目 錄

[1 Module 1. Vue.js 課程開發工具與課程安排說明 3](#_Toc47514540)

[1.1 Vue.js開發環境、Vue.js devtools偵錯工具 安裝 3](#_Toc47514541)

[1.2 網頁設計基礎複習 4](#_Toc47514542)

[1.3 課堂實作與練習目標設置 4](#_Toc47514543)

[2 Module 2. Vue.js 簡介與使用 5](#_Toc47514544)

[2.1 官方文件導讀 5](#_Toc47514545)

[2.2 認識基本設置，說明引用順序、作用範圍與規則 5](#_Toc47514546)

[2.3 建置第一個套用 Vue.js 框架的網頁 6](#_Toc47514547)

[3 Module 3. Vue.js 結構與運作原理 8](#_Toc47514548)

[3.1 MVVM 8](#_Toc47514549)

[3.2 object options 8](#_Toc47514550)

[3.3 生命週期 10](#_Toc47514551)

[4 Module 4. Vue directive (v-指令) 13](#_Toc47514552)

[4.1 v-html、v-once、v-pre 13](#_Toc47514553)

[4.2 條件控制：v-if、v-show比較 16](#_Toc47514554)

[4.3 動態產生多筆資料：v-for 19](#_Toc47514555)

[5 Module 5. 資料流和樣式綁定 23](#_Toc47514556)

[5.1 顯示資料 23](#_Toc47514557)

[5.2 單向/雙向綁定 23](#_Toc47514558)

[5.3 class 和 style 綁定 27](#_Toc47514559)

[6 Module 6. 網頁操作行為互動和自定義指令 33](#_Toc47514560)

[6.1 事件處理和繫結 33](#_Toc47514561)

[6.2 事件修飾符：Event-Modifiers 35](#_Toc47514562)

[6.3 Vue.directive(id, definition) 37](#_Toc47514563)

[7 Module 7. JavaScript ES6 - 宣告和常用方法 39](#_Toc47514564)

[7.1 變數宣告的方式與作用域 39](#_Toc47514565)

[7.2 陣列常用的方法 40](#_Toc47514566)

[7.3 匯入和輸出 42](#_Toc47514567)

[8 Module 8. JavaScript ES6 - Function 44](#_Toc47514568)

[8.1 Arrow function 44](#_Toc47514569)

[8.2 Function chaining 45](#_Toc47514570)

[8.3 撰寫方式比較 45](#_Toc47514571)

[9 Module 9. Vue 元件 - 樣板功能 47](#_Toc47514572)

[9.1 註冊全域/區域樣板 47](#_Toc47514573)

[9.2 return object 建構資料格式 51](#_Toc47514574)

[9.3 Props 觀念和型別預設值 51](#_Toc47514575)

[10 Module 10. 動態設定樣板內容 54](#_Toc47514576)

[10.1 不具名 slot 54](#_Toc47514577)

[10.2 具名slot (Named Slot) 55](#_Toc47514578)

[10.3 scoped-slot 57](#_Toc47514579)

[11 Module 11. Vue 元件 - 資料和程式碼 59](#_Toc47514580)

[11.1 重複使用程式碼 59](#_Toc47514581)

[11.2 保留表單狀態 60](#_Toc47514582)

[11.3 多檔案連結 62](#_Toc47514583)

[12 Module 12. 事件傳遞 64](#_Toc47514584)

[12.1 watch & computed 比較 64](#_Toc47514585)

[12.2 $emit 67](#_Toc47514586)

[12.3 event bus 69](#_Toc47514587)

[13 Module 13. Vue transition mode 72](#_Toc47514588)

[13.1 對應的指令說明 72](#_Toc47514589)

[13.2 自行設計CSS效果或使用第三方套件 79](#_Toc47514590)

[13.3 透過JavaScript操作 80](#_Toc47514591)

[14 Module 14. Vue Router 83](#_Toc47514592)

[14.1 配置流程 83](#_Toc47514593)

[14.2 動態路由 89](#_Toc47514594)

[14.3 Navigation Guards 91](#_Toc47514595)

[15 Module 15. Vuex 97](#_Toc47514596)

[15.1 基本結構 97](#_Toc47514597)

[15.2 狀態管理模式 98](#_Toc47514598)

[15.3 跨元件間的溝通 100](#_Toc47514599)

[16 Module 16. Vue.Cli v3 建立SPA專案 108](#_Toc47514600)

[16.1 透過 npm 安裝 @vue/cli，然後以vue create指令建立專案（文字模式） 108](#_Toc47514601)

[16.2 透過 vue ui指令啟動「Vue專案管理器」，然後以GUI介面建立專案（網頁模式） 109](#_Toc47514602)

[16.3 巡覽建立後資料夾內容並測試 109](#_Toc47514603)

[16.4 使用Visual Studio快速建立SPA專案 112](#_Toc47514604)

[17 Module 17. 導引實作練習：主題網站 - 元件設立 115](#_Toc47514605)

[17.1 導覽列設置 115](#_Toc47514606)

[17.2 商品展示 117](#_Toc47514607)

[17.3 條件篩選 119](#_Toc47514608)

[18 Module 18. 導引實作練習：主題網站 - 資料傳遞連動 123](#_Toc47514609)

[18.1 加入購物車(SPA) 123](#_Toc47514610)

[18.2 結帳頁面(SPA) 133](#_Toc47514611)

[18.3 表單驗證 134](#_Toc47514612)

[19 附錄 138](#_Toc47514613)

[19.1 過濾器(Filters) 138](#_Toc47514614)

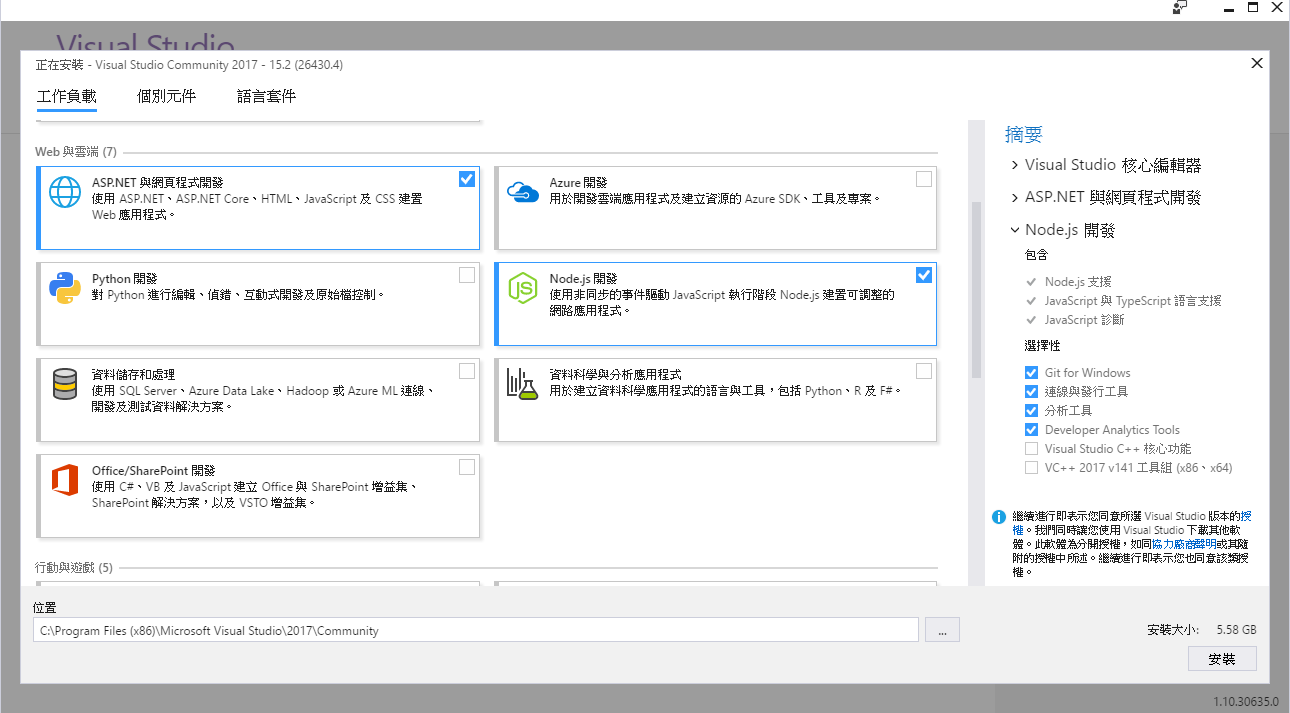
[19.2 使用v-clock指令 139](#_Toc47514615)

# Module 1. Vue.js 課程開發工具與課程安排說明

## Vue.js開發環境、Vue.js devtools偵錯工具 安裝

### Vue.js開發環境

* Visual Studio 2019是Vue.js應用程式的一個常用的IDE(Integrated Development Environment) 開發工具。
* Visual Studio 2019安裝步驟：
  + 下載、安裝Node.js runtime
    - 下載網址: https://nodejs.org/en/download/ 或https://nodejs.org/zh-tw/download/
    - 下載檔案: node-v12.#.#-x64.msi (Windows Installer / LTS (Long Term Support) version / 64-bit) 並執行之(預設安裝路徑：C:\Program Files\nodejs)
  + 下載、安裝Visual Studio (Community) 2019
    - 安裝過程中: 選取「Node.js開發」工作負載(workload)



### Vue.js devtools安裝與啟用

* Vue.js devtools 是一個支援Vue.js應用程式偵錯的瀏覽器(Chrome、Firefox)的擴充功能
* 在Google Chrome安裝Vue.js devtools的步驟
  + 開啟Google Chrome瀏覽器
  + 在網址列輸入下列網址，以連結「chrome 線上應用程式商店」之Vue.js devtools擴充功能：

https://chrome.google.com/webstore/detail/vuejs-devtools/nhdogjmejiglipccpnnnanhbledajbpd

* + 在Vue.js devtools擴充功能畫面上點擊「加到Chrome」按鈕
* 設定Vue.js devtools支援網址為file://開頭的網頁
  + 開啟Chrome擴充功能畫面
    - Chrome > 自訂及管理Google Chrome > 更多工具 > 擴充功能
  + 在Vue.js devtools項目上點選「詳細資訊」按鈕
  + 啟用「允許存取檔案網址」



## 網頁設計基礎複習

參考「建置現代化的網頁 – 使用Vue.js 附錄-網頁設計基礎.docx」

## 課堂實作與練習目標設置

* 短期：了解Vue.js架構和運行概念，並能理解v-指令使用時機
* 中期：透過改寫先前於網頁相關科目的作業/專題，熟悉元件間資料的傳遞方式以及事件呼叫
* 長期：能配合專案需求挑選適合方法實作Vue.js技術

# Module 2. Vue.js 簡介與使用

## 官方文件導讀

* 參考網址：<https://vuejs.org/v2/guide/>
* 請特別注意：Vue的生命週期 (參考3.3.1)。

## 認識基本設置，說明引用順序、作用範圍與規則

### Vue.js簡介

* Vue (發音如view) 是一套用於建構用戶界面(前端)的JavaScript框架(framework)，支援透過資料異動而更新頁面。
* 與其他大型框架(monolithic frameworks)不同的是，Vue 被設計為可以自底向上逐層應用。Vue的核心函式庫只關注MVC模式中的視圖層(view layer)，不僅易於上手，還便於與第三方函式庫或既有專案整合。
* 另一方面，當Vue與其他較現代化的工具以及各種支持性的函式庫結合使用時，也能够為複雜的單網頁應用程式(Single-Page Applications/SPA)提供其強大的能力。
  + A single-page application (SPA) is a web application or website that interacts with the web browser by dynamically rewriting the current web page with new data from the web server, instead of the default method of the browser loading entire new pages. (Corelab 17-2)
  + The goal is faster transitions that make the website feel more like a native app.
  + In a SPA, all necessary HTML, JavaScript, and CSS code is either retrieved by the browser with a single page load, or the appropriate resources are dynamically loaded and added to the page as necessary, usually in response to user actions.
* 使用Vue.js的知名公司有阿里巴巴、百度、GitLab等

### 引用Vue.js函式庫

* 建立.html文件
* 在.html文件之<head>元素內使用<script src="">元素引用Vue.js函式庫。

例如：

<!-- 開發環境版本(development version)，包含了有帮助的主控台警告訊息-->

<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>

或

<script src="javascripts/vue.js"></script>

<!-- 生產環境版本(production version)，優化了檔案大小和速度 -->

<script src="https://cdn.jsdelivr.net/npm/vue"></script>

或

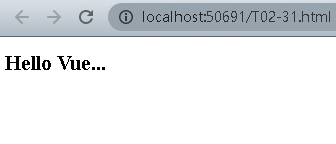
<script src="javascripts/vue.min.js"></script>

### Vue.js應用程式之運作核心

* 在.html文件之<body>元素內建立「模板」(Template)
* 在.html文件內且在Vue.js函式庫引用之後，使用<script>元素在其內建立Vue實例
* Vue實例之資料與模板之DOM進行雙向資料流

## 建置第一個套用 Vue.js 框架的網頁

### 範例：T02-31.html



<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>我的第一個Vue網頁</title>

<!-- 開發環境版本(development version)，包含了有帮助的主控台警告訊息-->

<!--<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>-->

<script src="javascripts/vue 2.6.11/vue.js "></script>

<!-- 生產環境版本(production version)，優化了檔案大小和速度-->

<!--<script src="https://cdn.jsdelivr.net/npm/vue"></script>-->

<!--<script src="javascripts/vue 2.6.11/vue.min.js"></script>-->

</head>

</head>

<body>

<div id="app">

<h1> **{{**message**}}** </h1>

</div>

<script>

var vm = new Vue({

el: "#app",

//Provide the Vue instance with an existing DOM element. It can be a CSS selector

string, an actual HTMLElement, or a function that returns an HTMLElement.

//Note that the provided element merely serves as a mounting point; it will be replaced

if a template is also provided.

data: { message: "Hello Vue..." }

});

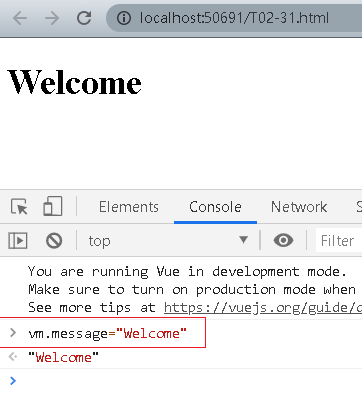
</script>

</body>

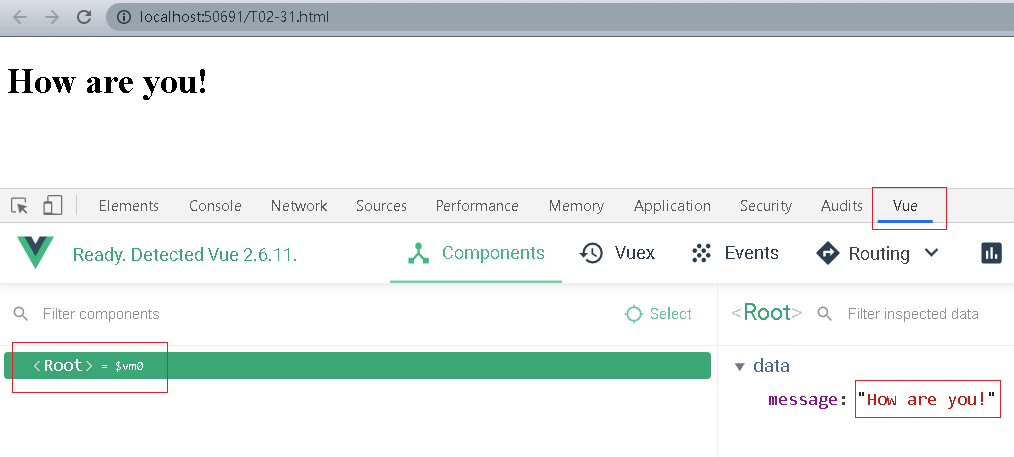
</html>

### 範例特別說明

* Vue.js的核心是一個允許採用簡潔的模板(template)語法來聲明式地將資料渲染(render)到DOM上(declaratively render data to the DOM)的系統。
* 上面範例之資料和DOM已經被建立了關聯，它具有響應式(reactive)的特性。例如：
  + - 開啟瀏覽器主控台，輸入 vm.message = "Welcome"; 即可產生如下內容的頁面。



* + - 開啟devtools工具，message輸入 "How are you!"; 即可產生如下內容的頁面。



# Module 3. Vue.js 結構與運作原理

## MVVM

* Vue在網頁技術的MVC架構中扮演View(視圖)與Model(模型)之間的中介互動角色，稱為ViewModel 層，是View與Model之間的橋樑，這種「軟體架構模式」就稱為MVVM(Model–View–Viewmodel)。



參考資料來源：https://zh.wikipedia.org/wiki/MVVM

* Vue 透過Viewmodel中介層進行頁面(View)與資料(Model)更新，達到雙向資料綁定功能。

## object options

### 建立Vue實例

* 每個Vue應用程式都是透過建立一個新的Vue實例開始的：

var vm = new Vue({

// options

})

* 當建立一個Vue實例時，你可以傳入一個選項物件(options object)。
* 一個Vue應用程式由一個利用 new Vue所建立的根Vue實例(root Vue instance)，以及選擇性的巢狀的、可重複利用的元件樹(a tree of components)所組成。



參考資料來源：<https://vuejs.org/v2/guide/components.html#Organizing-Components>

* + 例如，你可能會有頁頭、側邊欄、内容區等元件，每個元件又包含了其它的元件，如導航鏈結、博文(blog posts)之類的元件。
  + 所有的Vue元件都是Vue實例。

### Vue實例的資料(Data)與方法(Method)

* 當一個 Vue實例被建立時，它將資料物件(data object)中的所有的屬性(properties)加入到Vue的響應式系統(reactivity system)中。當這些屬性的值發生改變時，視圖將會產生「響應(react)」，即更新為對應的新的值。
* 除了資料屬性(data properties)，Vue 實例還內建了一些有用的實體屬性與方法。它們都有前綴 $，以便與使用者定義的屬性區分開來。例如：

vm.$data 代表The data object that the Vue instance is observing.

