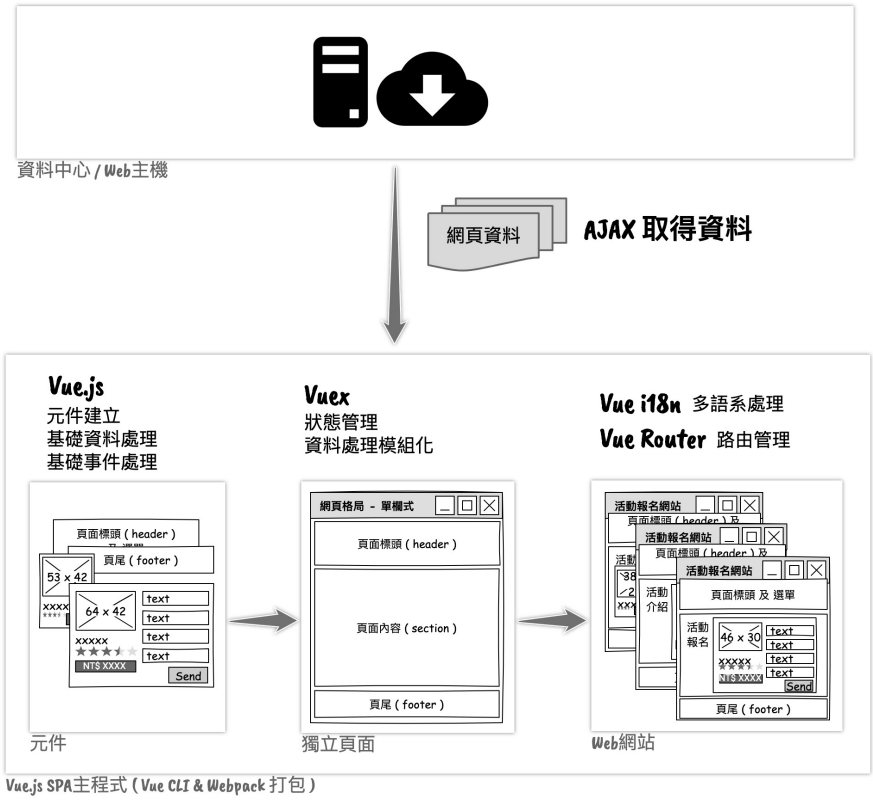

圖 6-1　SPA 架構圖

使用 Vue.js 開發大型 Web 應用程式時，SPA 架構概念圖如圖 6-1 所示，系統中的資料將會存於雲端的資料中心，當使用者使用瀏覽器進入系統時，網頁載入的 Vue.js SPA 程式將依頁面需求使用 AJAX 技術載入需要的資料。Vue.js SPA 主程式使用以下 JS 套件：

- Webpack & Vue CLI

  Vue CLI 可協助開發人員快速建立前端專案，Vue CLI 建立的專案，會有基礎的檔案結構，並搭配 Webpack 套件使撰寫的 Vue.js、CSS 等程式打包輸出，供網頁檔使用。Vue.js CLI 使用方法將於本章向讀者介紹。

- Vue.js

  使用 Vue.js 可有效地建立網頁元件，並且在每個元件中建立資料模型、監聽事件並作資料處理，其使用方式為本書基礎篇主要介紹的內容。

- AJAX

  JavaScript 程式向雲端的資料中心取得資料時，須透過 AJAX 技術，發送 HTTP 或 HTTPS 請求向伺服器要資料，有關 AJAX 技術，將於本書第 7 章介紹。

- Vuex

  Web 應用程式開發的功能越多時，各頁面的資訊將會越來越多，程式將越來越肥大，且元件資料的傳遞可能越來越多層。透過 Vuex 可建立全域共用的狀態資訊，讓元件可直接讀取，不須透過 props 方式傳遞。此外，還可以建立模組便於分類管理程式，相關內容將於本書第 8 章介紹。

- Vue-Router

  透過 Vue-Router 套件可賦予頁面元件實體的 URL。當使用者輸入網址時，可導至對應的頁面元件，相關內容將於本書第 9 章介紹。

- Vue-i18n

  當開發的 Web 應用程式需要支援多國語系時，可使用 Vue-i18n 套件，相關內容將於本書第 10 章介紹。

接下來的內容，將以本書基礎篇 Vue.js 功能為基底，介紹 SPA 各部份的使用方式。讓我們一起建構 Vue.js SPA 程式吧！

6-2 Vue CLI 與 Webpack

Vue CLI 是為了協助 Vue.js 開發的命令列介面（Command Line Interface），可快速且簡單地建構 Vue.js 的開發環境。透過 Vue CLI 建構的開發環境，具有以下特色：

- 具有 SPA 架構的檔案結構，可將原始檔分類管理
- 結合 Webpack 功能，可支援 SASS 及 JavaScripti ES6 以上版本的語法解析
- 具有 Hot Reload 功能，在開發時可即時看見打包完後的 js 或 cssl 效果
- 可設置 ES Lint 統一開發人員 JavaScript 語法風格

Vue CLI 構建的 Vue.js 專案採用 SPA 架構，將原始碼模組化並拆分為數個獨立檔案進行管理。由於 Vue.js SPA 架構的原始碼使用了 JavaScript ES6、Vue SFC、SASS 等語法導致瀏覽器無法直接載入。因此，Vue CLI 使用打包工具－Webpack 將程式轉換為瀏覽器可讀的程式。

Webpack

圖 6-2　Webpack 官方網站（網址：https://webpack.js.org/）

Webpack 為 Web 網頁前端開發使用的打包工具，目前最新版本為 Webpack 5，官方網站如圖 6-2 所示。Webpack 可將模組化的眾多檔案轉換、結合並打包成一包檔案。

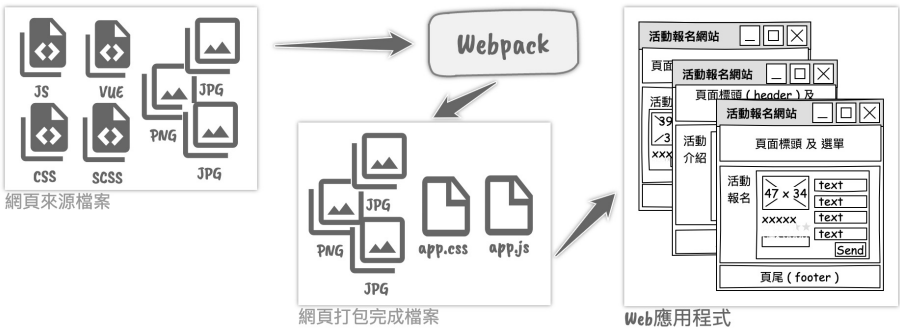

圖 6-3　Webpack 打包概念圖

Webpack 具有各類的 loader，不僅處理 JavaScript 檔案的打包，它也可依據載入的 loader 處理 vue 檔、sass 檔、scss 檔、圖片…等網頁資源。Vue.js SPA 架構中透過 Webpack 打包如圖 6-3 所示，Webpack 將原始程式碼打包時主要進行以下處理：

- Vue.js SFC 檔透過 vue-loader 轉換為 JavaScript 可執行的程式碼
- js 檔透過 babel-loader 將 JavaScript 語法轉換為瀏覽器可執行 ES5 語法
- sass 檔透過 sass-loader 將 sass 樣式檔轉換為 css 檔
- JavaScript 及 CSS 檔將分別整合為獨立檔案供網頁載入

Node.js 與 Vue CLI 安裝

使用本章介紹的 Vue CLI 與 Webpack 時，必須在本機安裝 Node.js。Node.js 採用 Chrome 的 V8 JavaScript 引擎，可建立本機執行 JavaScript 的開發環境。Node.js 官方網站如圖 6-4 所示，目前最新長期維護版本

（LTS）為 16.17.0。Node.js 的安裝方法可參考本書附錄 A-4 JavaScript 套件管理工具安裝。Node.js 安裝完成後，建立的 JavaScript 執行環境具有 NPM 套件管理指令，可管理本機或專案的 JavaScript 套件。有關 JavaScript 套件管理詳細介紹可參考本書附錄 B JavaScript 套件管理。

圖 6-4　Node.js 官方網站

Vue CLI 安裝時，須至 Mac 的終端機或 Windows 的命令提示字元執行指令「npm install @vue/cli -g」。

```
➜  ch06 git:(master) ✗ npm install @vue/cli -g
npm WARN config global `--global`, `--local` are deprecated. Use `--location=global` instead.
(                ) ⠸ idealTree:tar-stream: sill placeDep node_modules/@vue/cli wcwidth@1.0.1 OK for: ora
```

圖 6-5　安裝 Vue CLI

執行完上述指令後，可執行指令「vue --version」確認已安裝的 Vue CLI 版本為 5.0.8。

```
➜  ch06 git:(master) ✗ vue --version
@vue/cli 5.0.8
➜  ch06 git:(master) ✗
```

圖 6-6　Vue CLI 版本確認

Vue CLI 創建專案

接下來將介紹使用 Vue CLI 創建 Vue.js SPA 專案。Vue CLI 創建專案的語法如下：

SHELL

```
# 建立 Vue 專案
vue create [專案名稱]
```

以上述指令執行後，Vue CLI 將會詢問問題，以確認 Vue.js SPA 開發環境的需求。創建專案大致上可分為預設及手動二種方式。

首先，以創建「hello-spa」專案介紹預設專案創建。當我們輸入「vue create hello-spa」指令後，Vue CLI 將會顯示如圖 6-7 的問題，詢問要使用哪個樣板。在此，我們選擇 Vue 3 的預設樣板。

```
Vue CLI v5.0.8
? Please pick a preset: (Use arrow keys)
❯ Default ([Vue 3] babel, eslint)
  Default ([Vue 2] babel, eslint)
  Manually select features
```

圖 6-7　預設專案創建 - 詢問使用樣板

接著，Vue CLI 詢問要使用哪個套件管理工具， Yarn 與 NPM 均是 JavaScript 的套件管理工具，由於 Yarn 的執行效率較好，本書建議讀者可使用 Yarn 為主。

```
Vue CLI v5.0.8
? Please pick a preset: Default ([Vue 3] babel, eslint)
? Pick the package manager to use when installing dependencies: (Use arrow keys)
❯ Use Yarn
  Use NPM
```