### 模板語法(Template Syntax)

* Vue.js使用了基於HTML的模板(HTML-based template)語法，允許開發者以聲明方式(declaratively)將DOM綁定至Vue實例的底層的資料。
* 所有Vue.js的模板都是合法的HTML，所以能被遵循規格的瀏覽器和 HTML解析器所解析。
* 在底層的實現上，Vue將模板編譯成特別的函數。結合響應系統(reactivity system)，當應用程式的狀態發生變更時，Vue能够智能地計算出最少需要重新渲染多少元件，並把DOM操作次數减到最少。

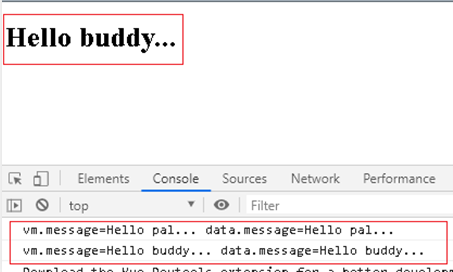
### 文字插值(Text Interpolation)

* 資料綁定(data binding)最常見的形式就是使用「小鬍子(Mustache)語法」 (雙大括號)的文字插值：

<h1>Message: {{ message }}</h1>

* Mustache標籤將會被替代為對應資料物件上message屬性的值。無論何時，綁定的資料物件的message屬性發生了改變，插值處的内容都會更新。

#### 範例：T03-21.html



<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<title>Vue的資料屬性與實體屬性</title>

<script src="javascripts/vue 2.6.11/vue.js "></script>

</head>

<body>

<div id="app">

<h1> {{message}} </h1>

</div>

<script>

var vm = new Vue({

//el: "#app",

//If the option is available at instantiation, the instance will immediately enter compilation;

//otherwise, the user will have to explicitly call vm.$mount() to manually start the compilation.

data: { message: "Hello world..." }

});

vm.$mount("#app");

//If the Vue instance didn’t get an el option at instantiation, you can manually call

$mount(el) to start the compilation phase.

//By default, the mounted element will be replaced by the instance’s template.

vm.message = "Hello pal...";

</script>

</body>

</html>

## 生命週期

* 每個Vue實例在被建立時都要經過一系列的初始化過程，例如，需要資料監聽(data observation)、編譯模板(compile the template)、將實例掛載到DOM(mount the instance to the DOM) 並在資料發生變化時更新DOM等。
* 同時，在這個過程中也會執行一些叫做生命週期鈎子(lifecycle hooks)的函數，這給了使用者在不同階段添加自己的程式碼的機會。例如：
  + created鈎子可以用來在一個實例被建立之後執行程式碼(可以開始使用Vue實例內的資料物件(data object)內的資料)。
  + mounted鈎子可以用來開始使用vm.$el物件
    - vm.$el ：Vue實例使用的根DOM元素(root DOM element)。
    - 在此階段所有的DOM元素已完成渲染。
* 還有一些其它的鈎子在實例生命週期的不同階段被呼叫，如updated 和destroyed。



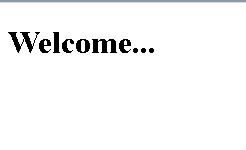
*vm.$destroy()：*

* *In normal use cases you shouldn’t have to call this method yourself.*
* *Prefer controlling the lifecycle of child components in a data-driven fashion using v-if and v-for.*

Lifecycle Diagram

參考資料來源：<https://vuejs.org/v2/guide/instance.html#Lifecycle-Diagram>

### 範例：T03-31.html



<!DOCTYPE html>

<html>

<head>

(略)

</head>

<body>

<div id="app">

<h1> **{{**message**}}** </h1>

</div>

<script>

var vm = new Vue({

el: "#app",

data: { message: "歡迎光臨…" },

created: function () {

//this指向vm實例

this.message = "Welcome…";

}

});

</script>

</body>

</html>

# Module 4. Vue directive (v-指令)

* 指令是帶有v-前綴(prefix)的特殊屬性(attributes)。
* 指令屬性的值預期是單一JavaScript運算式(v-for是例外情况)。
* 指令的功能是當運算式的值改變時，將其產生的連帶影響，響應式地作用於DOM。
* 例如 <p v-if="seen">Now you see me</p>

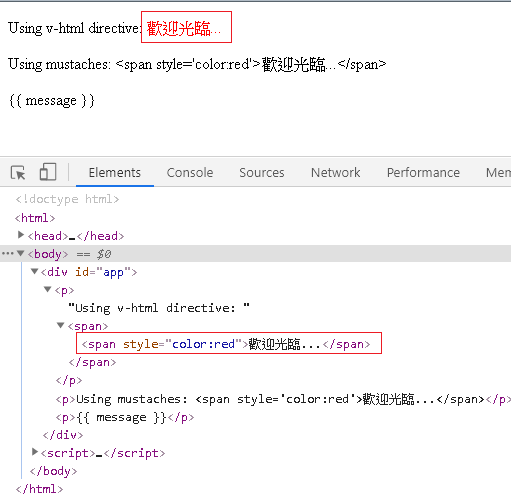
這裡v-if指令將根據運算式seen的值的真/假來插入/移除<p>元素。

## v-html、v-once、v-pre

### 原始HTML插值(v-html指令)

* 雙大括號會將資料解釋為普通文字，而非HTML程式碼。為了輸出真正的HTML，你需要使用v-html指令以進行原始HTML(Raw HTML)插值。

#### 範例：T04-11.html (I)



<!DOCTYPE html>

<html>

(略)

<body>

<div id="app">

<p>

Using v-html directive:

<span v-html="message"></span>

<!--span的内容將會被替換成rawHtml屬性的值，直接作為HTML。-->

</p>

(略)

</div>

<script>

var vm = new Vue({

el: "#app",

data: {

message: "<span style='color:red'>歡迎光臨...</span>"

}

});

</script>

</body>

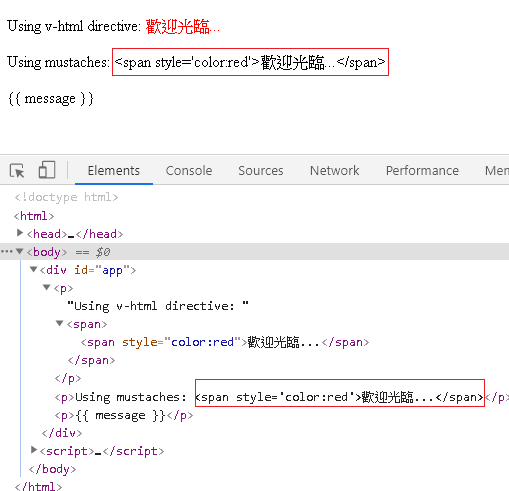
</html>

### 一次性文字插值(v-once指令)

* 基本上Mustache標籤會被替代為對應資料物件上的屬性的值。無論何時，綁定的資料物件的屬性發生了改變，插值處的内容都會更新。
* 透過使用v-once指令，你也能執行一次性地插值，當資料改變時，插值處的内容不會更新：

<p v-once>這個將不會改變: {{ message }}</p>

#### 範例：T04-11.html (II)



<!DOCTYPE html>

<html>

(略)

<body>

<div id="app">

(略)

<p v-once>Using mustaches: **{{** message **}}**</p>

(略)

</div>

<script>

var vm = new Vue({

el: "#app",

data: {

message: "<span style='color:red'>歡迎光臨...</span>"

}

});

</script>

</body>

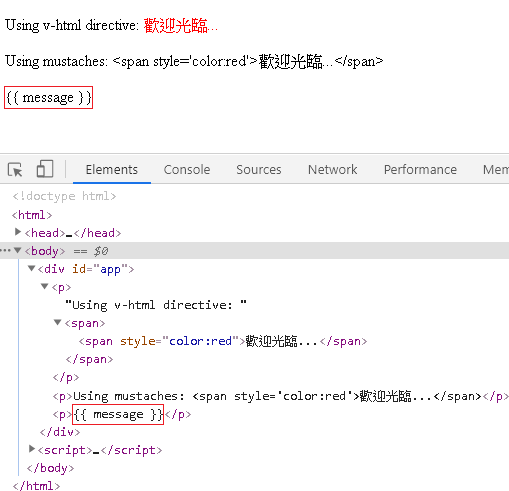
</html>

### 跳過插值(v-pre指令)

* v-pre指令會跳過(skip)所屬元素和它的子元素的編譯過程。
* 可以用來顯示原始Mustache標籤。
* 例如：

<p v-pre>{{ message }}</p> (渲染如下圖)

#### 範例：T04-11.html (III)



<!DOCTYPE html>

<html>

(略)

<body>

<div id="app">

(略)

<p v-pre>**{{** message **}}**</p>

</div>

<script>

var vm = new Vue({

el: "#app",

data: {

message: "<span style='color:red'>歡迎光臨...</span>"

}

});

</script>

</body>

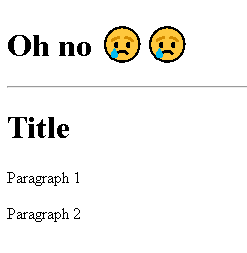
</html>

## 條件控制：v-if、v-show比較

### v-if/v-else

* v-if指令用於條件性地渲染一個區塊内容。這個區塊内容只會在指令的運算式傳回真值的時候被渲染。
* 你可以選擇性地使用v-else指令來表示v-if 的“else區塊”。
* v-else元素必須緊跟在v-if 或者v-else-if 的元素的後面，否則它將不會被識別。
* 又因為v-if是一個指令，所以必須將它添加到一個元素上。但是如果想切換多個元素呢? 此時可以把一個<template>元素當作不可見的包裹元素(invisible wrapper)，並在上面使用v-if。最終的渲染結果將不包含 <template> 元素。

#### 範例：T04-21.html



<!DOCTYPE html>

<html>

(略)

<body>

<div id="app">

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

<h1 v-else>Oh no 😢&#x1F622;</h1><!--https://dict.emojiall.com/zh-hant -->

<hr/>

<template v-if="ok">

<h1>Title</h1>

<p>Paragraph 1</p>

<p>Paragraph 2</p>

</template>

</div>

<script>

var vm = new Vue({

el: '#app',

data: {

awesome: false,

ok: true

}

})

</script>

</body>

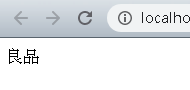
</html>

</html>

### v-else-if

* v-else-if，顧明思義，充當 v-if 的“else-if區塊”，可以連續使用。
* 類似於 v-else，v-else-if 也必須紧跟在帶 v-if 或者 v-else-if 的元素之後。

#### 範例：T04-22.html



<!DOCTYPE html>

<html>

(略)

<body>

<div id="app">

<div v-if="rank=='A'">極品</div>

<div v-else-if="rank=='B'">高級品</div>

<div v-else-if="rank=='C'">良品</div>

<div v-else="rank=='D'">次級品</div>

</div>

<script>

var vm = new Vue({

el: '#app',

data: {

rank: 'C'

}

})

</script>

</body>

</html>

### v-show

* 另一個用於根據條件展示元素的選項是v-show指令。用法大致與v-if指令一樣：

<h3 v-show="ok">歡迎光臨</h3>

* 不同的是：
  + 帶有v-show 的元素始終會被渲染並保留在DOM中。v-show只是簡單地切換元素的CSS屬性display以顯示或隱藏該元素。(注意，v-show不支持<template>元素，也不支持v-else。)
  + v-if 是“真正”的條件渲染(conditional rendering)，因為它會確保在切換過程中條件區塊內的事件監聽器和子元件(child components)適當地被銷毀和重新建立。
  + v-if 也是惰性的(lazy)：如果在初始渲染時條件為假，則什麼也不做——直到條件第一次變為真時，才會開始渲染條件區塊。
  + 相比之下，v-show 就簡單得多——不管初始條件是什麼，元素總是會被渲染，並且只是簡單地基於CSS進行切換。
* 一般來說，v-if有較高的切換成本，而v-show有較高的初始渲染成本。因此，如果需要非常頻繁地切換，則使用 v-show 較好；如果在執行時條件很少改變，則使用 v-if 較好。

#### 範例：T04-23.html



<!DOCTYPE html>

<html>

(略)

<body>

<div id="app">

<div style="height:60px; width:200px; text-align:center;

margin-top: 20px; border:1px solid #444">

<h3 v-show="ok">歡迎光臨</h3>

</div>

<input type="button" value="切換顯示/隱藏" onclick="vm.btnClick();" />

</div>

<script>

var vm = new Vue({

el: '#app',

data: {

ok: true

},

methods: {

btnClick: function () {

this.ok = !this.ok;

}

}

})

</script>

</body>

</html>

## 動態產生多筆資料：v-for

### 以v-for遍歷陣列元素

* 我們可以用v-for指令來渲染一個一個陣列的項目列表。
* v-for指令需要使用「item in/of items」形式的特殊語法，其中items是來源資料陣列，而item則是被迭代的陣列元素的别名。
* 在v-for區塊中，我們可以存取所有父範圍的屬性(parent scope properties)。
* v-for 還支持一個選擇性的第二個引數，即當前項目的索引：「item[,index] in/of items」
* 狀態維護(Maintaining State)
  + 當Vue更新使用v-for所渲染的元素列表(a list of elements)時，它預設使用“就對應位置更新”的策略。如果資料項的順序被改變，Vue將不會移動DOM元素匹配資料項的順序，而是就對應位置更新每個元素，並且確保它們在每個索引位置正確渲染。
  + 這個預設的模式是高效的，但是只適用於元素列表渲染輸出時不依賴子元件的狀態的情況(例如：表單輸入值)。
  + 為了給Vue一個提示，以便它能跟踪每個節點的身份，從而重複使用和重新排序現有元素，你需要為每項節點提供一個唯一key屬性(attribute)，例如：

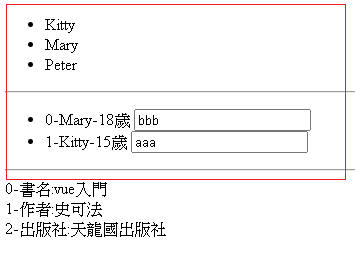
<div v-for="item in items" v-bind:key="item.id">

<!-- 内容 -->

</div>

* + - v-bind:key= "…"可以簡寫(shorthand)成**:**key= "…"。(有人將shorthand解釋為一種語法糖/syntax sugar)
    - v-bind是一種單向的資料綁定(data 🡪 DOM)。
* 建議儘可能在使用v-for時提供key屬性(Vue會透過Key值判斷元素是否需要重新渲染，例如元素的key值變更時，Vue會重新渲染該元素) (Corelab7-2)，除非遍歷輸出的DOM内容非常簡單，或者是刻意依賴預設行為以獲取性能上的提升。

#### 範例：T04-31.html (I)



<!DOCTYPE html>

<html>

(略)

<body>

<div id="app">

<ul>

<li v-for="item in items">

**{{**item**}}**

</li>

</ul>

<hr />

<ul>

<li v-for="(student,index) in students" v-bind:key="student.id">

**{{**index**}}**-**{{**student.name**}}**-**{{**student.age**}}**歲

<input type="text" size="20" />

</li>

</ul>

<hr />

(略)

</div>

<script>

var vm = new Vue({

el: '#app',

data: {

items: ["Kitty","Mary","Peter"],

students: [{ name: 'Kitty', age: 15, id: 1},

{ name: 'Mary', age: 18, id: 2}],

(略)

}

});

//測試: v-for with/without v-bind:key 的效果

$("#app input:eq(0)").val("aaa");

$("#app input:eq(1)").val("bbb");

setTimeout(function () {

vm.students.reverse(); //變更陣列資料(students)的元素順序時，若對應DOM元素

有綁定key，則Vue將據以移動DOM元素的相關位置

}, 2000);

</script>

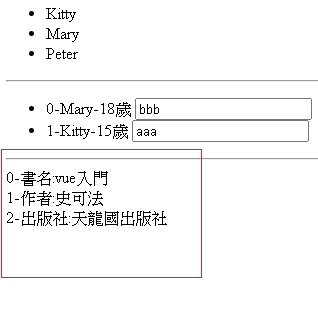
</body>

</html>

### 以v-for遍歷物件屬性

* 你也可以用 v-for來遍歷一個物件的屬性(properties)。
* v-for指令需要使用「value in/of object」形式的特殊語法，其中object 是來源資料物件，而value則是被迭代的物件屬性的值。
* 你也可以提供選擇性的第二個引數，即物件的屬性名稱(也就是鍵名)，以及選擇性的第三個引數，即當前屬性的索引：「value[,name[,index]] in/of object」

#### 範例：T04-31.html (II)



<!DOCTYPE html>

<html>

(略)

<body>

<div id="app">

(略)

<div v-for="(value,name,index) in book">

**{{**index**}}**-**{{**name**}}**:**{{**value**}}**

</div>

</div>

<script>

var vm = new Vue({

el: '#app',

data: {

(略)

book: {書名:"vue入門",作者:"史可法",出版社:"天龍國出版社"}

}

})

</script>

</body>

</html>

# Module 5. 資料流和樣式綁定

## 顯示資料

* 前面提到Vue.js的核心是一個允許採用簡潔的模板語法來聲明式地將資料渲染到DOM上(declaratively render data to the DOM)的系統。
* 資料綁定(data binding)最常見的形式就是文字插值。例如：

<h1>Message: {{ message }}</h1>

* + Mustache標籤將會被替代為對應資料物件上message屬性的值。
  + 無論何時，綁定的資料物件的message屬性發生了改變，插值處的内容都會更新。

## 單向/雙向綁定

* 到目前為止，我們知道Vue.js的響應式系統(reactivity system)會在資料發生變更時跟著更新模板，這是單向資料綁定(One-Way Data Binding)。
* 你可以用v-model指令在表單<input>、<textarea>及<select>元素上建立雙向資料綁定(Two-Way Data Binding)。
* v-model會根據輸入元素(<input>)的類型自動選取正確的方法來更新元素。
  + text和textarea元素使用value屬性和input事件
  + checkbox和radio使用checked屬性和change事件
  + select元素使用value屬性和change事件
* v-model會忽略所有表單元素的value、checked、selected屬性的初始值而總是將Vue實例的資料(data)作為資料來源。你應該透過JavaScript在元件的data選項中宣告初始值。