圖 6-8　預設專案創建 - 選擇套件管理工具

選擇完套件管理工具後，Vue CLI 將自動建立專案資料夾並執行專案初始化。

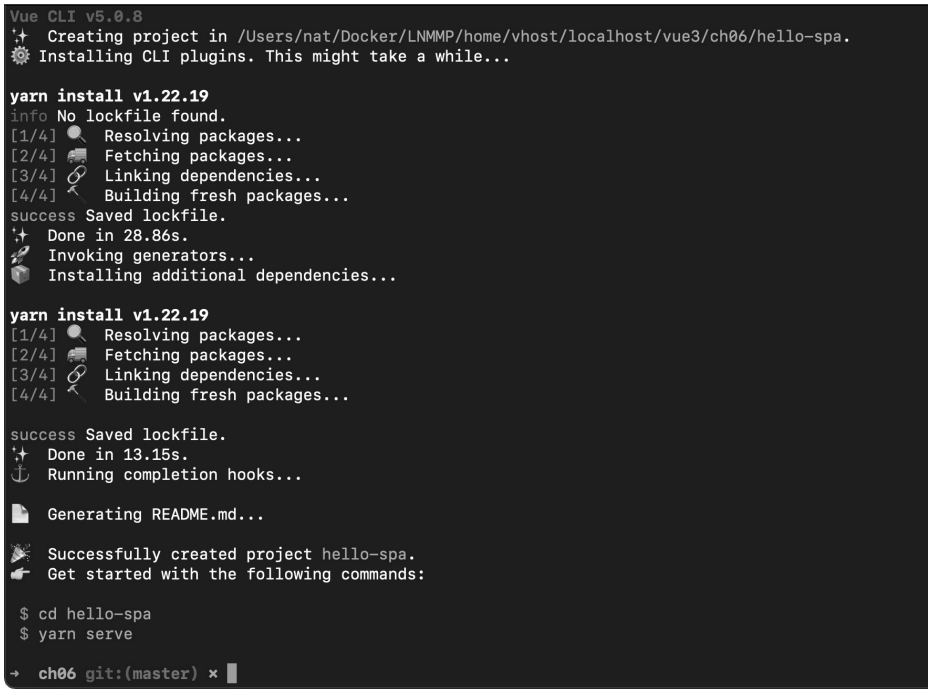

圖 6-9　預設專案創建 - 專案創建完成

前面介紹以預設樣板的創建專案以幾個步驟便可快速建立專案。接著，將透過建立「hello-spa-manually」專案介紹手動創建。當輸入「vue create hello-spa-manually」後，在詢問樣板的地方可選擇「Manually select features」進入手動選擇模式。

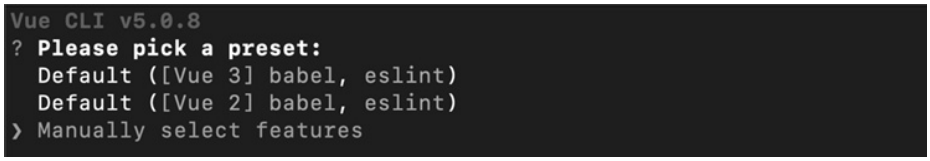

圖 6-10　手動專案創建 - 詢問使用樣板

進入手動選擇模式後，Vue CLI 詢問專案需用套件，預設有 Babel 及 Linter / Formatter。這裡可依需求增加 CSS Pre-processors，讓專案支援進階的 css 預處理器。

```
Vue CLI v5.0.8
? Please pick a preset: Manually select features
? Check the features needed for your project: (Press <space> to select, <a> to
toggle all, <i> to invert selection, and <enter> to proceed)
 ◉ Babel
 ◯ TypeScript
 ◯ Progressive Web App (PWA) Support
 ◯ Router
 ◯ Vuex
❯◉ CSS Pre-processors
 ◉ Linter / Formatter
 ◯ Unit Testing
 ◯ E2E Testing
```

圖 6-11　手動專案創建 - 選擇專案需用套件

接著，Vue CLI 將詢問專案使用的 Vue.js 版本，在此選擇 3.x 版。

```
Vue CLI v5.0.8
? Please pick a preset: Manually select features
? Check the features needed for your project: Babel, CSS Pre-processors, Linter
? Choose a version of Vue.js that you want to start the project with (Use arrow
keys)
❯ 3.x
  2.x
```

圖 6-12　手動專案創建 - 選擇 Vue.js 版本

Vue CLI 提供 Sass/SCSS、Less、Stylus 等 3 個 CSS 預處理器，這裡可選擇使用常見的 Sass/SCSS。

```
Vue CLI v5.0.8
? Please pick a preset: Manually select features
? Check the features needed for your project: Babel, CSS Pre-processors, Linter
? Choose a version of Vue.js that you want to start the project with 3.x
? Pick a CSS pre-processor (PostCSS, Autoprefixer and CSS Modules are supported
by default): (Use arrow keys)
❯ Sass/SCSS (with dart-sass)
  Less
  Stylus
```

圖 6-13　手動專案創建 - 選擇 CSS 預處理器

linter / formater 的選項可選擇 ESLint 的預設設定範本。ESLint 為決定專案中 JavaScript 程式撰寫風格，它建立一套統一的規範，讓專案的開發人員可以依循，使程式變得更易閱讀及管理。本書選擇 ESLint + Airbnb config，後續的範例將以此為主。

```
Vue CLI v5.0.8
? Please pick a preset: Manually select features
? Check the features needed for your project: Babel, CSS Pre-processors, Linter
? Choose a version of Vue.js that you want to start the project with 3.x
? Pick a CSS pre-processor (PostCSS, Autoprefixer and CSS Modules are supported
by default): Sass/SCSS (with dart-sass)
? Pick a linter / formatter config:
  ESLint with error prevention only
❯ ESLint + Airbnb config
  ESLint + Standard config
  ESLint + Prettier
```

圖 6-14　手動專案創建 - 選擇 ESLint 規範範本

決定好 ESLint 規範的範本後，接著將選擇檢查程式碼風格的時機，「Lint on save」為存檔時檢查，「Lint and fix on commit」為提交程式碼時檢查。本書選擇第 1 個存檔時檢查，讀者可以二者都選擇。

```
Vue CLI v5.0.8
? Please pick a preset: Manually select features
? Check the features needed for your project: Babel, CSS Pre-processors, Linter
? Choose a version of Vue.js that you want to start the project with 3.x
? Pick a CSS pre-processor (PostCSS, Autoprefixer and CSS Modules are supported
by default): Sass/SCSS (with dart-sass)
? Pick a linter / formatter config: Airbnb
? Pick additional lint features: (Press <space> to select, <a> to toggle all,
<i> to invert selection, and <enter> to proceed)
❯◉ Lint on save
 ◯ Lint and fix on commit
```

圖 6-15　手動專案創建 - 選擇 ESLint 檢查時機

Babel、ESLint 等套件的設定檔放置位可選擇「In dedicated config files」，讓每個套件有一個獨立的設定檔。若讀者希望全部存至 package.json，可選擇第二個選項「In package.json」。

```
Vue CLI v5.0.8
? Please pick a preset: Manually select features
? Check the features needed for your project: Babel, CSS Pre-processors, Linter
? Choose a version of Vue.js that you want to start the project with 3.x
? Pick a CSS pre-processor (PostCSS, Autoprefixer and CSS Modules are supported
by default): Sass/SCSS (with dart-sass)
? Pick a linter / formatter config: Airbnb
? Pick additional lint features: Lint on save
? Where do you prefer placing config for Babel, ESLint, etc.?
❯ In dedicated config files
  In package.json
```

圖 6-16　手動專案創建 - 選擇套件設定檔儲存位置

最後，可選擇目前的設定檔是否要存為範本，存為範本後，可給下次建立時選擇，如同預設專案創建的方式一樣，會加快創建速度。在此，先選擇 n 代表不存為樣板。

```
Vue CLI v5.0.8
? Please pick a preset: Manually select features
? Check the features needed for your project: Babel, CSS Pre-processors, Linter
? Choose a version of Vue.js that you want to start the project with 3.x
? Pick a CSS pre-processor (PostCSS, Autoprefixer and CSS Modules are supported
by default): Sass/SCSS (with dart-sass)
? Pick a linter / formatter config: Airbnb
? Pick additional lint features: Lint on save
? Where do you prefer placing config for Babel, ESLint, etc.? In dedicated
config files
? Save this as a preset for future projects? (y/N) n
```

圖 6-17　手動專案創建 - 是否要將目前的設定存爲樣板

選擇完成後，與預設專案創建相同，Vue CLI 將自動開始執行專案初始化程序。

使用了箭頭函式後，是否覺得程式更為簡潔呢？進入 Vue.js SPA 的世界後，讀者將會更常看見許多 ES6 或後續版本方便且簡潔的語法，例如：第 7 章將介紹的 async/await，便是 ES7 的語法。接著，就讓我們繼續探索 Vue.js SPA 的世界吧！

6-4 Vue.js 源碼程式

Vue CLI 建立的專案有基礎的設置，以預設範本為例，Vue.js 原始碼放在專案根目錄的 src 資料夾中，包含以下內容：

- assets 資料夾：放置需要打包的圖片或 CSS、SCSS 等檔案
- components 資料夾：放置 Vue.js 元件
- App.vue：Web 應用程式的根元件
- main.js：Web 應用程式的進入點