### 表單輸入綁定(Form Input Binding)

#### 文字(Text) (範例：T05-21.html I)



<div id="app">

使用者名稱: <input v-model="username" placeholder="請輸入使用者名稱">

<p>您的使用者名稱: **{{** username **}}**</p>

<hr />

</div>

<script>

var vm = new Vue({

el: '#app',

data: {

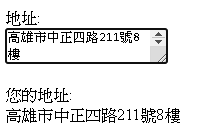
username:''

}

})

</script>

#### 文字區域(Textarea) (範例：T05-21.html II)



<div id="app">

地址:<br />

<textarea v-model="address" placeholder="請輸入地址"></textarea>

<p>您的地址:<br />**{{** address**}}**</p>

<hr />

</div>

<script>

var vm = new Vue({

el: '#app',

data: {

address:''

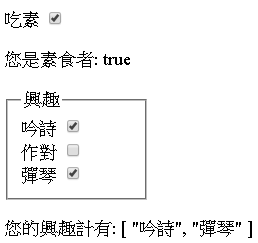
}

})

</script>

* 在文字區域插值 (<textarea>{{text}}</textarea>)並不會生效，應用v-model來代替。

#### 複選框(Checkbox) (範例：T05-21.html III)



<div id="app">

<!--單個複選框，綁定到布林值-->

<label for="vegetarian">吃素</label>

<input v-model="veg" type="checkbox" id="vegetarian" /><br />

<p>您是素食者: **{{** veg **}}**</p>

<!--多個複選框，綁定到同一個陣列-->

<fieldset style="width:100px;">

<legend>興趣</legend>

<label for="poem">吟詩</label>

<input v-model="hobbies" type="checkbox" id="poem" value="吟詩" /><br />

<label for="against">作對</label>

<input v-model="hobbies" type="checkbox" id="against" value="作對" /><br />

<label for="play">彈琴</label>

<input v-model="hobbies" type="checkbox" id="play" value="彈琴" />

</fieldset>

<p>您的興趣計有: **{{** hobbies **}}**</p>

<hr />

</div>

<script>

var vm = new Vue({

el: '#app',

data: {

veg: false,

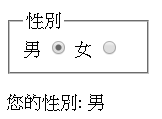
hobbies:[]

}

})

</script>

#### 單選按鈕(Radio) (範例：T05-21.html IV)



<div id="app">

<fieldset style="width:100px;">

<legend>性別</legend>

<label for="male">男</label>

<input v-model="sex" type="radio" id="male" value="男" />

<label for="female">女</label>

<input v-model="sex" type="radio" id="female" value="女" />

</fieldset>

<p>您的性別: **{{** sex **}}**</p>

<hr />

<script>

var vm = new Vue({

el: '#app',

data: {

sex: '男'

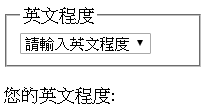
}

})

</script>

#### 選擇框(Select) (範例：T05-21.html V)

##### 單選時機(綁定到單一值)



<div id="app">

<fieldset style="width:150px;">

<legend>英文程度</legend>

<select v-model="level">

<option disabled="disabled" value="">請輸入英文程度</option>

<option>A級</option>

<option>B級</option>

<option>C級</option>

</select>

</fieldset>

<p>您的英文程度: **{{** level **}}**</p>

</div>

<script>

var vm = new Vue({

el: '#app',

data: {

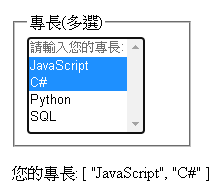
level:''

}

})

</script>

##### 多選時機(綁定到一個陣列)



<div id="app">

<fieldset style="width:150px;">

<legend>專長(多選)</legend>

<select v-model="specialties" multiple size="5">

<option disabled="disabled" value="">請輸入您的專長:</option>

<option>JavaScript</option>

<option>C#</option>

<option>Python</option>

<option>SQL</option>

</select>

</fieldset>

<p>您的專長: **{{** specialties **}}**</p>

<hr />

</div>

<script>

var vm = new Vue({

el: '#app',

data: {

specialties: []

}

})

</script>

## class 和 style 綁定

* 操作HTML元素的類別列表(class list)和行內樣式表(inline styles)是資料綁定的一個常見需求。
* 因為它們都是屬性(attributes)，所以我們可以用v-bind處理它們：只需要計算出運算式的最後字串結果即可。
* 不過，字串串連麻煩且易錯。因此，在將v-bind用於class和style時，Vue.js做了專門的增强。運算式結果的型別除了字串之外，還可以是物件或陣列。

### 綁定HTML Classes

#### 物件語法(Object Syntax)

* 我們可以傳給v-bind:class一個物件，以動態地切換class：

<p v-bind:class="{active:isActive, bold:isBold }">Hello World</p>

* + 上面的語法表示active這個class的存在將取決於資料屬性isActive的真值。
  + 你可以在物件中傳入更多屬性(properties)來動態切換多個class。此外，v-bind:class指令也可以與普通的class屬性(attribute)共存。例如：

**T05-31.html**

<div id="app">

<p class="large" v-bind:class="{active:isActive, bold:isBold }">

class object(inline)</p>



</div>

<script>

var vm = new Vue({

el: '#app',

data: {

isActive: true,

isBold: false

}

})

</script>

* 綁定的物件並非一定要定義在模板行內(inline)，也可以使用資料屬性，例如：

**T05-31.html**

<div id="app">

<p class="large" v-bind:class="classObject">

class object(data property)</p>



</div>

<script>

var vm = new Vue({

el: '#app',

data: {

classObject: { active: true, bold: true }

}

})

</script>

* 我們也可以在這裡綁定一個傳回物件的計算屬性。這是一個常用且强大的模式，例如：

**T05-31.html**

<div id="app">

<p class="large" v-bind:class="classObject2">

class object(computed property)</p>



</div>

<script>

var vm = new Vue({

el: '#app',

computed: {

classObject2: function () {

return {

active: true,

bold: true

};

}

}

})

</script>

#### 陣列語法(Array Syntax)

* 我們可以把一個陣列傳給v-bind:class，以應用一個class列表，例如：

**T05-31.html**

<div id="app">

<p class="large" v-bind:class="[activeClass,{bold:true}]">class array</p>



</div>

<script>

var vm = new Vue({

el: '#app',

data: {

activeClass: "active"

}

})

</script>

### 綁定HTML Inline Styles

#### 物件語法(Object Syntax)

* v-bind:style的物件語法十分直覺—看起來非常像CSS，但其實是一個JavaScript物件。
* CSS屬性名可以用駝峰式(camelCase)或短橫線分隔(kebab-case)(需用引號括起來)來命名：
* 直接綁定到一個樣式物件(style object)通常更好，這會讓模板更清晰。
* 同樣的，物件語法常常結合傳回物件的計算屬性使用。
* 例如：

**T05-31.html**

<div id="app">

<div v-bind:style="{ color: activeColor, fontSize: fontSize + 'px' }">

style object(inline)</div>

<div v-bind:style="styleObject">style object(data property)</div>



</div>

<script>

var vm = new Vue({

el: '#app',

data: {

activeColor: 'red',

fontSize: 26,

styleObject: {

color: 'red',

"font-size":"26px"

}

}

})

</script>

#### 陣列語法(Array Syntax)

* v-bind:style的陣列語法可以將多個樣式物件應用到同一個元素上。
* 例如：

<div id="app">

<div v-bind:style="[styleObject2,{fontSize: fontSize+'px'}]">style array</div>



</div>

<script>

var vm = new Vue({

el: '#app',

data: {

fontSize: 26,

styleObject2: {

color: 'red'

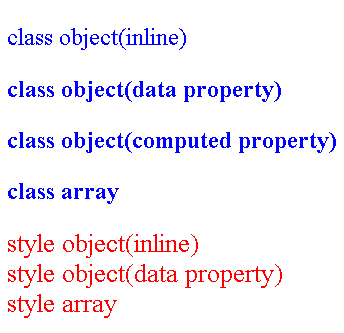
}

}

})

</script>

### HTML Class與Style綁定完整範例：T05-31.html



<!DOCTYPE html>

<html>

<head>

(略)

<style>

.active { color:blue; }

.bold {font-weight:bold;}

.large {font-size:1.5em;}

</style>

</head>

<body>

<div id="app">

<p class="large" v-bind:class="{active:isActive, bold:isBold }">class object(inline)</p>

<p class="large" v-bind:class="classObject">class object(data property)</p>

<p class="large" v-bind:class="classObject2">class object(computed property)</p>

<p class="large" v-bind:class="[activeClass,{bold:true}]">class array</p>

<div v-bind:style="{ color: activeColor, fontSize: fontSize+'px' }">style object(inline)</div>

<div v-bind:style="styleObject">style object(data property)</div>

<div v-bind:style="[styleObject2,{fontSize: fontSize+'px'}]">style array</div>

</div>

<script>

var vm = new Vue({

el: '#app',

data: {

isActive: true,

isBold: false,

classObject: {

active: true,

bold: true

},

activeClass: "active",

activeColor: 'red',

fontSize: 26,

styleObject: {

color: 'red',

"font-size":"26px"

},

styleObject2: {

color: 'red'

}

},

computed: {

classObject2: function () {

return {

active: true,

bold: true

};

}

}

})

</script>

</body>

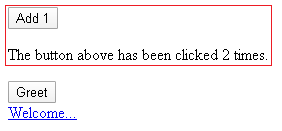
</html>

# Module 6. 網頁操作行為互動和自定義指令

## 事件處理和繫結

* 我們可以用v-on指令監聽DOM事件，並在觸發時執行一些JavaScript程式碼。
* v-on:指令的簡寫為@(有人將shorthand解釋為一種語法糖/syntax sugar)。例如：v-on:click可以簡寫為@click

### 範例：T06-11.html (I)



<body>

<div id="app">

<button v-on:click="counter += 1">Add 1</button>

<p>The button above has been clicked **{{** counter **}}** times.</p>

</div>

<script>

var vm = new Vue({

el: '#app',

data: {

counter: 0

}

})

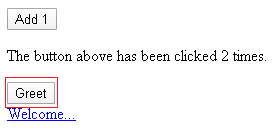
</script>

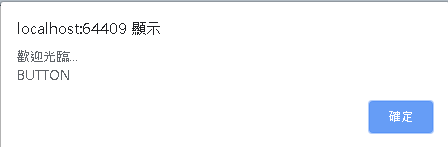
</body>

</html>

* 然而許多事件處理邏輯會更為複雜，所以直接把JavaScript程式碼寫在v-on 指令中是不可行的。因此v-on還可以接收一個需要呼叫的方法名稱。

### 範例：T06-11.html (II)





<div id="app">

<button v-on:click="greet">Greet</button>

</div>

<script>

var vm = new Vue({

el: '#app',

methods: {

greet: function (event) {// `event` is the native DOM event

alert("歡迎光臨...\n" + event.target.tagName);

//event.target指向最初觸發事件的DOM物件。

//event.currentTarget屬性總會指向目前於冒泡階段正在處理該事件

之事件處理器所註冊的DOM物件。

}

}

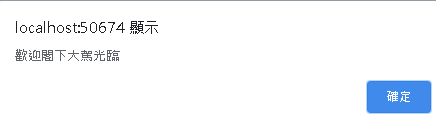
})

</script>

* 除了如上述直接綁定一個方法名稱，也可以在JavaScript陳述式中呼叫方法，且有時可能需要在事件處理器方法中存取原始的DOM事件，則可以用特殊的變數$event把它傳入方法：例如：

### 範例：T06-11.html (III)





<div id="app">

<a href="#" v-on:click="welcome('歡迎閣下大駕光臨', $event);">Welcome...</a>

</div>

<script>

var vm = new Vue({

el: '#app',

methods: {

welcome: function (message, event) {

alert(message);

event.preventDefault();

}

}

})

</script>

## 事件修飾符：Event-Modifiers

* 在事件處理器方法中呼叫event.preventDefault()或event.stopPropagation()是非常常見的需求。儘管我們可以在方法中輕鬆實現這點，但更好的方式是：方法只有純粹的資料邏輯，而不是處理DOM 事件細節。
* 為了解决這個問題，Vue.js為v-on提供了事件修飾符。修飾符是以點(dot)開頭的指令後綴(directive postfixes)來表示的。
  + <a v-on:click.stop="onClick"></a>

阻止點擊事件繼續傳播。

* + - Event Bubbling：When an event happens on an element, it first runs the handlers on it, then on its parent, then all the way up on other ancestors.
    - Almost all events bubble. (For instance, a focus event does not bubble.)
  + <a v-on:click.prevent="onClick"></a>

取消事件的預設行為。

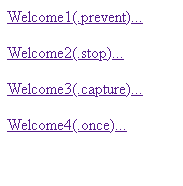
* + <a v-on:click.capture="onClick"></a>

添加事件監聽器時使用事件捕獲模式/Capture mode(即内層元素/inner element觸發的事件先在此(外層元素)處理，然後才交由該內層元素進行處理)。

* + <a v-on:click.once="onClick"></a>

點擊事件將只會觸發一次。

### 範例：T06-21.html



<!DOCTYPE html>

<html>

(略)

<body>

<div id="app">

<a href="https://vuejs.org/" v-on:click.prevent=

"welcome('1 click event from <a>');">Welcome1(.prevent)...</a>

<div v-on:click="welcome('2 click event from <div>')">

<p v-on:click="welcome('2 click event from <p>')">

<a href="#" v-on:click.stop=

"welcome('2 click event from <a>');">Welcome2(.stop)...</a>

</p>

</div>

<div v-on:click.capture="welcome('3 click event from <div>')">

<p v-on:click.capture="welcome('3 click event from <p>')">

<a href="#" v-on:click=

"welcome('3 click event from <a>');">Welcome3(.capture)...</a>

</p>

</div>

<div v-on:click.once="welcome('4 click event from <div>')">

<p v-on:click="welcome('4 click event from <p>')">

<a href="#" v-on:click.once=

"welcome('4 click event from <a>');">Welcome4(.once)...</a>

</p>

</div>

</div>

<script>

var vm = new Vue({

el: '#app',

methods: {

welcome: function (message) {

alert(message);

}

}

})

</script>

</body>

</html>

## Vue.directive(id, definition)

### 自定義指令簡介

* Vue除了核心功能所預設内置的指令外，也允許註冊自定義指令(custom directives)。
* 在Vue2.0 中，程式碼復用和抽象化(code reuse and abstraction)的主要形式是元件(Components)。然而，有的情况下，你仍然需要對普通DOM元素進行底層操作，這時候就會用到自定義指令。
* 自定義指令API
  + Vue.directive( id, [definition] )
    - Register a global directive.
    - Arguments:
      * {string} id
      * {Object} [definition]
    - 一個指令定義物件(directive definition object)可以提供如下幾個鈎子函數(均為選用的)：
      * bind：只呼叫一次，指令第一次綁定到元素時呼叫。在這裡可以進行一次性的初始化設置。
      * inserted：被綁定元素插入父節點時呼叫。
      * update：指令所在元件的VNode更新時呼叫，但是可能發生在其子VNode更新之前。
      * componentUpdated：指令所在元件的VNode及其子VNode全部更新後呼叫。
      * unbind：只呼叫一次，指令與元素解綁時(例如Vue移除該元素時，如v-if 指令值由true變為false)呼叫。
    - 簡例：

Vue.directive('my-directive', {

bind: function (el) {},

inserted: function (el) {},

update: function (el) {},

componentUpdated: function (el) {},

unbind: function (el) {}

});

* 舉個輸入框「聚焦」的例子如下，當頁面載入時該元素將獲得焦點。

### 範例：T06-31.html

<!DOCTYPE html>

<html>

(略)

<body>

<div id="app">

<p>使用者名稱 <input v-focus type="text" name="username" size="10"/></p>

<p>密碼 <input type="text" name="password" size="10" /></p>

</div>

(Corelab 18-2)

<script>

// Register a global custom directive called `v-focus`

//Vue.directive( id, [definition] ) Arguments:{string} id, {Function | Object} [definition]

Vue.directive("focus", {

// When the bound element is inserted into the DOM...

inserted: function (el) {

el.focus(); // Focus the element

}

});

var vm = new Vue({

el: '#app',

directives: {

// Register a local custom directive called `v-focus` (優先於global版)

focus: {

// directive definition

inserted: function (el) {

el.focus();// Focus the element

}

}

}

});

</script>

</body>

</html>

# Module 7. JavaScript ES6 - 宣告和常用方法

## 變數宣告的方式與作用域

* 變數宣告方式主要有二：
  + var 變數名;
  + let 變數名; (ES6)
* let宣告的變數具有區塊可視範圍/作用域(scope)；var宣告的變數則不具有區塊可視範圍;。
* let變數在同一個區塊內不能重複宣告；var變數則可重複宣告。

### 範例：T07-11.html

<script>

{

var v1 = 100; //全域變數

let v2 = 100; //區塊變數

console.log("v1=%d v2=%d", v1, v2);//print: v1=100 v2=100

}

console.log("v1=%d", v1);//print: v1=100

//console.log("v2=%d", v2); //Uncaught ReferenceError: v2 is not defined

function f1() {

var v3 = 100; //區域變數

let v4 = 100; //區塊變數

console.log("v3=%d v4=%d", v3, v4); //print v3=100 v4=100

}

//console.log("v3=%d", v3); //Uncaught ReferenceError: v3 is not defined

//console.log("v4=%d", v4); //Uncaught ReferenceError: v4 is not defined

f1();

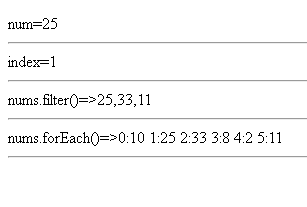
//console.log("v3=%d", v3); //Uncaught ReferenceError: v3 is not defined

//console.log("v4=%d", v4); //Uncaught ReferenceError: v4 is not defined

</script>

## 陣列常用的方法

### 範例：T07-21.html



<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<title>陣列常用的方法</title>

</head>

<body>

<p id="p1"></p>

<script>

var nums = [10, 25, 33, 8, 2, 11];

var p = document.getElementById("p1");

var num = nums.find(function (value, index, array) {

return value > 10;