Vue.js 3.x 預設範本主程式

在專案中，原始碼裡的 main.js 為程式的進入點，Vue.js 3.x 預設樣板建出的程式碼如下：

JS JavaScript vue3/ch06/6-3/src/main.js

```
import { createApp } from 'vue';
import App from './App.vue';

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

SPA 架構可引用網上發佈的 JavaScript 套件，引用前須使用 npm 或 yarn 等 JavaScript 套件管理工具，將套件安裝至專案中。安裝完成後，再使用 ES6 的模組引入語法 - import，以套件名稱將套件引入。有關套件管理工具，可參照附錄 B，裡頭有詳細的使用說明。

main.js 主程式引用了 vue 套件裡的 createApp()方法來建立 Vue.js 實體，createApp()方法須帶入 Web 應用程式的根元件，並使用 Vue 實體的 mount()方法決定 HTML 裡渲染的位置，範例中綁定的位置為 id 值等於 app 的 HTML 標籤。

緊接著，來看看 main.js 主程式引用的 App 根元件內容吧！與 vue 套件相同，App 概元件也以 ES6 模組引入的語法引用 App.vue 的 SFC 元件檔，其程式碼內容如下：

Vue SFC vue3/ch06/6-3/src/App.vue

```
<template>
  <img alt="Vue logo" src="./assets/logo.png">
  <HelloWorld msg="Welcome to Your Vue.js App"/>
</template>

<script>
import HelloWorld from './components/HelloWorld.vue';

export default {
  name: 'App',
  components: {
    HelloWorld,
  },
};
</script>

<style lang="scss">
#app {
```

```
  font-family: Avenir, Helvetica, Arial, sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  text-align: center;
  color: #2c3e50;
  margin-top: 60px;
}
</style>
```

第 5 章介紹 SFC 元件檔案包含了撰寫樣板內容的<template>標籤、撰寫 Vue.js 程式的<script>標籤與撰寫元件使用樣式的<style>標籤，App.vue 也包含了以上三個標籤。在 SPA 架構中，SFC 程式包含的區域與第 5 章的相同，但是，由於 SPA 架構中引入了 ES6、SCSS 等語法特性，程式碼會有些許的不同。App.vue 的各區域說明如下：

- <template>標籤
  - 撰寫元件樣板的區域
  - 當元件有搭配的圖片資源時，可使用相對路徑引用圖片檔，例：App.vue 裡引用了「./assets/logo.png」圖檔
- <script>標籤
  - 撰寫 Vue.js 程式的區域
  - 使用 ES6 的 import 語法引用 SFC 元件檔、JavaScript 程式、JavaScript 套件及 CSS 檔
  - 使用 ES6 的 export default 語法導出元件
- <style>標籤
  - 撰寫元件需使用的樣式
  - 可以在<style>標籤加入 lang 屬性，決定要使用的預處理器，例：scss
  - 當<style>標籤加入 scope 時，產出的 css 只對元件本身有影響性

Vue.js 2.x 預設主程式

Vue.js 2.x 預設的主程式與 Vue.js 3.x 大致上相同，不同的地方在於，Vue.js 3.x 引用 vue 套件後，須使用 new 的方式建立 Vue 實體，且根元件須要使用 render 的方式帶入。以下為 Vue.js 2.x 預設主程式：

JS JavaScript vue2/ch06/6-3/src/main.js

```
import Vue from 'vue'
import App from './App.vue'

Vue.config.productionTip = false

new Vue({
  render: h => h(App),
}).$mount('#app')
```

Vue.js 2.x 的 SFC 元件與 Vue.js 3.x 大致相同，其中的差異在於 Vue.js 3.x 的<template>標籤中可以有多個根標籤，Vue.js 2.x 只能有 1 個根元件。SFC 的範例程式如下：

Vue SFC vue2/ch06/6-3/src/App.vue

```
<template>
  <div id="app">
    <img alt="Vue logo" src="./assets/logo.png">
    <HelloWorld msg="Welcome to Your Vue.js App"/>
  </div>
</template>

<script>
import HelloWorld from './components/HelloWorld.vue'

export default {
  name: 'App',
```

```
  components: {
    HelloWorld
  }
}
</script>

<style>
#app {
  font-family: Avenir, Helvetica, Arial, sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  text-align: center;
  color: #2c3e50;
  margin-top: 60px;
}
</style>
```

6-5 Vue.js 自定義插件（Plug-in）

註冊插件

Vue.js 豐富的生態體系中，具有各類的插件（Plug-in），例如：Vuex、Vue-Router、Vuetify…等。當我們要使用時，除了需要使用 import 引用外，也須使用 use()方法註冊。假設要將 myPlugin 插件註冊至 Vue.js 時，Vue.js 2.x 與 Vue.js 3.x 的註冊 Plug-in 語法如下：

JS JavaScript main.js

Vue.js 2.x	Vue.js 3.x
import Vue from 'vue'; import myPlugin from 'my-plugin'; Vue.use(myPlugin) ; var app = new Vue({ render: h => h(App) }).$mount('#app');	import { createApp } from 'vue'; import myPlugin from 'my-plugin'; createApp(App) .use(myPlugin) .mount('#app')

對照 Vue.js 2.x 與 Vue.js 3.x 的註冊方式後，將發現 2.x 版本使用 Vue 的 use()方法，3.x 版本須建立 Vue 的實體後才可使用 use()方法註冊。Vue 3.x 將 Vue 2.x 的全域概念移植到根元件上，當頁面中具有多個 Vue 實體，可以避免 Vue 實體間相互的影響。

自定義插件

Vue.js 提供了製作自定義插件的方式，這在 Web 大型應用程式的開發中使程式碼提高了再利用性。插件（Plug-in）的撰寫方式如下：

JS JavaScript