});

p.innerHTML = `num=${num}<hr/>`; //num=25

//`...` 樣板字面值(Template literals)是允許嵌入運算式的字串字面值(string literals)。

//------------------------------------------------------------------//

var index = nums.findIndex(function (value) {

return value > 10;

});

p.innerHTML += `index=${index}<hr/>`; //index=1

//------------------------------------------------------------------//

var arr = nums.filter( function (el) {

return el>10;

});

p.innerHTML += `nums.filter()=>${arr.toString()}<hr/>`; //nums.filter()=>25,33,11

//------------------------------------------------------------------//

s = "";

nums.forEach(function (el, index) {

s += `${index}:${el} `;

});

p.innerHTML += `nums.forEach()=>${s}<hr/>`; //nums.forEach()=>0:10 1:25 2:33 3:8 4:2 5:11

//------------------------------------------------------------------//

</script>

</body>

</html>

* Array.find()：會回傳第一個滿足所提供之測試函數的元素值。否則回傳undefined。
  + 語法：array.find( function(currentValue[, index, array]) )

T07-21.html

<p id="p1"></p>

<script>

var nums = [10, 25, 33, 8, 2, 11];

var p = document.getElementById("p1");

var num = nums.find(function (value, index, array) {

return value > 10;

});

p.innerHTML = `num=${num}<br/>`; //num=25

//`...` 樣板字面值(Template literals)是允許嵌入運算式的字串字面值(string literals)。

</script>

* Array.findIndex()：將依據提供的測試函數，尋找陣列中符合的元素，並返回其index(索引)。如果沒有符合的對象，將返回 -1。
  + 語法：array.findIndex( function(currentValue[, index, array]) )

T07-21.html

<p id="p1"></p>

<script>

var nums = [10, 25, 33, 8, 2, 11];

var p = document.getElementById("p1");

var index = nums.findIndex(function (value) {

return value > 10;

});

p.innerHTML += `index=${index}<br/>`; //index=1

</script>

* Array.filter()：會建立一個經指定之函式運算後，由原陣列中透過該函式檢驗之元素所構成的新陣列。
  + 語法：array.filter( function(currentValue[, index, array]) )

T07-21.html

<p id="p1"></p>

<script>

var nums = [10, 25, 33, 8, 2, 11];

var p = document.getElementById("p1");

var arr2 = nums.filter( function (el) {

return el>10;

});

p.innerHTML += `nums.filter()=>${arr2.toString()}<br/>`; //nums.filter()=>25,33,11

</script>

* Array.forEach()：會將陣列內的每個元素，皆傳入並執行給定的函式一次。
  + 語法：array.forEach( function(currentValue[, index, array]) )

T07-21.html

<p id="p1"></p>

<script>

var nums = [10, 25, 33, 8, 2, 11];

var p = document.getElementById("p1");

s = "";

nums.forEach(function (el, index) {

s += `${index}:${el} `;

});

p.innerHTML += `nums.forEach()=>${s}<br/>`; //nums.forEach()=>0:10 1:25 2:33 3:8 4:2 5:11

</script>

## 匯入和輸出

\*\*Corelab小測驗題目有誤\*\*

### 匯入(import)

* import敘述用於匯入由另一個模組(module)所匯出的函數、物件、變數、常數或類別等。
* import 敘述無法用於嵌入式腳本（embedded scripts），除非使用<script type="module">。
* 常用語法
  + import defaultExportName from "module-name";
  + import defaultExportName, { exportName [ , [...] ] } from "module-name";

說明：

* + - defaultExportName

“其他模組所預設匯出(export default) ”在匯入模組的名稱。

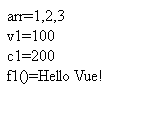
* + - module-name
* 要匯入的模組名。
* 通常包含 .js 模組檔案的相對或絕對路徑名。
* 只允許使用單引號和雙引號字串。
  + - exportName

其他模組所匯出(export)的名稱。

### 輸出(export)

* export敘述可用來從模組匯出函數、物件、變數、常數或類別等，其他程式則可透過 import敘述將之匯入。
* 匯出敘述無法使用於嵌入式腳本（embedded script）。
* 常用語法
  + // 匯出個別項目
    - export let name1 = …, name2 = …, …, nameN; // also var, const
    - export function functionName(){...}
    - export class ClassName {...}
  + // 匯出列表(list)
    - export { name1, name2, …, nameN };
  + // 預設匯出
    - export default expression; (註: expression不可是變數宣告如: ~~var~~ v1=100 )
    - export default function [name1](…) { … } // also class

### 範例：T07-31.js、T07-31.html



T07-31.html

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<title></title>

<script type="module">

import arr, { v1, c1, f1 } from "./T07-31.js"; //Relative references must start with either

"/", "./", or "../"

var s = "";

s += "arr=" + arr + "<br/>";

s += "v1=" + v1 + "<br/>";

s += "c1=" + c1 + "<br/>";

s += "f1()=" + f1() + "<br/>";

document.getElementById("app").innerHTML = s;

</script>

</head>

<body>

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

</body>

</html>

T07-31.js

//1. 預設匯出(Default Exports) (Zero or one per module)

export default [1,2,3];

//2. 具名匯出(Named Exports) (Zero or more exports per module)

//2-1. 匯出個別項目

export var v1 = 100;

//2-2 匯出列表(list)

const c1 = 200;

function f1() { return "Hello Vue!"; };

export {c1, f1};

# Module 8. JavaScript ES6 - Function

## Arrow function

* 箭頭函數有比Function Literal還簡短的語法。
* 箭頭函數沒有自己的 this 等語法，適用於非物件方法的函數，但不能被用作建構子函數(constructor)。例如：

var myObject = {

name:'小倩',

laugh: () => {

console.log(this.name + '笑笑'); //(註:此處this是window)

},

cry: function () {

console.log(this.name + '哭哭');

}

}

myObject.laugh(); //print: 笑笑

myObject.cry(); //print: 小倩哭哭

* 箭頭函數的結構：

( parameters ) => { Statement1; Statement2; }

* + 當參數只有一個時，(parameters)的小括號可以省略；當參數為零個時，(parameters) 的小括號不可以省略。
  + 當主體只含一個敘述時，左右大括號可以省略(但敘述結尾的分號必須拿掉)。
  + 左右大括號省略時，如果唯一的敘述是return value;，則return關鍵字與分號必須省略。
  + 函數的主體內若沒有return一個值，則該函數的回傳值為undefined 。(Corelab 9-1)

### 範例：T08-11.html

<script>

var f1 = (v1) => { console.log(v1); }

f1(100); //print: 100

//-------------------------------------------------------//

var nums = [10, 25, 33, 8, 2, 11];

var num = nums.find(e => e > 10);

console.log("num=%d", num); //print: num=25

//-------------------------------------------------------//

var o1 =

{

name: '史奴比',

cry: () => {

console.log(this.name + "汪汪汪...");//this: window

},

cry2: function () {

console.log(this.name + "汪汪汪..."); //this: current object

}

}

o1.cry(); //print: 汪汪汪...

o1.cry2(); //print: 史奴比汪汪汪...

//-------------------------------------------------------//

</script>

## Function chaining

* 函數鏈(Function chaining)是物件導向程式語言利用一個敘述呼叫多個方法的通用語法。
* 函數鏈的實作主要是透過在函數或方法的主體裡傳回this來達成的。

### 範例：T08-21.html

<script>

function f1() {

var v1 = "f1...";

console.log(v1);

return this; //this: window

}

function f2() {

let v1 = "f2...";

console.log(v1);

return this; //this: window

}

f1().f2();

//print: f1...

//print: f2...

class Dog {

constructor(name) {

this.name = name;

}

//原型方法

cry() {

console.log(this.name + "嘻嘻嘻...");

return this; //this: current object

}

laugh() {

console.log(this.name + "哇哇哇...");

return this; //this: current object

}

}

new Dog("小白").cry().laugh();

//print: 小白嘻嘻嘻...

//print: 小白哇哇哇...

</script>

## 撰寫方式比較

### 範例：T08-31.html

<script>

//具名函數

function f1(x, y) {

console.log(`${x}+${y}=${x+y}`);

}

f1(10,20); //print: 10+20=30

//匿名函數

var f2 = function (x, y) {

console.log(`${x}+${y}=${x + y}`);

}

f2(10,20); //print: 10+20=30

//箭頭函數

var f3 = (x, y) => console.log(`${x}+${y}=${x + y}`);

f3(10,20); //print: 10+20=30

</script>

# Module 9. Vue 元件 - 樣板功能

## 註冊全域/區域樣板

### Vue元件導論

* 元件(Component)是可重複使用的Vue實例，且帶有一個名字。
* 我們可以在一個透過new Vue 建立的Vue根實例中，把這個元件作為自定義元素來使用。
* 因為元件是可重複利用的Vue實例，所以它們與new Vue 接收相同的選項，例如data、computed、watch、methods 以及生命週期鈎子等。僅有的例外是像el這項根實例特有的選項。
* 通常一個應用程式會以一棵巢狀的元件樹的形式來組織：



參考資料來源：<https://vuejs.org/v2/guide/components.html#Organizing-Components>

* + 例如，你可能會有頁頭、側邊欄、内容區等元件，每個元件又包含了其它的元件，如導航鏈結、博文(blog posts)之類的元件。
  + 為了能在模板中使用，這些元件必須先註冊以便Vue能够識別。這裡有兩種元件的註冊(component registration)類型：全域(global)註冊和區域(local)註冊。

### 元件註冊(Component Registration)

#### 全域註冊

* 全域註冊的元件可以用在其被註冊之後的任何(透過new Vue)新建立的Vue根實例(也包括其元件樹中的所有子元件)的模板中。
* 元件全域註冊API
  + Vue.component( id, [definition] )
    - Register a global component. Registration also automatically sets the component’s name with the given id.
    - Arguments:
      * {string} id (id可以使用駝峰式(camelCase)或短橫線分隔(kebab-case)來命名，例如my-component或myComponent;但因HTML不區分大小寫，所以在對應模板中只能使用短橫線分隔(kebab-case)，例如<my-component>) (Corelab 11-3)
      * {Function | Object} [definition]
  + 例如：

//註冊元件

Vue.component('component-a', { /\* options object \*/ })

Vue.component('component-b', { /\* options object \*/ })

Vue.component('component-c', { /\* options object \*/ })

//建立根Vue實例

new Vue({ el: '#app' })

//根Vue實例之模板

<div id="app">

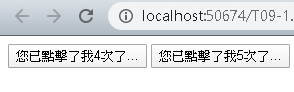
<component-a></component-a>

<component-b></component-b>

<component-c></component-c>

</div>

##### 範例：T09-11.html



<!DOCTYPE html>

<html>

(略)

<template id="template1">

<button v-on:click="count++">

您已點擊了我{{count}}次了...

</button>

</template >

<body>

<div id="app">

<button-counter></button-counter>

<button-counter></button-counter>

</div>

<script>

//定義(註冊)一個名為button-counter的全域的新元件

//Vue.component( id, [definition] )

Vue.component("button-counter", {

data: function () {

return {

count: 0

}

},

//\*\*\*template屬性設定方法1: (template屬性值必須是"具單一根元素格式"之HTML字串)

template: `<button v-on:click='count++'>

您已點擊了我{{count}}次了...

</button>`

//\*\*\*template屬性設定方法2:(template屬性值必須是"<template>元素之ID選擇器"之字串)

//template: '#template1'

});

var vm = new Vue({

el: '#app'

})

</script>

</body>

</html>

* 注意：元件的data屬性必須是一個函數，如此當點擊按鈕時，每個元件都會各自獨立維護它的count。

#### 區域註冊

* 全域註册往往是不够理想的。比如，如果你使用一個像webpack這樣的建構系統(build system)，全域註冊所有的元件意味著即便你已經不再使用一個元件了，它仍然會被包含在你最終的建構結果中。這造成了用戶下載的JavaScript的無謂的增加。
* 在這些情况下，你可以定義你的元件為一個普通的JavaScript物件：

var ComponentA = { /\* options object \*/ }

var ComponentB = { /\* options object \*/ }

var ComponentC = { /\* options object \*/ }

然後在components選項中定義你想要使用的元件：

new Vue({

el: '#app',

components: {

'component-a': ComponentA,

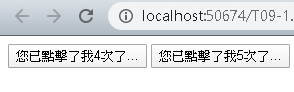
'component-b': ComponentB

}

})

* 對於components物件中的每個屬性來說，其屬性名就是自定義元素(元件)的名字，其屬性值就是這個元件的選項物件。

##### 範例：T09-12.html



<!DOCTYPE html>

<html>

(略)

<body>

<div id="app">

<button-counter></button-counter>

<button-counter></button-counter>

</div>

<script>

var buttonCounter = {

data: function () {

return {

count: 0

}

},

template: `<button v-on:click='count++'>

您已點擊了我{{count}}次了...

</button>`

};

var vm = new Vue({

el: '#app',

//註冊一個名為button-counter的區域元件

components: {

"button-counter": buttonCounter

}

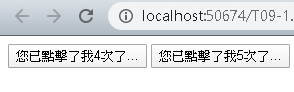
})

</script>

</body>

</html>

##### 範例：T09-12-import.html，T09-12-export.js



T09-12import.html

<!DOCTYPE html>

<html>

(略)

<body>

<div id="app">

<button-counter></button-counter>

<button-counter></button-counter>

</div>

<script type="module"> //Cannot use import statement outside a module

import buttonCounter from './T09-12export.js';

var vm = new Vue({

el: '#app',

components: {

buttonCounter //相當於 buttonCounter: buttonCounter

}

})

</script>

</body>

</html>

T09-12export.js

export default

{

data: function () {

return {

count: 0

}

},

template: `<button v-on:click='count++'>

您已點擊了我{{count}}次了...

</button>`

//template必須是"單一根元素"的形式

};

## return object 建構資料格式

* 你可以將元件進行任意次數的重複利用。因為你每用一次元件，就會有一個它的新實例被建立。
* 元件的data屬性必須是一個函數，如此每個實例才可以維護一份被傳回物件的獨立的拷貝。(Corelab 12-1)
* 範例程式請參考9.1.2.1.1

## Props 觀念和型別預設值

### 以Prop向子元件(Child Components)傳遞資料

* Prop是你可以在元件上註冊的一些自定義屬性(custom attributes)。
* 當一個值傳遞給一個prop屬性(attribute)的時候，它就變成了那個元件實例的一個屬性(property)。

Vue.component('blog-post', {

props: ['title'],

template: '<h3>{{ title }}</h3>'

});

* 一個元件預設可以擁有任意數量的prop，任何值都可以傳遞給任何 prop。
* 一個prop被註冊之後，你就可以把資料作為一個自定義屬性值傳遞進來：

<blog title="My journey with Vue"></blog>

### Prop型別預設值

* 元件的props屬性值除了是字串陣列形式，如：

props: ['title', 'likes', 'isPublished', 'commentIds', 'author']

* 通常你會希望每個 prop的值都有指定的型別。這時你可以以物件形式列出 props，物件的屬性名稱和值分别是prop各自的名稱和型別：

props: {

title: String,

likes: Number,

isPublished: Boolean,

commentIds: Array,

author: Object,

callback: Function

}

* 這不僅為你的元件提供了說明文件，還會在它們遇到錯誤的型別時從瀏覽器的JavaScript主控台提示用戶。
  + 型別檢查(Type Check)的type
    - 可以是下列**原生**建構子函數中的一個，例如：
      * String
      * Number
      * Boolean
      * Array
      * Object
      * Date
      * Function
    - 還可以是一個自定義的建構子函數(custom constructor function)。
* 除了上述型別檢查(type check)以外，我們可以為元件的prop指定預設值：

Vue.component('my-component', {

props: {

// 帶有**數目**預設值

propD: {

type: Number,

default: 100

},

// 帶有**物件**預設值

propE: {

type: Object,

// 物件或陣列預設值必須從一個工廠函數獲取

default: function () {

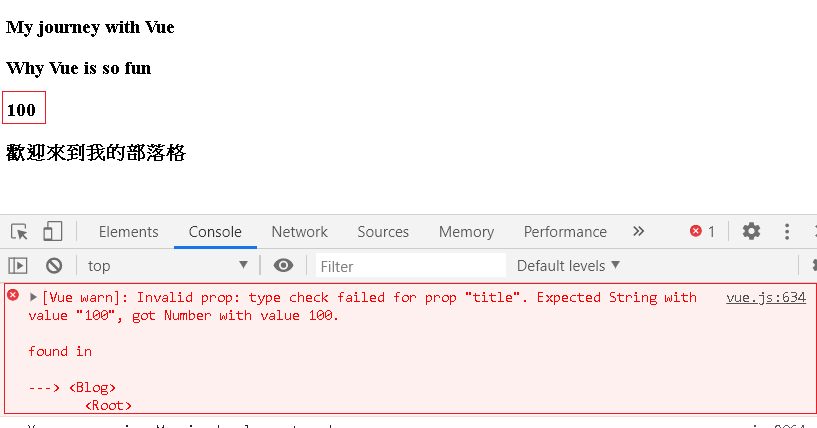
return { message: 'hello' }

}

}

}

### 範例：T09-31.html



<!DOCTYPE html>

<html>

(略)

<body>

<div id="app">

<blog title="My journey with Vue"></blog>

<blog title="Why Vue is so fun"></blog>

<blog v-bind:title="num"></blog>

<blog></blog>

</div>

<script>

//定義(註冊)一個名為blog的全域的新元件

Vue.component('blog', {

//props: ['title'],

props: {

'title': {

type: String,

default: '歡迎來到我的部落格'

}

},

template: '<h3>{{ title }}</h3>'

});

var vm = new Vue({

el: '#app',

data: {

num: 100

}

})

</script>

</body>

</html>

# Module 10. 動態設定樣板內容

## 不具名 slot

* 和HTML元素一樣，我們經常需要向一個元件傳遞内容。
* Vue 實作了一套内容分發(content distribution)的API，將插槽(<slot>)元素作為承載分發内容的出口(distribution outlets)。
* 插槽内可以包含任何模板程式碼，包括 HTML。
* 一個不帶name的<slot> 出口會帶有隱含的名字「default」。

### 範例：T10-11.html



<!DOCTYPE html>

<html>

(略)

<body>

<div id="app">

<alert-box>

Something bad happened.

</alert-box>

</div>

<script>

Vue.component('alert-box', {

template:`

<div class="alert-box">

<strong>Error!</strong>

<slot></slot>

</div>

`

});

new Vue({

el: '#app'

});

</script>

</body>

</html>

* 有時為一個插槽設置具體的後備内容(fallback content) ，也就是預設內容(default content)是很有用的，它只會在没有提供内容的時候被渲染。例如
* 在一個 <submit-button>元件中定義template：

<button type="submit">

<slot>Submit</slot>

</button>

* 當在一個父級元件中使用 <submit-button>並且不提供任何插槽内容時，如

<submit-button></submit-button>

* 則後備内容“Submit”將會被渲染：

<button type="submit">

Submit

</button>

## 具名slot (Named Slot)

* 有時我們需要多個插槽。對於這樣的情况，<slot> 元素有一個特殊的 attribute：name， 可以用來定義額外的插槽。

<div class="container">

<header>

<slot name="header"></slot>

</header>

<main>

<slot></slot> (註：name=”default”)

</main>

<footer>

<slot name="footer"></slot>

</footer>

</div>

* 在向具名插槽提供内容的時候，我們可以在一個<template>元素上使用v-slot 指令，並以v-slot 的引數的形式提供其名稱：
* 任何没有被包裹在帶有v-slot 的<template>中的内容都會被視為預設插槽的内容。
* 然而，如果你希望更明確一些，仍然可以在一個<template>中包裹預設插槽的内容：

<base-layout>

<template v-slot:header>

<h1>Here might be a page title</h1>

</template>

<template v-slot:default>

<p>A paragraph for the main content.</p>

<p>And another one.</p>

</template>

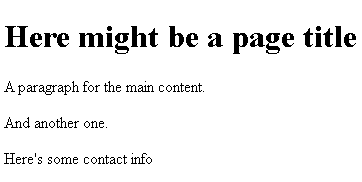
<template v-slot:footer>

<p>Here's some contact info</p>

</template>

</base-layout>

### 範例：T10-21.html



<!DOCTYPE html>

<html>

(略)

<body>

<div id="app">

<base-layout>

<template v-slot:header>

<h1>Here might be a page title</h1>

</template>

<p>A paragraph for the main content.</p>

<p>And another one.</p>

<template v-slot:footer>

<p>Here's some contact info</p>

</template>

</base-layout>

</div>

<script>

Vue.component('base-layout', {

template:

`<div class="container">

<header>

<slot name="header"></slot>

</header>

<main>

<slot></slot>

</main>

<footer>

<slot name="footer"></slot>

</footer>

</div>

`

});

new Vue({

el: '#app'

});

</script>

</body>

</html>

## scoped-slot

* 有時讓父級元件的插槽内容(slot content)能够存取子元件中才有的資料是很有用的。
* 為了讓子元件的資料(例如user物件)能在父級元件的插槽内容中可用，我們可以將子元件的資料物件內的data\_user物件綁定為<slot>元素的一個attribut：

<span>

<slot v-bind:user="data\_user">

{{ data\_user.lastName }}

</slot>

</span>

* 綁定在<slot>元素上的attribute被稱為插槽props(slot props)。現在在父級作用域(parent scope)中，我們可以使用帶值的v-slot來定義我們提供的slot props的名字：

<current-user>

<template v-slot:default="slotProps">

{{ slotProps.user.firstName }}

</template>

</current-user>

* 在這個例子中，我們選擇將包含所有**插槽props**的物件命名為 slotProps，但你也可以使用任意你喜歡的名字。

### 範例：T10-31.html



<!DOCTYPE html>

<html>

(略)

<body>

<div id="app">

<current-user v-slot="slotProps">

Welcome: **{{** slotProps.user.firstName **}}**

</current-user>

</div>

<script>

Vue.component('current-user', {

data: () => ({ data\_user: {firstName:'Kitty',lastName:'Nancy'} }),

template:

`<span>

<slot v-bind:user="data\_user">

Welcome: {{ data\_user.lastName }}

</slot>

</span>

`

});

new Vue({

el: '#app'

});

</script>

</body>

</html>

# Module 11. Vue 元件 - 資料和程式碼

## 重複使用程式碼

* 在Vue2.0 中，元件(Components)是程式碼復用和抽象化(code reuse and abstraction)的主要形式。
* 動態元件(Dynamic Components)
  + 有的時候，在不同元件之間進行動態切換是非常有用的，比如在一個多標籤(tabbed)的頁面裡。
  + 動態元件切換可以透過Vue的<component>元素加一個特殊的is屬性(其值設定為特定元件的id)來實現：

<!--元件會在`currentTabComponent`改變時改變 -->

<component v-bind:is="currentTabComponent"></component>

* + 上述示例的currentTabComponent 可以包括已註冊元件的名字或一個元件的選項物件

### 範例：T11-11.html(動態元件)



<!DOCTYPE html>

<html>

<head>

(略)

<style>

.tab-button {

padding: 6px 10px; border-top-left-radius: 3px;

border-top-right-radius: 3px; border: 1px solid #ccc;

cursor: pointer; background: #f0f0f0; margin-bottom: -1px;

}

.tab-button:hover {

background: #e0e0e0;

}

.tab-button.active {

background: #e0e0e0;

}

.tab {

border: 1px solid #ccc;

padding: 10px;

}

</style>

</head>

<body>

<div id="app">

<button v-for="tab in tabs"

v-bind:key="tab"

v-bind:class="['tab-button', { active: currentTab === tab }]"

v-on:click="currentTab = tab">

**{{** tab **}}**

</button>

<!--元件會在`currentTabComponent`改變時改變 -->

<component v-bind:is="currentTabComponent" class="tab">

</component>

</div>

<script>

//註冊一個名為tab-home的全域的新元件

Vue.component("tab-home", {

template: "<div>Home component</div>"

});

Vue.component("tab-posts", {

template: "<div>Posts component</div>"

});

Vue.component("tab-archive", {

template: "<div>Archive component</div>"

});

new Vue({

el: "#app",

data: {

currentTab: "Home",

tabs: ["Home", "Posts", "Archive"]

},

computed: {

currentTabComponent: function () {

return "tab-" + this.currentTab.toLowerCase();

}

}

});

</script>

</body>

</html>

## 保留表單狀態

* 當在動態元件之間切換的時候，你有時會想保持這些元件的狀態，以避免反覆重渲染(re-rendering)導致的性能(performance)問題。
* 在前面的範例裡，每次切換新標籤的時候，Vue都建立了一個新的currentTabComponent實例。
* 重新建立動態元件的行為通常是非常有用的，但是在這個案例中，我們更希望那些標籤對應的元件實例能够在它們第一次被建立的時候被快取(cached)下來。
* 為了解决這個問題，我們可以用一個<keep-alive>元素將其動態元件包裹起來。(Corelab 10-3)

<keep-alive>

<component v-bind:is="currentTabComponent"></component>

</keep-alive>

### 範例：T11-21.html



<!DOCTYPE html>

<html>

<head>

(略)

</head>

<body>

<div id="app">

<button v-for="tab in tabs"

v-bind:key="tab"

v-bind:class="['tab-button', { active: currentTab === tab }]"

v-on:click="currentTab = tab">

**{{** tab **}}**

</button>

<keep-alive>

<!--元件會在`currentTabComponent`改變時改變 -->

<component v-bind:is="currentTabComponent" class="tab"></component>

</keep-alive>

</div>

<script>

//註冊一個名為tab-home的全域的新元件

Vue.component("tab-home", {

template: "<div>Home component<br><input type='text' /></div>"

});

Vue.component("tab-posts", {

template: "<div>Posts component<br><input type='text' /></div>"

});

Vue.component("tab-archive", {

template: "<div>Archive component<br><input type='text' /></div>"

});

new Vue({

el: "#app",

data: {

currentTab: "Home",

tabs: ["Home", "Posts", "Archive"]

},

computed: {

currentTabComponent: function () {

return "tab-" + this.currentTab.toLowerCase();

}

}

});

</script>

</body>

</html>

## 多檔案連結

* 異步元件(Async Components)
  + 在大型應用中，我們可能需要將應用程式分割成小一些的程式區塊，並且只在需要的時候才從伺服器載入一個元件。
  + 為了簡化，Vue允許你以一個工廠函數(factory function)的方式定義你的元件，這個工廠函數會異步解析你的元件定義。
  + Vue只有在這個元件需要被渲染的時候才會觸發該工廠函數，且會把結果快取(cache)起來供未來再渲染(re-render)用。例如：

Vue.component("tab-home", function (resolve, reject) {

$.getScript("./T11-31.js", function () {

resolve(componentOptionsObject);

});

//jQuery.getScript(url, success)

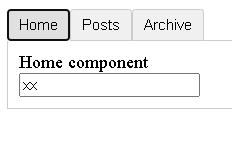
//Load a JavaScript file from the server using a GET HTTP request, then execute it.

});

* + 如你所見，這個工廠函數會收到一個 resolve回呼函數(callback)，這個回呼函數會在你從伺服器得到元件定義的時候被呼叫。你也可以呼叫 reject(reason)來表示載入失敗。

### 範例：T11-31.html(異步元件)、T11-31.js

T11-31.html



<!DOCTYPE html>

<html>

(略)

<body>

<div id="app">

<button v-for="tab in tabs"

v-bind:key="tab"

v-bind:class="['tab-button', { active: currentTab === tab }]"

v-on:click="currentTab = tab">

**{{** tab **}}**

</button>

<keep-alive>

<!--元件會在`currentTabComponent`改變時改變 -->

<component v-bind:is="currentTabComponent" class="tab"></component>

</keep-alive>

</div>

<script>

//註冊一個名為tab-home的全域的新元件

Vue.component("tab-home", function (resolve, reject) {

$.getScript("./T11-31.js", function () {

resolve(componentOptionsObject);

});

//jQuery.getScript(url, success)

//Load a JavaScript file from the server using a GET HTTP request, then execute it.

});

Vue.component("tab-posts", {

template: "<div>Posts component<br><input type='text' /></div>"

});

Vue.component("tab-archive", {

template: "<div>Archive component<br><input type='text' /></div>"

});

new Vue({

el: "#app",

data: {

currentTab: "Home",

tabs: ["Home", "Posts", "Archive"]

},

computed: {

currentTabComponent: function () {

return "tab-" + this.currentTab.toLowerCase();

}

}

});

</script>

</body>

</html>

T11-31.js

var componentOptionsObject =

{

template: "<div>Home component<br><input type='text' /></div>"

}

# Module 12. 事件傳遞

## watch & computed 比較

### 計算屬性(Computed Properties)

* 模板内的運算式非常便利，但是設計他們的初衷是用於簡單運算的。在模板中放入太多的邏輯會讓模板過大且難以維護。例如：

<div id="app">

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

</div>

* 所以，對於任何複雜邏輯，你都應當使用計算屬性。例如：

<div id="app">

**{{**reversedMessage**}}**

</div>

<script>

var vm = new Vue({

el: "#app",

data: {

message: 'Hello'

},

computed: {

reversedMessage: function () { (Corelab11-1)

return this.message.split('').reverse().join('');

}

}

});

</script>

* 計算屬性是基於它們的響應式依賴(reactive dependencies)進行快取的(cached)。只在相關響應式依賴發生改變時它們才會重新求值。(Corelab11-2)
* 這就意味著只要message還沒有發生改變，多次存取 reversedMessage計算屬性會立即傳回之前的計算结果，而不必再次執行函數。
* 如果你不希望有快取，請用方法來替代。例如下面的計算屬性將不再更新，因為Date.now()不是響應式依賴，所以使用方法較合理：

computed: {

now: function () {

return Date.now()

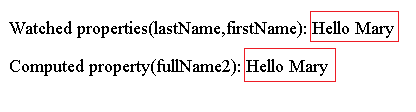
}

}

### 監聽屬性(Watched Properties)

* Vue提供了一種更通用的方式來觀察和響應(observe and react to) Vue實例上的資料變動：監聽屬性(watched properties)。
* 不過當我們有一些資料需要随著其它資料變動而變動時，通常較好的做法是使用**計算屬性**而不是命令式的watch回呼函數。
  + watched屬性初始化時，其函式不會被呼叫。
  + computed屬性初始化時，其函式則會被呼叫。
* 雖然計算屬性在大多數情況下更合適，但有時也需要監聽屬性。
  + 當需要在資料變化時執行非同步(asynchronous)或費時的操作時，這個方式是最有用的。
  + 在範例T12-12.html中，使用watch選項允許我們執行非同步操作(存取一個API)，並在我們得到最終結果前，設置中間狀態(Thinking…)。這些都是計算屬性無法做到的。

### 範例：T12-11.html(計算屬性vs監聽屬性)



<!DOCTYPE html>

<html>

<head>

(略)

</head>

<body>

<div id="app">

<p>Watched properties(lastName,firstName): **{{**fullName**}}**</p>

<p>Computed property(fullName2): **{{**fullName2**}}**</p>

</div>

<script>

var vm = new Vue({

el: "#app",

data: {

firstName: "Kitty",

lastName: "Hello",

fullName: 'Hello Kitty'

},

watch: {

firstName: function (newValue, oldValue) { (Corelab 18-3)

this.fullName = this.lastName + ' ' + newValue;

},

lastName: function ( newValue, oldValue) {

this.fullName = newValue + ' ' + this.firstName;

}

},

computed: {

fullName2: function () {

return this.lastName + " " + this.firstName;

}

}

});

window.setTimeout(function () {

vm.firstName = "Mary";

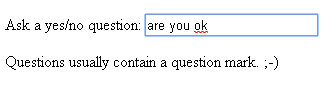
}, 2000);

</script>

</body>

</html>

### 範例：T12-12.html(監聽屬性最常應用)



<!DOCTYPE html>

<html>

(略)

<body>

<div id="app">

<p>

Ask a yes/no question:

<input v-model="question">

</p>

<p>**{{** answer **}}**</p>

</div>

<script src="https://code.jquery.com/jquery-3.4.1.min.js"></script>

<script>

var vm = new Vue({

el: '#app',

data: {

question: '',

answer: 'I cannot give you an answer until you ask a question!'

},

watch: {

// whenever question changes, this function will run

question: function (newQuestion, oldQuestion) {

this.getAnswer();

}

},

methods: {

getAnswer: function () {

if (this.question.indexOf('?') === -1) {

this.answer =

'Questions usually contain a question mark. ;-)';

return;

}

this.answer = 'Thinking...';

var vm = this;

jQuery.ajax({

method: "get",

url: "https://yesno.wtf/api",

dataType: "json",

success: function (response) {

vm.answer = response.answer;

}

});

}

}

})

</script>

</body>

</html>

## $emit

### 監聽子元件的「自定義事件」

* Vue提供了一個自定義事件(custom events)的系統，讓父元件可以透過 v-on 監聽子元件實例的任意事件。
* 子元件則可以透過呼叫内建的 $emit方法並傳入事件名稱來觸發一個事件。(Corelab 10-1)

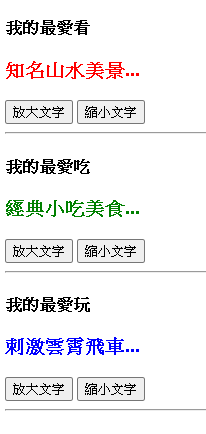
vm.$emit( eventName, […args] )

* + Trigger an event on the current instance. Any additional arguments will be passed into the listener’s callback function.
  + Arguments:
    - {string} eventName
    - [...args] (其餘參數(rest parameter)語法可以讓我們表示不確定數量的參數，並將其視為一個陣列)

### 以自定義事件發出一個值

* 有的時候用一個事件來發出一個特定的值是非常有用的。這時可以使用 $emit的第二個參數來提供這個值。
* 當在父級元件監聽這個事件的時候，我們可以透過$event存取到被發出的這個值。如果這個事件處理函數(event handler)是一個方法，那麼這個值將會作為第一個參數傳入這個方法。

### 範例：T12-21.html



<!DOCTYPE html>

<html>

<head>

(略)

</head>

<body>

<div id="app">

<div v-bind:style="{ fontSize: postFontSize + 'em' }">

<blog v-for="item in posts"

v-bind:key="item.id" v-bind:post="item"

v-on:enlarge-text="postFontSize += $event"

v-on:reduce-text="onReduceText"

>

</blog>

</div>

</div>

<script>

//註冊一個名為blog的全域的新元件

Vue.component('blog', {

props: ['post'],

template: `

<div class="blog">

<h4>{{ post.title }}</h4>

<h3 v-html="post.content"></h3>

<button v-on:click="$emit('enlarge-text', 0.1)">放大文字</button>

<button v-on:click="$emit('reduce-text', 0.1)">縮小文字</button><hr/>

</div>

`

});

new Vue({

el: "#app",

data: {

posts: [

{ id: 1, title: '我的最愛看',

content: ' <div style="color:red">知名山水美景... </div>' },

{ id: 2, title: '我的最愛吃',

content:' <div style="color:green">經典小吃美食... </div>' },

{ id: 3, title: '我的最愛玩',

content: ' <div style="color:blue">刺激雲霄飛車... </div>' }

],

postFontSize: 1

},

methods: {

onReduceText: function (reduceAmount) {

this.postFontSize -= reduceAmount;

}

}

});

</script>

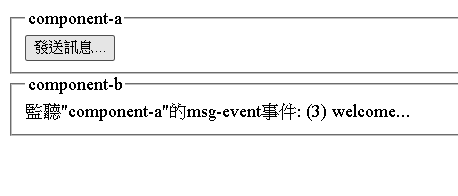
</body>

</html>

## event bus

* EventBus 主要是為了方便對等元件之間彼此溝通，互相傳遞資料，但又不需要用到vuex這麼專業的狀態管理工具時所使用的方式。
* 基本上，父子元件之間傳遞資料使用props(父對子傳遞)與$emit(子對父傳遞)。簡單的非父子關係的元件之間傳遞資料則可使用EventBus。
* Event Bus就像一個所有元件所共用的事件中心，它的使用方式簡單來說就是建立一個vue的實例來充當EventBus，當A元件要將資料發送給B元件時，可透過在Event Bus發送($emit)一個特定事件， B元件則透過事先(如mounted鈎子)向EventBus註冊監聽($on) A元件的特定事件，即可適時接收A元件所發送出來的該事件。