```
const myPlugin = {
  install (Vue) {
    [處理程式]
  }
}

export default myPlugin
```

插件內容為物件(Object)，該物件裡必須包含 install()方法。Install()方法帶入的 Vue 實體。當插件在主程式中，使用 Vue 的 use()方法註冊時，將會呼叫 install()方法，並以 Vue 實體作為 install()方法的第 1 個

參數執行。由於 Vue 實體的帶入，install()方法中可執行 Vue 本身提供的 API，例如：

- 註冊全域變數

  註冊全域變數時，須使用帶入 Vue 的 property 註冊。假設要註冊全域變數 - globalVar 時，語法如下：

JS JavaScript

```
const myPlugin = {
  install (Vue) {
    Vue.property.$globalVar = 'My Global Variable';
  }
};

export default myPlugin;
```

- 註冊全域元件

  註冊全域元件須先引入元件檔，並使用帶入 Vue 的 component 方法註冊。假設要註冊 my-component 元件，語法如下：

JS JavaScript

```
import MyComponent from './my-component.vue';

const myPlugin = {
  install (Vue) {
    Vue.component('my-component', MyComponent);
  }
};

export default myPlugin;
```

套件開發與使用

本例以範例 5-6 為基礎，將已建立的元件製作成套件，並改以 Vue.js SPA 的架構改寫。改寫 Vue.js SPA 的步驟如下：

Step 1 以 Vue CLI 建立預設 Vue.js 3 專案

執行指令「vue create 6-4」，並有樣板的地方選擇「Default ([Vue 3] babel, eslint)」選項建立預設樣板的 Vue 3 專案。執行成功畫面如下：

```
✨  Done in 31.92s.
🚀  Invoking generators...
📦  Installing additional dependencies...

yarn install v1.22.19
[1/4] 🔍  Resolving packages...
[2/4] 🚚  Fetching packages...
[3/4] 🔗  Linking dependencies...
[4/4] 🔨  Building fresh packages...

success Saved lockfile.
✨  Done in 11.67s.
⚓  Running completion hooks...

📄  Generating README.md...

🎉  Successfully created project 6-4.
👉  Get started with the following commands:

 $ cd 6-4
 $ yarn serve

➜  ch06 git:(master) ✗ ls -alh
```

圖 6-36　Vue CLI 建立專案完成畫面

專案建立完成後，專案資料夾結構如右：

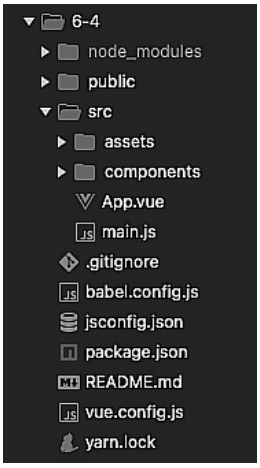

圖 6-37　Vue.js SPA 專案資料夾結構

Step 2 建立 my-plugin 插件

建立插件前，須在 src 資料夾建立 packages 資料夾，放置自己開發的套件。本例將建立 my-plugin 套件，故在 packages 資料夾建立後，再建立名為 my-plugin 的資料夾。此時，src 資料夾結構如右：

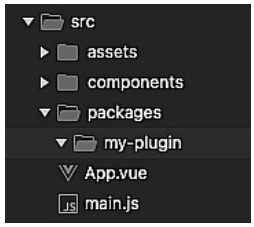

圖 6-38　建立插件用資料夾

建立完插件根目錄後，可建立 components 資料夾，將範例 5-6 建立過的 SFC 元件複製至 components 資料夾。接著，在插件根目錄 my-plugin 新增插件主程式－index.js，撰寫內容如下：

JS JavaScript　vue3/ch06/6-4/src/packages/my-plugin/index.js

```
import CheckBox from './components/check-box.vue';
import RadioBox from './components/radio-box.vue';
```

```
import SelectField from './components/select-field.vue';
import TextField from './components/text-field.vue';

const myPlugin = {
    install: function(Vue) {
        Vue.component('check-box', CheckBox);
        Vue.component('radio-box', RadioBox);
        Vue.component('select-field', SelectField);
        Vue.component('text-field', TextField);
    }
}

export default myPlugin;
```

插件主程式中，引入了 CheckBox 、 RadioBox 、 SelectField、TextField 等 4 個元件，並在 install()方法中，以 Vue.component()方法分別註冊元件。最後以 export default 匯出插件。完成後，src 資料夾結構如右：

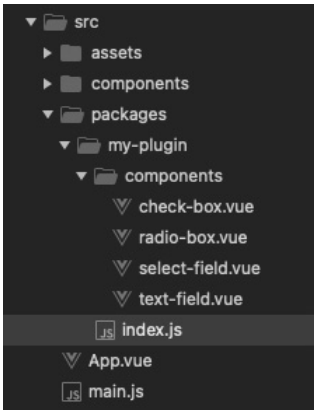

圖 6-39　插件完成資料夾結構

Step 3 註冊 my-plugin 插件

完成 my-plugin 插件後，須至 Vue.js 主程式 - main.js 中，以 import 引用 my-plugin 插件，並使用 use()方法註冊插件。修改後，主程式程式碼如下：

JS JavaScript　vue3/ch06/6-4/src/main.js