### 範例：T12-31.html



<!DOCTYPE html>

<html>

<head>

(略)

</head>

<body>

<div id="app">

<component-a></component-a>

<component-b></component-b>

</div>

<script>

const eventBus = new Vue();

Vue.component('component-a', {

methods: {

emitMessage: function (msg) {

eventBus.$emit('msg-event', msg);

}

},

template:

`

<fieldset>

<legend>component-a</legend>

<button v-on:click='emitMessage("welcome...")'>發送訊息...</button>

</fieldset>

`

})

Vue.component('component-b', {

data: function () {

return {

count:0,

receivedMessage:''

}

},

mounted: function () {

var thisComponent = this;

eventBus.$on('msg-event', function (msg) {

thisComponent.receivedMessage = msg;

thisComponent.count++;

})

},

template:

`

<fieldset>

<legend>component-b</legend>

監聽"component-a"的msg-event事件: ({{count}}) {{receivedMessage}}

</fieldset>

`

})

new Vue({

el: "#app"

});

</script>

</body>

</html>

# Module 13. Vue transition mode

## 對應的指令說明

### 單元素/元件的過渡(Transitioning Single Elements/Components)

* Vue在**插入**、**更新**或者**移除**DOM時，提供多種不同方式來應用過渡效果(transition effects)。包括以下工具：
  + 在CSS過渡中自動應用class。
  + 可以配合使用第三方CSS動畫庫，如Animate.css。
  + 在過渡鈎子函數中使用JavaScript直接操作DOM。
  + 可以配合使用第三方JavaScript動畫庫。
* Vue 提供了 transition的封裝元件(wrapper component)，主要應用於下列情形中，可以給任何元素添加進入/離開過渡(entering/leaving transitions)
  + - 條件渲染/Conditional rendering (使用 v-if)
    - 條件展示/Conditional display (使用 v-show)
    - 動態元件(Dynamic components)
* 當插入或删除包含在transition元件中的元素時，Vue將會做以下處理：
  + 自動嗅探目標元素是否應用了CSS過渡，如果是，在恰當的時機添加/删除CSS類別名稱。
  + 如果過渡元件提供了[JavaScript鈎子函數](https://cn.vuejs.org/v2/guide/transitions.html#JavaScript-%E9%92%A9%E5%AD%90)，這些鈎子函數將在恰當的時機被呼叫。
  + 如果没有找到JavaScript鈎子並且也没有檢測到CSS過渡，DOM操作(插入/删除)在下一幀(frame)中立即執行。(注意：此指瀏覽器逐幀動畫(frame animation)機制)
* 在進入/離開的過渡(enter/leave transitions)中，會有6個class切換。(Corelab 14-2)



* + v-enter：
    - 定義進入過渡的開始狀態。
    - 在元素被插入之前生效，在元素被插入之後的下一幀移除。
  + v-enter-active：
    - 定義進入過渡生效時的狀態。
    - 代表整個進入過渡的階段。
    - 在元素被插入之前生效，在過渡/動畫完成之後移除。
    - 這個class可以被用來定義進入過渡的過程時間，延遲和緩動曲線函數(duration, delay and easing curve)。
  + v-enter-to：
    - 定義進入過渡的結束狀態。
    - 在元素被插入之後下一幀生效 (在此同時 v-enter被移除)，在過渡/動畫完成之後移除。
  + v-leave：
    - 定義離開過渡的開始狀態。
    - 在離開過渡被觸發時立刻生效，下一幀被移除。
  + v-leave-active：
    - 定義離開過渡生效時的狀態。
    - 代表整個離開過渡的階段。
    - 在離開過渡被觸發時立刻生效，在過渡/動畫完成之後移除。
    - 這個classs可以被用來定義離開過渡的過程時間，延遲和緩動曲線函數(duration, delay and easing curve)。
  + v-leave-to：
    - 定義離開過渡的結束狀態。
    - 在離開過渡被觸發之後下一幀生效 (在此同時 v-leave 被删除)，在過渡/動畫完成之後移除。
* 對於這些在過渡中切換的class名稱來說，如果你使用一個没有名字的 <transition>，則 v- 是這些class名稱的預設前綴。如果你使用了 <transition name="my-transition">，那麼 v-enter 會替換為 my-transition-enter。
* 您可以透過appear attribute設置節點在**初始渲染**(載入DOM樹狀結構)時執行過渡效果。(Corelab 13-1)

<transition appear>

<!-- ... -->

</transition>

#### 範例：T13-11.html



<!DOCTYPE html>

<html>

<head>

(略)

<style>

.v-enter-active, .v-leave-active {

transition: opacity 1s;

}

.v-enter, .v-leave-to {

opacity: 0;

}

</style>

</head>

<body>

<div id="app" style="width:200px;margin:auto;">

<button v-on:click="show = !show">

Toggle

</button>

<transition appear>

<p v-if="show" style="border:1px solid #999;width:300px;height:100px;

text-align:center; line-height:100px;">

Welcome to Vue programming...</p>

</transition>

</div>

<script>

new Vue({

el: '#app',

data: {

show: true

}

})

</script>

</body>

</html>

#### CSS Transition附註

* CSS transitions allows you to change property values smoothly, over a given duration. 參考資料來源：<https://www.w3schools.com/css/css3_animations.asp>
* CSS Transition Properties

|  |  |
| --- | --- |
| **Property** | **Description** |
| transition | * A shorthand property for setting the four transition properties into a single property:   transition: property duration timing-function delay;   * e.g. transition: width 2s; |
| transition-delay | * Specifies a delay (in seconds) for the transition effect * Default value: 0s |
| transition-duration | * Specifies how many seconds(s) or milliseconds(ms) a transition effect takes to complete * Default value: 0s |
| transition-property | * Specifies the name of the CSS property the transition effect is for * Default value: all |
| transition-timing-function | * Specifies the speed curve of the transition effect * Default value: ease * The transition-timing-function property can have the following values:   + ease - a slow start, then fast, then end slowly   + linear - the same speed from start to end   + ease-in - a slow start   + ease-out - a slow end   + ease-in-out - a slow start and end   + cubic-bezier(n,n,n,n) - lets you define your own values in a cubic-bezier(三次貝茲曲線) function   + ease等同於 cubic-bezier(0.25, 0.1, 0.25, 1.0)   + linear等同於 cubic-bezier(0.0, 0.0, 1.0, 1.0)   + ease-in等同於 cubic-bezier(0.42, 0, 1.0, 1.0)   + ease-out等同於 cubic-bezier(0, 0, 0.58, 1.0)   + ease-in-out等同於 cubic-bezier(0.42, 0, 0.58, 1.0)   四個參數的貝茲曲線  https://developer.mozilla.org/@api/deki/files/5226/=transition-timing-function.png |

### 多個元素間互相切換的過渡(Transitioning Between Elements)

* 您可以使用 v-if/v-else對多個原生元素(raw elements)之間值進行過渡動作切換。
* 有一點需要注意：
  + 當有相同標籤名的元素切換時，需要透過 key attribute設置唯一的值來標記以讓 Vue 區分它們，否則 Vue為了效率只會替換相同標籤内部的内容。
  + 即使在技術上没有必要的情況下，給在 <transition>元件中的多個元素設置 key 是一個較好的實踐(good practice)。

<transition>

<button v-if="on" key="on" @click="on=false">

On

</button>

<button v-else="" key="off" @click="on=true">

Off

</button>

</transition>

* 這裡還有一個問題，試著點擊上面的按鈕：

在“on”按鈕和“off”按鈕的過渡中，兩個按鈕都被重新渲染了，一個離開過渡的時候另一個開始進入過渡。這是 <transition> 的預設行為–進入和離開同時發生。

* + 當兩個按鈕以絕對定位(position: absolute)，彼此重疊時，上述問題可以被解決。
  + 但是同時生效的進入和離開的過渡不能滿足所有要求，所以 Vue 提供了過渡模式(Transition Mode)： (Corelab 13-2) (Corelab 14-1)
    - in-out：

新元素先進行過渡進入(transitions in)，完成之後當前元素過渡離開(transitions out)。

* + - out-in：

當前元素先進行過渡離開(transitions out)，完成之後新元素過渡進入(transitions in)。

<transition mode="out-in">

<button v-if="on" key="on" @click="on=false">

On

</button>

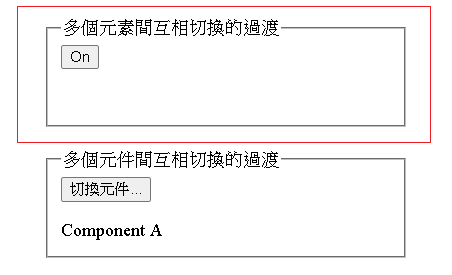
<button v-else="" key="off" @click="on=true">

Off

</button>

</transition>

#### 範例：T13-12.html (I)



<!DOCTYPE html>

<html>

<head>

(略)

<style>

.v-enter-active, .v-leave-active {

transition: opacity .5s;

}

.v-enter, .v-leave-to {

opacity: 0;

}

#app fieldset {

width:300px; height:80px; margin:20px auto;

}

</style>

</head>

<body>

<div id="app">

<fieldset>

<legend>多個元素間互相切換的過渡</legend>

<transition mode="out-in">

<button v-if="on" key="on" @click="on=false">

On

</button>

<button v-else="" key="off" @click="on=true">

Off

</button>

</transition>

</fieldset>

<fieldset>

(略)

</fieldset>

</div>

<script>

new Vue({

el: '#app',

data: {

on: true,

(略)

},

components: {

(略)

}

})

</script>

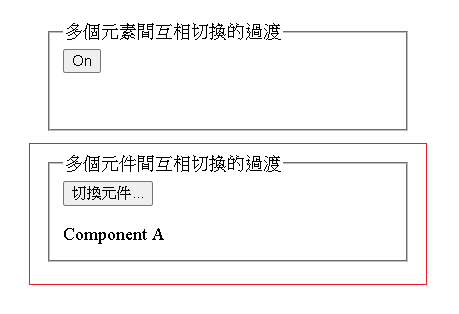
</body>

</html>

### 多個元件間互相切換的過渡(Transitioning Between Components)

* 多個元件的過度簡單很多，我們不需要使用 key attribute。相反，我們只需要使用動態元件。

#### 範例：T13-12.html (II)



<!DOCTYPE html>

<html>

<head>

(略)

<style>

.v-enter-active, .v-leave-active {

transition: opacity .5s;

}

.v-enter, .v-leave-to {

opacity: 0;

}

#app fieldset {

width:300px; height:80px; margin:20px auto;

}

</style>

</head>

<body>

<div id="app">

<fieldset>

(略)

</fieldset>

<fieldset>

<legend>多個元件間互相切換的過渡</legend>

<button key="on" @click="currentCompIndex = ++currentCompIndex%2">

切換元件...</button>

<transition mode="out-in">

<component v-bind:is="componentIds[currentCompIndex]"></component>

</transition>

</fieldset>

</div>

<script>

new Vue({

el: '#app',

data: {

(略)

currentCompIndex: 0,

componentIds: ['componentA','componentB']

},

components: {

componentA: {

template:`<p>Component A</p>`

},

componentB: {

template: `<p>Component B</p>`

}

}

})

</script>

</body>

</html>

### 可復用的過渡(Reusable Transitions)

* 過渡可以透過 Vue 的元件系統(Component System)實現復用。
* 要建立一個可復用的過渡元件，你只需要將 <transition>作為根元件，然後將任何子元件放置在其中就可以了。(Corelab 13-3)

## 自行設計CSS效果或使用第三方套件

### 自定義過渡類別(Custom Transition Classes)

* 我們可以透過以下<transition>元件的attribute來自定義過渡class名稱：
  + enter-class
  + enter-active-class
  + enter-to-class
  + leave-class
  + leave-active-class
  + leave-to-class
* 他們的優先級高於普通的class名稱，這對於Vue 的過渡系統和其他第三方 CSS動畫庫(CSS animation library)(如 [Animate.css](https://daneden.github.io/animate.css/))的結合使用十分有用。
  + Animate.css： 文件網址https://github.com/daneden/animate.css

### 範例 T13-21.html



<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>Vue Transition Mode</title>

<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>

<link href="https://cdn.jsdelivr.net/npm/animate.css@3.5.1" rel="stylesheet" type="text/css">

</head>

<body>

<div id="app">

<button v-on:click="show = !show">

切換「彈右離開」與「抖動進入」

</button>

<!--使用第三方套件(Animate.css)-->

<transition enter-active-class="animated tada"

leave-active-class="animated bounceOutRight">

<h1 v-if="show" style="animation-duration: 2s">welcome</h1>

</transition>

<!--Animate.css: https://github.com/daneden/animate.css-->

<!--Note: The animated class has a default speed of 1s. If you need custom duration,

add it directly to your own CSS code.-->

</div>

<script>

new Vue({

el: '#app',

data: {

show: true

}

})

</script>

</body>

</html>

## 透過JavaScript操作

* 我們可以在transition的屬性(attributes)中定義JavaScript鈎子函數(JavaScript Hooks)。常用者如下：

<transition

v-on:before-enter="beforeEnter"

v-on:enter="enter"

v-on:after-enter="afterEnter"

v-on:before-leave="beforeLeave"

v-on:leave="leave"

v-on:after-leave="afterLeave"

<!-- ... -->

</transition>

new Vue({

el: '#app',

data: {

show: false

},

methods: {

beforeEnter: function (el) { },

enter: function (el, done) {

//...

done();//將觸發afterEnter被呼叫

},

afterEnter: function (el) { },

beforeLeave: function (el) { },

leave: function (el, done) {

//...

done();//將觸發afterLeave被呼叫

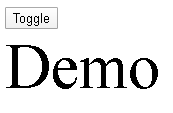
},

afterLeave: function (el) { }

}

});

### 範例：T13-31.html



<!DOCTYPE html>

<html>

<head>

(略)

<script src="javascripts/jquery-3.4.1/jquery-3.4.1.min.js"></script>

<script src="javascripts/jquery ui 1.12.1/jquery-ui.min.js"></script>

</head>

<body>

<div id="app">

<button @click="show = !show">

Toggle

</button>

<transition v-on:before-enter="beforeEnter"

v-on:enter="enter"

v-on:leave="leave"

v-bind:css="false"> <!-Vue可省去偵測CSS Transition的動作-->

<div v-if="show">Demo</div>

</transition>

</div>

<script>

new Vue({

el: '#app',

data: {

show: false

},

methods: {

beforeEnter: function (el) {

$(el).css({ fontSize: '1em', opacity: 0});

},

enter: function (el, done) {

$(el).animate({ opacity: 1, fontSize: '4em' }, 1000, 'easeOutBounce');

},

leave: function (el, done) {

$(el).animate({ opacity: 0, fontSize: '1em' }, 600, 'easeInSine');

}

}

});

</script>

</body>

</html>

# Module 14. Vue Router

## 配置流程

### Vue Router導論

* Vue Router 是Vue.js官方的路由管理器。它和Vue.js的核心可以深度整合。
* 使用Vue.js，我們已經可以將元件組合成應用程式。若要把Vue Router添加進來，我們只需要將元件映射到路由(map our components to the routes)，然後告訴Vue Router這個映射資訊。
* Vue Router配置流程基本上分成四步驟：
  + 定義元件(Define components)

const PageContactInfo = {

template: '<p>This is the contact info page</p>'

};

* + 定義路由(Define routes)

var routes = [

{ path: '/contact-info', component: PageContactInfo },

{ path: '/about', component: PageAbout }

];

|  |
| --- |
| \*\* Vue Router 1.0的舊方法已不再支援：\*\*  router.map({  '/a': { component: {} },  '/b': { component: {} }  })  (Corelab 15-1) |

* + 建立路由器實例(the router instance)並傳入`routes`選項

const router = new VueRouter({

routes: routes

});

* + 建立/掛載Vue根實例並傳入`router`選項

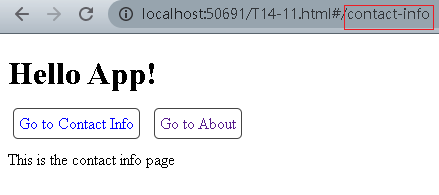
new Vue({

el: '#app',

router: router,

});

#### 範例T14-11.html



<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<title>Vue Router</title>

<script src="javascripts/vue 2.6.11/vue.js"></script>

<script src="javascripts/vue-router 3.0/vue-router.min.js"></script> (Corelab 14-3)

<!--<script src="https://unpkg.com/vue-router/dist/vue-router.js"></script>-->

<style>

.router-link-active {

color: blue;

/\* A <router-link> automatically gets the .router-link-active class when its target route is matched.\*/

}

.a-button {

border: 1px solid #444;

border-radius: 5px;

text-decoration: none;

padding: 5px;

margin: 5px;

}

</style>

</head>

<body>

<div id="app">

<h1>Hello App!</h1>

<p>

<!-- use router-link component for navigation. -->

<!-- specify the link by passing the `to` prop. -->

<!-- `<router-link>` will be rendered as an `<a>` tag by default -->

<router-link class="a-button" to="/contact-info">Go to Contact Info</router-link>

<router-link class="a-button" to="/about">Go to About</router-link>

|  |
| --- |
| * <router-link>元件支援用戶在具有路由功能的應用程式中(點擊)導航。 (Corelab 16-2) * <router-link>元件的目標位址可透過to屬性(prop)來指定，該元件預設會渲染成帶有正確href的<a>標籤。 |

</p>

<!-- route outlet -->

<!-- component matched by the route will render here -->

<router-view></router-view>

</div>

<script>

// 1. Define components

const PageContactInfo = {

template: '<p>This is the contact info page</p>'

};

const PageAbout = {

template: '<p>This is the about page</p>'

};

// 2. Define routes

// Each route should map to a component.

var routes = [

{ path: '/contact-info', component: PageContactInfo },

{ path: '/about', component: PageAbout }

];

// 3. Create the router instance and pass the `routes` option

const router = new VueRouter({

routes: routes

});

// 4. Create and mount the root Vue instance.