```
import { createApp } from 'vue';
import myPlugin from './packages/my-plugin';
```

```
import App from './App.vue'

createApp(App)
  .use(myPlugin)
  .mount('#app')
```

Step 4 修改根元件 App.vue 內容

完成 my-plugin 插件註冊後，在主程式以下載入的元件均可使用插件中註冊的元件。接著須建立根元件的內容，根元件也為 SFC 元件，故可分為 3 個部份：

- 元件樣板<template></template>

  將範例 5-6 index.html 的渲染區作為元件樣板的內容。

- 元件 Vue.js 程式<script></script>

  將範例 5-6 app.js 中，Vue.createApp()方法裡的物件，作為元件 Vue.js 程式以 export default 匯出的內容。

- 元件樣式<style></style>

  將範例 5-6 page.css 的內容，作為元件樣式區域的內容。

以上各區修改完成程式如下：

Vue SFC vue3/ch06/6-4/src/App.vue

```
<template>
    <div id="app">
        <h4>活動報名表單</h4>
        <form>
            <component
                v-for="(info, index) in fields"
                :key="index"
                :is="info.component_name"
                :column-id="info.id"
                :column-name="info.column_name"
                :items="info.items"
```

```
                :placeholder="info.placeholder"
                v-model="form[info.id]"
            ></component>
            <button type="button" class="btn btn-primary"
               @click="send">送出</button>
        </form>
        <div class="form-info" v-if="show">
            送出表單資訊：
            <ul>
                <li>姓名：{{ show.fullName }}</li>
                <li>性別：{{ show.gender }}</li>
                <li>聯絡電話：{{ show.tel }}</li>
                <li>想參加的活動：{{ show.willingness.
                   join(',') }}</li>
                <li>交通方式：{{ show.transportation }}</li>
            </ul>
        </div>
    </div>
</template>

<script>
export default {
    name: 'App',
    data: () => ({
        // 表單資訊
        form: {
            fullName: '',
            gender: '',
            tel: '',
            willingness: [],
            transportation: '',
        },
        // 顯示資訊
        show: false,
        // 性別選單項目
        gender: [
            { text: '男', value: '男', },
            { text: '女', value: '女', },
```

```
        ],
        // 活動選單項目
        activities: [
            { text: '唱歌', value: '唱歌' },
            { text: '烤肉', value: '烤肉' },
            { text: '桌遊', value: '桌遊' },
            { text: '看展', value: '看展' },
        ],
        // 交通方式選單項目
        transportations: [
            { text: '搭遊覽車', value: '搭遊覽車' },
            { text: '自行騎車', value: '自行騎車' },
            { text: '自行開車', value: '自行開車' },
        ],
    }),
    computed: {
        fields: function() {
            return [
                {
                    component_name: 'text-field',
                    id: 'fullName',
                    column_name: '姓名',
                    placeholder: '請輸入您的姓名'
                },
                {
                    component_name: 'radio-box',
                    id: 'gender',
                    column_name: '性別',
                    items: this.gender
                },
                {
                    component_name: 'text-field',
                    id: 'tel',
                    column_name: '聯絡電話',
                    placeholder: '電話格式：(xx)xxxx-xxxx'
                },
                {
                    component_name: 'check-box',
```

```
                    id: 'willingness',
                    column_name: '想參加的活動',
                    items: this.activities
                },
                {
                    component_name: 'select-field',
                    id: 'transportation',
                    column_name: '交通方式',
                    items: this.transportations
                },
            ];
        },
    },
    methods: {
        // 送出
        send() {
            this.show = {
                fullName: this.form.fullName,
                gender: this.form.gender,
                tel: this.form.tel,
                willingness: this.form.willingness,
                transportation: this.form.transportation,
            };
        },
    },
}
</script>

<style>
#app {
    padding: 1rem;
}

.form-info {
    padding-top: 3rem;
    font-size: 1.25rem;
}
</style>
```

Step 5 修改 public 下的 index.html 頁面檔，將需用的外部 JavsScript 或 CSS 載入。最後，由於範例 5-6 使用了 Bootstrap、jQuery 等外部套件，故須在 public 資料夾裡的 index.html 網頁檔中，引入套件的 JavaScript 或 CSS 檔。修改完程式如下：

HTML5 vue3/ch06/6-4/public/index.html

```
<!DOCTYPE html>
<html lang="">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width,
       initial-scale=1.0">
    <link rel="icon" href="<%= BASE_URL %>favicon.ico">
    <title><%= htmlWebpackPlugin.options.title %></title>
    <link rel="stylesheet" href="https://stackpath.
       bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap. min
       .css" integrity="sha384-MCw98/SFnGE8fJT3GxwEOngsV7Z
       t27NXFoaoApmYm81iuXoPkFOJwJ8ERdknLPMO" crossorigin=
       "anonymous">
  </head>
  <body>
    <noscript>
      <strong>We're sorry but <%= htmlWebpackPlugin.
         options.title %> doesn't work properly without
         JavaScript enabled. Please enable it to continue.
         </strong>
    </noscript>
    <div id="app"></div>
    <!-- built files will be auto injected -->
    <script src="https://code.jquery.com/jquery-3.3.1.
       slim.min.js" integrity="sha384-q8i/X+965DzO0rT7abK
       41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo"
       crossorigin="anonymous"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/
       popper.js/1.14.3/umd/popper.min.js" integrity=
       "sha384-ZMP7rVo3mIykV+2+9J3UJ46jBk0WLaUAdn689aCwoq
```