// Make sure to inject the router with the router option to make the whole app router-aware.

// By injecting the router, we get access to it as this.$router as well as the current route as this.$route inside of any component.

new Vue({

el: '#app',

router: router,

});

</script>

</body>

</html>

* 具名路由(Named Routes)
  + 有時候透過一個名稱來識別一個路由顯得更方便一些，特别是在連結(linking)一個路由的時候。
  + 你可以在建立Router實例的時候，在routes選項中給某個路由設定名稱。

const router = new VueRouter({

routes: [

{

path: '/contact-info',

name: 'r-contact',

component: {}

}

]

})

* + 要連結(link)到一個具名路由，可以給router-link元件的to屬性(prop)傳一個物件：

<router-link :to="{ name: 'r-contact'}">User</router-link>

* 重新導向(Redirecting)也是透過routes 配置(routes configuration)來完成的，下面例子是從 /a 重新導向 /b：

const router = new VueRouter({

routes: [

{ path: '/a', redirect: '/b' } (Corelab 15-3)

]

})

### 巢狀路由(Nested Routes)

* 實際上應用程式的使用者介面通常由多層巢狀的元件組合而成。URL中的區段(segments)通常也對應著特定的巢狀元件(nested components)的結構，
* 例如：

/user/foo/profile /user/foo/posts

+------------------+ +-----------------+

| User | | User |

| +--------------+ | | +-------------+ |

| | Profile | | +------------> | | Posts | |

| | | | | | | |

| +--------------+ | | +-------------+ |

+------------------+ +-----------------+

* 借助 vue-router，使用巢狀路由配置(nested route configurations)，就可以很簡單地表達這種關係。

#### 範例T14-12.html



<!DOCTYPE html>

<html>

<head>

(略)

<style>

.router-link-active {

color: blue;

/\*Notice that a <router-link> automatically gets the .router-link-active class when its target route is matched.\*/

}

.a-button {

border:1px solid #444;

border-radius:5px;

text-decoration:none;

padding:5px;

margin:5px;

}

.parent-box {

width:300px;

height:200px;

margin:20px;

padding:20px;

box-shadow:5px 5px 10px #444;

}

.child-box {

border:1px solid #999;

width:80%;

height:50%;

margin:auto;

}

</style>

</head>

<body>

<div id="app">

<h1>Hello App!</h1>

<p>

<!-- use router-link component for navigation. -->

<!-- specify the link by passing the `to` prop. -->

<!-- `<router-link>` will be rendered as an `<a>` tag by default -->

<router-link class="a-button" to="/user/Kitty">Go to Kitty 首頁</router-link>

<router-link class="a-button" to="/user/Kitty/profile">Go to Kitty profile</router-link>

<router-link class="a-button" to="/user/Kitty/posts">Go to Kitty posts</router-link>

</p>

<!-- route outlet -->

<!-- component matched by the route will render here -->

<router-view></router-view>

<!-- The <router-view> here is a top-level outlet. It renders the component matched by a top level route.

Similarly, a rendered component can also contain its own, nested <router-view>. -->

* 這裡的 <router-view> 是最頂層的出口，匹配到最頂層路由的元件會在此被渲染。
* 同樣地，一個被渲染的元件同樣可以包含自己的巢狀的 <router-view>。
* 例如，在PageUser元件的模板裡添加一個 <router-view>(如下)：

</div>

<script>

// 1. Define components

const PageUser = {

template:

`<div class="parent-box">

<h2>Hello, kitty 您好!</h2>

<router-view></router-view>

</div>

`

};

const UserHome = { template: '<div class="child-box">Home</div>' }

const UserProfile = { template: '<div class="child-box">Profile</div>' }

const UserPosts = { template: '<div class="child-box">Posts</div>' }

* 要在巢狀的出口中渲染元件，需要在VueRouter建構子組態(constructor config)中使用 children選項： (Corelab 16-1)

// 2. Define routes

// Each route should map to a component.

var routes = [

{

path: '/user/kitty',

component: PageUser,

children: [

{

// UserHome will be rendered inside PageUser's <router-view>

// when /user/kitty is matched

path: '',

component: UserHome

},

{

// UserProfile will be rendered inside PageUser's <router-view>

// when /user/kitty/profile is matched

path: 'profile',

component: UserProfile

},

{

// UserPosts will be rendered inside PageUser's <router-view>

// when /user/kitty/posts is matched

path: 'posts',

component: UserPosts

}

]

}

];

// 3. Create the router instance and pass the `routes` option

const router = new VueRouter({

routes //ES6 Shorthand property names(in Object literal/Initializer)

});

// 4. Create and mount the root Vue instance.

// Make sure to inject the router with the router option to make the whole app router-aware.

// By injecting the router, we get access to it as this.$router as well as the current route as

this.$route inside of any component.

new Vue({

el: '#app',

router //ES6 Shorthand property names(in Object literal/Initializer)

});

</script>

</body>

</html>

## 動態路由

### 動態路由匹配(Dynamic Route Matching)

* 我們經常需要把某種模式(pattern)的路由都映射到同一個元件。
* 例如我們有一個User元件，對於ID不相同的所有用戶都要使用這個元件來渲染。那麼我們可以在vue-router的路由路徑使用“動態區段/路由參數”(dynamic segment) 來達到這個效果。
  + 一個“動態區段”使用冒號 **:** 標記。
  + 當匹配到一個路由時，參數值會被設定到this.$route.params，可以在每個元件内使用。
* 你可以在一個路由中設定置多段 “動態區段/路由參數”，對應的值都會設置到 $route.params 中。例如：

| **模式** | **匹配路径** | **$route.params** |
| --- | --- | --- |
| /user/:username  (Corelab 15-2) | /user/evan | { username: 'evan' } |
| /user/:username/post/:postid | /user/evan/post/123 | { username: 'evan', postid: '123' } |

* 有時候同一個路徑可以匹配多個路由，此時匹配的優先順序就按照路由定義的順序：先被定義的路由具有較高的優先順序。

### 捕獲所有路由或404 Not found路由

* 基本上參數只會匹配以 / 分隔的URL片段( URL fragments)中的字串。
* 如果想匹配任意路徑，我們可以使用\* (asterisk)：

{

// 會匹配所有路徑

path: '\*'

}

{

// 會匹配以 `/user-` 開頭的任意路徑

path: '/user-\*'

}

* 當使用\* (asterisk)路由時，請確保路由的順序是正確的，也就是說含有\*的路由應該放在最後。路由 { path: '\*' } 通常用於客户端 404 錯誤。

### 範例T14-21.html



<!DOCTYPE html>

<html>

<head>

(略)

</head>

<body>

<div id="app">

<h1>Hello App!</h1>

<p>

<router-link class="a-button" to="/user/kitty">

Go to kitty 首頁</router-link>

<router-link class="a-button" to="/user/mary">

Go to mary 首頁</router-link>

<router-link class="a-button" to="/user/peter">

Go to peter 首頁</router-link>

</p>

<router-view></router-view>

</div>

<script>

// 1. Define components.

const PageUser = {

template:

`<div class="user">

<h2>哈囉, {{$route.params.id}} 您好!</h2>

<router-view></router-view>

</div>

`

};

// 2. Define routes

// Each route should map to a component.

var routes = [

{

path: '/user/:id',

component: PageUser

},

{

path: '/',

component: {template:`<h5 style='color:blue'>

請點選上面按鈕以瀏覽相關葉面...</h5>`}

},

{

path: '\*',

component: { template: `<h5 style='color:red'>

您請求的頁面不存在(Not Found)...</h5>` }

},

];

// 3. Create the router instance and pass the `routes` option

const router = new VueRouter({

routes //ES6 Shorthand property names(in Object literal/Initializer)

});

// 4. Create and mount the root Vue instance.

var vm = new Vue({

el: '#app',

router //ES6 Shorthand property names(in Object literal/Initializer)

});

</script>

</body>

</html>

## Navigation Guards

### 導航守衛導論

* 「導航」表示路由正在發生改變。
* 正如其名，vue-router 提供的導航守衛主要用來透過重新導向或取消(redirecting or canceling)的方式守衛導航。
* 路由導航過程(the route navigation process)中可以植入鈎子函式，其中較常用者有全域前置守衛(Global Before Guards)以及元件內守衛(In-Component Guards)。

#### 全域前置守衛(Global Before Guards)

* 你可以使用 router.beforeEach 註冊一個全域前置守衛：

const router = new VueRouter({ ... });

router.beforeEach(to, from, next) => {

// ...

});

* 當一個導航觸發時，全域前置守衛按照建立的順序被呼叫。守衛會被異步地解決(resolved asynchronously)，此時導航在所有守衛被解析完之前一直處於等待(pending)中。
* 每個守衛方法接收三個參數：
* to: Route: 即將要進入的目標路由物件(the target Route Object being navigated to)
* from: Route: 當前導航正要離開的路由(the current route being navigated away from)
* next: Function: 一定要呼叫該方法來解決這個鈎子。執行效果視提供給 next方法的引數(arguments)而定。
  + - next(): 進行管道(pipeline)中的下一個鈎子。如果全部鈎子執行完了，則導航的狀態就是confirmed (確認的)。
    - next(false): 終止(abort)當前的導航。如果瀏覽器的URL改變了(可能是用戶手動或者瀏覽器後退按鈕)，那麼 URL 網址會重置(reset)到from路由對應的網址。
    - next('/') 或 next({ path: '/' }) 或 next({name: '路由名稱'})等: 重新導向到一個不同的網址。當前的導航被中斷，然後進行一個新的導航。
    - next(error): 如果傳入 next 的參數是一個 Error實例，則導航會被終止(aborted)且該錯誤會被傳遞給經 router.onError( callback )註冊過的回呼函式(callbacks)。

#### 元件內的守衛(In-Component Guards)

* 你可以在路由元件(route components: the ones passed to the router configuration)内直接定義以下路由導航守衛：
  + beforeRouteEnter
  + beforeRouteUpdate
  + beforeRouteLeave

const Foo = {

template: `...`,

beforeRouteEnter (to, from, next) {

//功能類似Router.beforeEach()全域前置守衛(除了有效範圍僅限相關聯的路由而已)

},

beforeRouteUpdate (to, from, next) {

//常常用來監聽“路由參數”的變更

},

beforeRouteLeave (to, from, next) {

//通常用來禁止用戶在還未保存修改前突然離開。這個導航可以透過next(false)來取消。

//例如

const answer = confirm('Do you really want to leave?')

if (answer) {

next()

} else {

next(false)

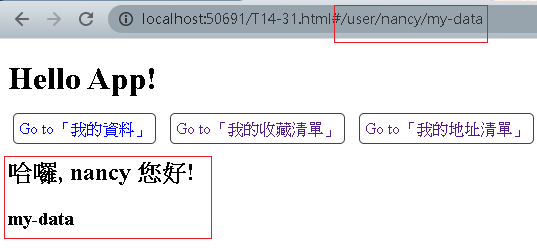
}

}

### 範例T14-31.html



\*\*\*若使用者尚未通過身分驗證，其請求就會被重新導向「登入畫面」\*\*\*



\*\*\*使用者通過身分驗證後，會被重新導向「我的資料」頁面\*\*\*

(改編T14-21.html)

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<title>Vue Router</title>

<script src="javascripts/vue 2.6.11/vue.js"></script>

<script src="javascripts/vue-router 3.0/vue-router.min.js"></script>

<script src="javascripts/jquery-3.4.1/jquery-3.4.1.min.js"></script>

<style>

(略)

</style>

<template id="template1">

<div style="border:1px solid #444; margin: 10px 0; padding:20px; width:250px;">

<form method="get" action="AjaxServer.aspx" id="form1">

<p>

<label>

UserName: <input type='text' name="username" value="nancy"/>

</label>

</p>

<p>

<label>

Password: <input type='password' name="password" value="davolio"/>

</label>

</p>

<p>

<input type='submit' />

</p>

</form>

</div>

</template>

</head>

<body>

<div id="app">

<h1>Hello App!</h1>

<p>

<router-link class="a-button" :to="myDataPath">Go to「我的資料」</router-link>

<router-link class="a-button" :to="myCollectionListPath">

Go to「我的收藏清單」</router-link>

<router-link class="a-button" :to="myAddressListPath">

Go to「我的地址清單」</router-link>

</p>

<keep-alive>

<router-view></router-view>

</keep-alive>

</div>

<script>

// 1. Define components

const PageUser = {

template:

`<div>

<h2>哈囉, {{$route.params.id}} 您好!</h2>

<h3>{{$route.params.section}}</h3>

</div>

`,

beforeRouteUpdate: function (to, from, next) {

//\*\*\*\*\*\*\*使用beforeRouteUpdate元件內導航守衛來監聽“路由參數”的變更\*\*\*\*\*\*\*//

if (isAuthenticated && (to.params.id != from.params.id) ) )//已通過身分驗證者

若變更網址列上的路由參數id，則將之重新導向至登入畫面

next('/login'); //重新導向

else

next(); //放行

}

};

const PageLogin = {

template: "#template1"

};

// 2. Define routes

// Each route should map to a component.

var routes = [

{

path: '/user/:id/:section',

component: PageUser

},

{

path: '/login',

component: PageLogin

},

{

path: '/',

component: {

template: `<h5 style='color:blue'>請點選上面按鈕以瀏覽相關頁面...</h5>` }

},

{

path: '\*',

component: {

template: `<h5 style='color:red'>您請求的頁面不存在(Not Found)...</h5>` }

}

];

// 3. Create the router instance and pass the `routes` option

const router = new VueRouter({

routes //ES6 Shorthand property names(in Object literal/Initializer)

});

var isAuthenticated = false;

//\*\*\*\*\*\*\*註冊一個全域前置守衛\*\*\*\*\*\*\*//

router.beforeEach(function (to, from, next) { //!isAuthenticated表示尚未通過身分驗證

if (to.path !== '/login' && !isAuthenticated)

next('/login'); //重新導向

else

next(); //放行

});

// 4. Create and mount the root Vue instance.

var vm = new Vue({

el: '#app',

data: {

myDataPath: '/',

myCollectionListPath: '/',

myAddressListPath: '/'

},

router, //ES6 Shorthand property names(in Object literal/Initializer)

mounted: function () {

//註冊使用者所填的登入資料以AJAX發送給伺服器

$("#form1").submit(function (){

$.ajax({

method: 'get',

url: $(this).attr("action"),

data: $(this).serialize(),

dataType:'json',

success: function (data) {

if (data.status == 'pass') {//通過身分驗證

isAuthenticated = true;

vm.myDataPath = `/user/${data.username}/my-data`;

vm.myCollectionListPath =

`/user/${data.username}/my-collection-list`;

vm.myAddressListPath =

`/user/${data.username}/my-address-list`;

$("#form1 p:last").append(

"<span id='pass-msg' style='color:blue;'>

恭喜您通過身分驗證!</span>");

setTimeout(function () {

$("#pass-msg").remove();

vm.$router.push(`/user/${data.username}/my-data`);

//相當於使用者點擊<router-link :to="...">

}, 2000);

}

}

});

return false;

});

},

});

</script>

</body>

</html>

AjaxServer.aspx

<%@ Page Language="C#" ContentType="text/html" %>

<%

if (Request["username"] == "nancy" && Request["password"] == "davolio")

Response.Write("{\"status\":\"pass\", \"username\":\"" + Request["username"] +"\"}");

//例如: {"status":"pass", "username":"nancy"}

else

Response.Write("{\"status\":\"fail\"}");

//例如: {"status":"fail"}

%>

# Module 15. Vuex

## 基本結構

### Vuex是什麼？

* Vuex是一個專為Vue.js應用程式開發的狀態管理模式+函式庫(state management pattern + library)。 (Corelab 16-3)
* Vuex的角色是作為應用程式所有元件的集中狀態儲存區(a centralized store)，它保證狀態只能以可預測的方式被更改。
* Vuex狀態管理模式示意圖
  + 透過定義和分隔狀態管理中的各種概念，並強制一些規則來維持視圖與狀態間的獨立性，Vuex的程式碼將會變得更結構化且易維護。



### 什麼情况下我應該使用Vuex？

* Vuex 可以幫助我們管理共享狀態(shared state management)，但也附帶了更多的概念和框架的學習成本。這需要對短期和長期效益進行權衡。
* 如果您不打算開發大型**單頁網頁(SPA)**，使用Vuex可能是繁瑣冗餘的。確實是如此—如果您的應用程式夠簡單，您最好不要使用Vuex。一個簡單的store模式(store pattern)就足够您所需了。
* 但是，如果您需要建構一個中大型單頁應用程式(a medium-to-large-scale SPA)，您很可能會考慮如何更好地在元件外部管理狀態，Vuex將會成為自然而然的選擇。

## 狀態管理模式

### Vuex狀態管理的核心：Store

* 每一個Vuex應用程式的核心就是store。“store”基本上就是一個容器，它包含著你的應用程式中大部分的狀態(state)。
* Vuex和單純的全域物件(a plain global object)有以下兩點不同：
  + Vuex stores是響應式的(reactive)。當Vue元件從store中讀取狀態的時候，若store中的狀態發生變化，那麼該元件也會相應地得到高效的更新。
  + 你不能直接改變store中的狀態。改變store中的狀態的唯一途徑就是明確地(explicitly)確認(commit) mutations。這樣使得我們可以方便地跟踪每一個狀態的變化。

(Corelab 17-1)

* 網頁應用程式使用Vuex的基本步驟
  + 安裝Vuex

<!--<script src="javascripts/vuex 3.3.0/vuex.js"></script>-->

<script src="https://unpkg.com/vuex@3.3.0"></script>

* + 建立一個store，並提供一個初始state物件和一些mutations：

const store = new Vuex.Store({

state: {

count: 0

},

mutations: {

increment: state => state.count++,

decrement: state => state.count--

}

});