```
        bBJiSnjAK/l8WvCWPIPm49" crossorigin="anonymous">
        </script>
    <script src="https://stackpath.bootstrapcdn.com/
        bootstrap/4.1.3/js/bootstrap.min.js" integrity=
        "sha384-ChfqqxuZUCnJSK3+MXmPNIyE6ZbWh2IMqE241rYi
        qJxyMiZ6OW/JmZQ5stwEULTy" crossorigin="anonymous">
        </script>
  </body>
</html>
```

Step 6 打包程式

程式修改完成後，便可以執行「yarn build」指令打包程式。打包完成畫面如下：

```
 DONE  Compiled successfully in 12962ms
下午4:32:01

  File                                 Size                    Gzipped

  dist/js/chunk-vendors.5fd78b17.js    77.93 KiB               28.94 KiB
  dist/js/app.6c638701.js              7.34 KiB                2.55 KiB
  dist/css/app.0516155d.css            0.06 KiB                0.07 KiB

  Images and other types of assets omitted.
  Build at: 2022-09-02T08:32:01.949Z - Hash: de8c5625d011a8f2 - Time: 12962ms

 DONE  Build complete. The dist directory is ready to be deployed.
 INFO  Check out deployment instructions at https://cli.vuejs.org/guide/deployment.html

✨  Done in 16.68s.
➜  6-4 git:(master) ✗
```

圖 6-40　範例打包完成畫面

程式打包完成後，將會自動建立 dist 資料夾，資料夾結構如右：

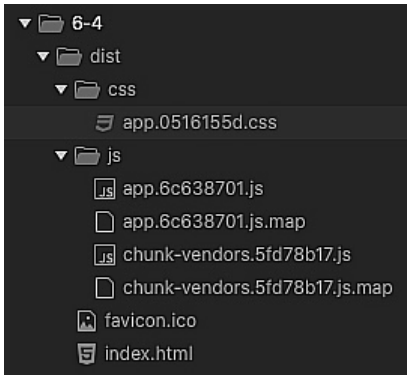

圖 6-41　打包完成 dist 資料夾結構

打包完的 index.html 由於自動產生的 js 或 css 路徑為絕對路徑，如果直接點開可能會打不開。若想開啟，須手動將 index.html 中引用打包完成的 js 或 css 改為相對路徑後，才可正常打開執行。修改範例如下：

HTML5 vue3/ch06/6-4/public/index.html

```
<!- (略) -->
    <script defer="defer"
src="./js/chunk-vendors.5fd78b17.js"></script>
    <script defer="defer"
src="./js/app.6c638701.js"></script>
    <link href="/css/app.0516155d.css" rel="stylesheet">
<!- (略) -->
```

修改完成後，執行畫面如下：

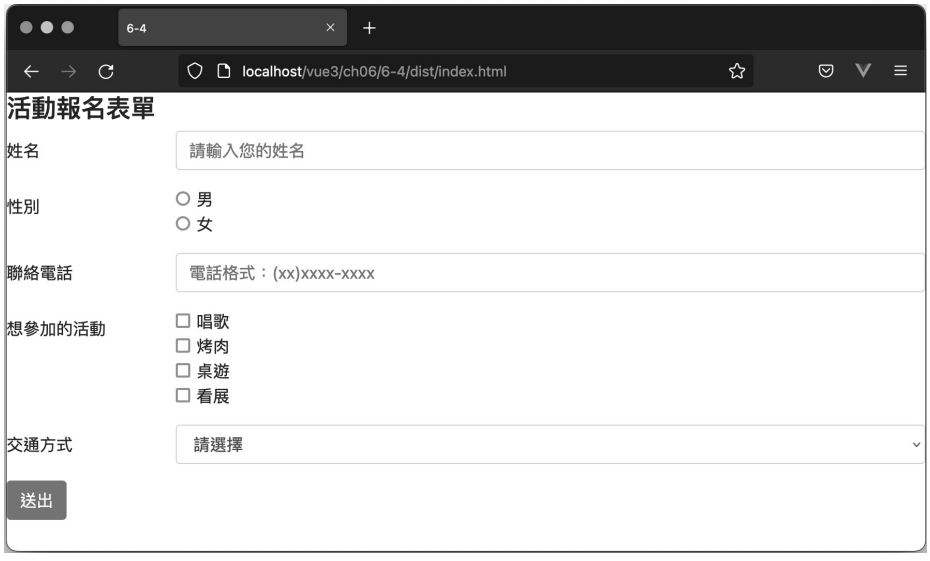

圖 6-42　範例執行完成圖