* + 現在，你可以在元件中
    - 透過store.state來攫取狀態物件。
      * 由於store中的狀態是響應式的，在元件中使用store中的狀態簡單到僅需要在計算屬性(computed property)中傳回該狀態即可。
    - 透過store.commit方法觸發狀態變更。
      * 觸發變化也僅僅是在元件的methods中確認(commit) mutation。

new Vue({

el: '#app',

computed: {

count: function () {

return store.state.count;

}

},

methods: {

incr() {

store.commit('increment');

},

decr() {

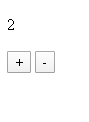
store.commit('decrement');

}

}

});

### 範例 T15-21.html



<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<title>Vuex</title>

<script src="javascripts/vue 2.6.11/vue.js"></script>

<script src=" javascripts/vuex 3.3.0/vuex.js"></script>

<!--<script src="https://unpkg.com/vuex@3.3.0"></script>-->

</head>

<body>

<!-- view -->

<div id="app">

<p>**{{** count **}}**</p>

<p>

<button @click="incr">+</button>

<button @click="decr">-</button>

</p>

</div>

<script>

const store = new Vuex.Store({

state: {

count: 0

},

mutations: {

increment: function (state) { state.count++; },

decrement: function (state) { state.count--; }

}

});

new Vue({

el: '#app',

computed: {

count: function () {

return store.state.count;

}

},

methods: {

incr() {

store.commit('increment');

},

decr() {

store.commit('decrement');

}

}

});

</script>

</body>

</html>

## 跨元件間的溝通

### 使用this.$store

* 為了在Vue元件中存取this.$store屬性(property)，你需要為Vue實例提供已建立的store。
* Vuex提供了一個機制讓你以store選項的方式將store從根元件“注入”所有子元件。

const **store** = new Vuex.Store({

state: { },

mutations: {

});

new Vue({

el: '#app',

store: **store**

})

* 現在我們可以從元件的方法確認(Commit)一個變更(Mutation)：

methods: {

increment() {

this.$store.commit('increment')

console.log(this.$store.state.count)

}

}

### 範例 T15-31.html



<!DOCTYPE html>

<html>

<head>

(略)

</head>

<body>

<div id="app">

<p>**{{** count **}}**</p>

<p>**{{** message('您點擊的按鈕次數: ') **}}**</p>

<p>

<button @click="incr">+</button>

<button @click="decr">-</button>

<button @click="incr2(2)">+2</button>

<button @click="decr2(2)">-2</button>

</p>

<message-box></message-box>

</div>

<script>

const **store** = new Vuex.Store({

state: {

count: 0

},

getters: {

count: function (state, getters) {

return state.count; //元件可以用屬性形式(store.getters.count)存取本getter

},

msg: function (state, getters) {

return function (prefix) { return prefix + state.count }; //元件可以用方法形式

(store.getters.msg('...'))存取本getter

}

},

mutations: {

increment: function (state) { state.count++; },

decrement: function (state) { state.count-- };

//------------Mutation with payload(載荷)------------//

increment2: function (state, amount) { state.count += amount; },

decrement2: function (state, amount) { state.count -= amount;}

},

actions: {

increment: function (context) {

//非同步操作…

**context.commit('increment')**;

},

decrement: function (context) {

//非同步操作…

**context.commit('decrement');**

}

//------------Action with payload(載荷)------------//

increment2: function (context, amount) {

//非同步操作…

**context.commit('increment2', amount);**

},

decrement2: function (context, amount) {

//非同步操作…

**context.commit('decrement2', amount);**

}

}

});

const MessageBox = {

template: `<h1 @click='increment'>{{count}}</h1>`,

computed: {

...Vuex.mapGetters(['count' ]) //展開後如下:

//count: function () {

// return this.$store.getters.count;

//}

//註: ...為展開運算子(Spread operator)

//ES2018支援Spread in object literals

//例如: { ...{ prop1: 'xxx', prop2: 'yyy' } } 結果為 { prop1: 'xxx', prop2: 'yyy' }

},

methods: {

...Vuex.mapMutations(['increment'])//展開後如下:

//increment: function() {

// this.$store.commit('increment');

//}

}

};

new Vue({

el: '#app',

computed: {

...Vuex.mapGetters(['count' ]) //展開後如下:

//count: function () {

// return this.$store.getters.count;

//},

...Vuex.mapGetters( {'message' : 'msg'} ) //展開後如下:

//message: function () {

// return this.$store.getters.msg;

//}

},

methods: {

...**Vuex.mapMutations**({

incr: 'increment', decr: 'decrement',

incr2: 'increment2', decr2: 'decrement2'

}), //展開後如下:

//incr: function() {

// this.$store.commit('increment');

//},

//decr: function() {

// this.$store.commit('decrement');

//},

//incr2: function (amount) {

// this.$store.commit('increment2', amount);//Mutation with payload(載荷)

//},

//decr2: function (amount) {

// this.$store.commit('decrement2', amount); //Mutation with payload(載荷)

//},

//...**Vuex.mapActions**({

// incr: 'increment', decr: 'decrement',

// incr2: 'increment2', decr2: 'decrement2'

//}),//展開後如下:

//incr: function() {

// this.$store.dispatch('increment');

//},

//decr: function() {

// this.$store.dispatch('decrement');

//},

//incr2: function (amount) {

// this.$store.dispatch('increment2', amount);//Action with payload(載荷)

//},

//decr2: function (amount) {

// this.$store.dispatch('decrement2', amount); //Action with payload(載荷)

//},

},

components: {

MessageBox: MessageBox

},

**store: store**,

created: function () {//for測試

console.log(this.$store.getters.count); //0

console.log(this.count); //0

console.log(this.$store.getters.msg('您點擊的按鈕次數: ')); //您點擊的按鈕次數: 0

console.log(this.message('您點擊的按鈕次數: ')); //您點擊的按鈕次數: 0

}

});

</script>

</body>

</html>

### Store中的Getters

* 有時候我們需要從store中的state中計算出一些衍生的狀態(to compute derived state)。
* Vuex允許我們在store中定義“getter”（可以當成是store的計算屬性）。就像計算屬性一樣，getter的返回值會根據它的依賴被快取(cached)起來，且只有當它的依賴值發生了改變才會被重新計算。

#### 在Store中定義Getters

* Getter接受state作為其第一個參數，接受其他getters作為第二個參數：

const store = new Vuex.Store({

state: {

count: 0

},

getters: {

count: function (state, getters) {

return state.count; //元件可以用屬性形式(store.getters.count)存取本getter

},

msg: function (state, getters) {

return function (prefix) { return prefix + state.count };//元件可以用方法形式

(store.getters.msg('...'))存取本getter

}

}

});

#### 在元件中存取Getters

* 以**屬性**形式存取(Property-Style Access)
  + Getters可以透過store.getters物件來取得，然後你可以以屬性的形式存取個別的Getter值。
  + 例如：

created: function () {

console.log(this.$store.getters.**count**); //0

}

* + 注意：透過屬性形式所存取的getters會被快取(cached) as part of Vue's reactivity system。
* 以**方法**形式存取(Method-Style Access)
  + 你也可以透過讓getter傳回一個函數，來實現傳遞引數給getter。這在對store裡的陣列進行查詢時非常有用。
  + 例如：

created: function () {

console.log(this.$store.getters.**msg**('您點擊的按鈕次數: ')); //您點擊的按鈕次數: 0

}

* + 注意：透過方法形式所存取的getters不會被快取(cached)，亦即該getter在每次被存取時，對應的方法都會被執行一次。

#### 映射元件的計算屬性到Getters

* Vuex.mapGetters()函數可將元件的計算屬性(computed properties) 映射(maps)到store中的getters（需要先在根Vue節點注入store）。

new Vue({

el: '#app',

computed: {

**...**Vuex.mapGetters(['count'])

//展開後如下:

//count: function () {

// return this.$store.getters.count;

//}

//...為展開運算子(Spread operator)

//ES2018支援Spread in object literals

//{ ...{prop1:'xxx', prop2:'yyy'} } 結果為 {prop1:'xxx', prop2:'yyy'}

},

store: store,

created: function () {

console.log(this.count); //0

}

});

* 如果你想將一個getter屬性另取一個名字，使用物件形式：

new Vue({

el: '#app',

computed: {

**...**Vuex.mapGetters({'message' : 'msg' })

//展開後如下:

//message: function () {

// return this.$store.getters.msg;

//}

},

store: store,

created: function () {

console.log(this.message('您點擊的按鈕次數: ')); //您點擊的按鈕次數: 0

}

};

### Store中的Mutations

#### 在Store中定義Mutations

* 每個mutation都有一個字串的類型(type)和一個處理函數(handler)。這個處理函數就是我們實際進行狀態更改的地方，並且它會接受state作為第一個引數：

const store = new Vuex.Store({

state: {

count: 0

},

mutations: { // Mutation必須是同步函數

**increment** : function(state) {

state.count++;

}

}

});

* 具載荷(payload)的mutation
  + 你可以向store.commit傳入額外的引數，即mutation的載荷(payload)。

mutations: {

**increment2**: function(state, amount) {

state.count += amount;

}

}

#### 在元件中確認Mutations

* 更改Vuex的store中的狀態的唯一方法是確認改變(committing a mutation)。
* 你不能直接呼叫mutation handler。要呼叫一個mutation handler，你需要以相應的type呼叫 store.commit 方法：

store.commit('increment')

store.commit('**increment2**', 2)

#### 映射元件的方法到Mutations

* 你可以使用Vuex.mapMutations()函數將**元件中的methods**映射(map)為store.commit()方法呼叫（需要先在根Vue節點注入store）。

methods: {

...Vuex.mapMutations([

'increment', // map `this.increment()` to `this.$store.commit('increment')`

'increment2' // map `this.increment2(amount)` to `this.$store.commit('increment2', amount)`

]),

...Vuex.mapMutations({ incr: 'increment' }) // map `this.incr()` to `this.$store.commit('increment')`

}

### Store中的Actions

* Action類似mutation，不同之處在於：
  + Action確認(commit) mutations，而不是直接變更state。
  + Action可以包含任意非同步操作(asynchronous operations)。

#### 在Store中定義Actions

* Action處理函數(Action handlers)接受一個與store實例具有相同方法和屬性的context物件，因此你可以
  + 呼叫context.commit 來確認(commit)一個mutation
  + 或者透過 context.state 和 context.getters 來獲取 state 和 getters。

const store = new Vuex.Store({

state: {

count: 0

},

mutations: {

increment : function(state) {

state.count++;

}

},

actions: {

increment: function (context) {

context.commit('increment');

}

}

})

#### 在元件中分發Actions

* 你可以在**元件**中使用 this.$store.**dispatch**('action~') 分發actions

#### 映射元件的方法到Actions

* 你也可使用 Vuex.mapActions()函數將**元件的methods**映射為 store.dispatch()方法呼叫（需要先在根Vue節點注入store）。
  + 乍看感覺多此一舉，我們直接dispatch(分發) mutation 豈不更方便？
  + 實際上並非如此，因為 mutation 必須同步執行，Action就不受約束，我們可以在action内部執行非同步操作。

methods: {

...Vuex.mapActions([

'increment', // map `this.increment()` to `this.$store.dispatch('increment')`

'increment2' // map `this.increment2(amount)` to `this.$store.dispatch('increment2', amount)`

]),

...Vuex.mapActions({

incr: 'increment' // map `this.incr()` to `this.$store.dispatch('increment')`

})

}}

# Module 16. Vue.Cli v3 建立SPA專案

* Vue CLI (Command Line Interface)是一個支援快速開發Vue.js應用程式的完整系统。 (Corelab 12-2)
* Vue CLI包含幾個重要的套件，其中 @vue/cli 是一個全域安裝的npm套件，提供了許多vue命令。

## 透過 npm 安裝 @vue/cli，然後以vue create指令建立專案（文字模式）

### 安裝@vue/cli

* 下載/安裝Node.js (Vue CLI 需要 Node.js 8.9 或更高版本)

Node.js Windows Installer的檔案名稱如 node-v12.16.3-x64.msi。

* 開啟命令提示字元視窗，輸入下列指令

C:\Users\->npm install -g @vue/cli

* 檢查Vue的版本是否正確

C:\Users\->vue --version

@vue/cli 4.3.1 (左列為執行結果例子)

### 以vue create指令建立一個新專案

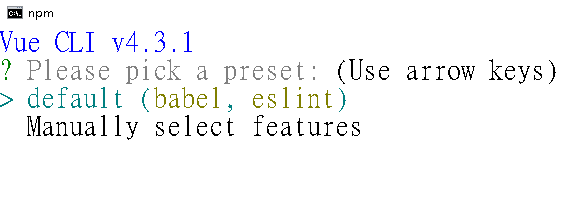
* 開啟命令提示字元視窗，變更目錄至C:\\_vue\_misc\cli\_projects

C:\Users\->cd C:\\_vue\_misc\cli\_projects

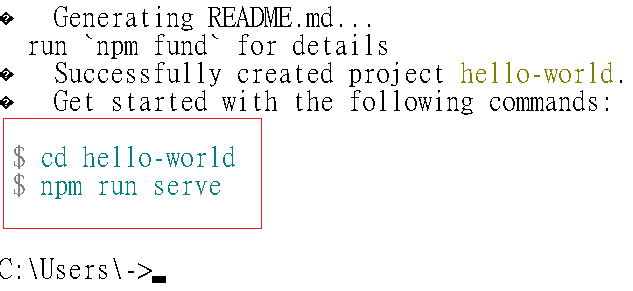
* 輸入下列指令即可建立一個新專案(專案名為hello-world)

C:\\_vue\_misc\cli\_projects>vue create hello-world

* + 你會被提示選取一個preset。
  + 你可以選取包含了基本的 Babel + ESlint 設定的預設的preset，也可以選取“手動選擇特性”來選取需要的特性。
    - Babel (又名Babel.js)是一個用於web開發，且自由開源的JavaScript編譯器。
    - ESLint是一種靜態程式碼分析工具(static code analysis tool)，用於識別在JavaScript程式碼中發現的有問題的模式。



* + 這個預設的設定非常適合快速建立一個新專案的原型(prototyping)，而手動設定則提供了更多的選項。
  + hello-world專案建立完成後會出現下列畫面，等候我們進行後續的專案測試或建置(build)。



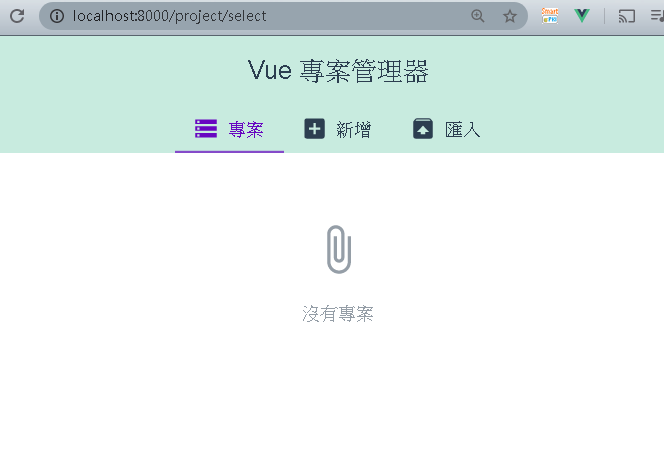
## 透過 vue ui指令啟動「Vue專案管理器」，然後以GUI介面建立專案（網頁模式）

### 以vue ui 指令啟動「Vue專案管理器」

* 開啟命令提示字元視窗，輸入下列指令

C:\Users\->vue ui

### 以GUI介面建立一個新專案



## 巡覽建立後資料夾內容並測試

### 專案目錄結構

hello-world/ # 專案目錄

│

├── dist/ # 使用npm run build指令進行建置後所產生的檔案的存放目錄

│

├── node\_modules/ # 專案所依賴的Node.js模組

│

├── public/ # 靜態檔案(如.html、.css、.js、.json、image等檔案)之存放目錄

│ ├── favicon.ico

│ └── index.html # 網站入口

│

├── src/ # Vue應用程式的原始碼檔案

│ │

│ ├── assets/ # logo.png

│ ├── components/ # HelloWorld.vue (根Vue實例的子元件)

│ ├── App.vue # 根Vue實例的選項物件(options object)與模板

│ │ (.vue是vue的單檔元件，具一定格式(template/script/style)，需要vue-loader解析)

│ └── main.js # 根Vue實例建立之處

│

└── package.json # 專案的組態檔

* **.\package.json**

{

"name": "hello-world",

"version": "0.1.0",

"private": true,

"scripts": {

"serve": "vue-cli-service serve", (vue-cli-service程式位置：專案目錄\node\_modules\.bin)

"build": "vue-cli-service build",

"lint": "vue-cli-service lint"

},

"dependencies": {

"core-js": "^3.6.4",

"vue": "^2.6.11"

},

"devDependencies": {

"@vue/cli-plugin-babel": "~4.3.0",

"@vue/cli-plugin-eslint": "~4.3.0",

"@vue/cli-service": "~4.3.0",

"babel-eslint": "^10.1.0",

"eslint": "^6.7.2",

"eslint-plugin-vue": "^6.2.2",

"vue-template-compiler": "^2.6.11"

},

(略)

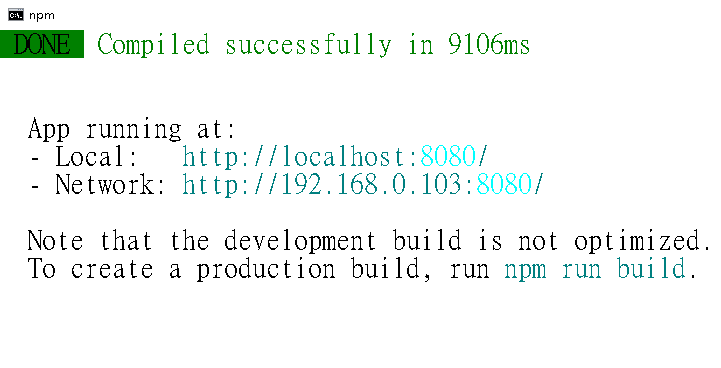
}

### 專案運作測試

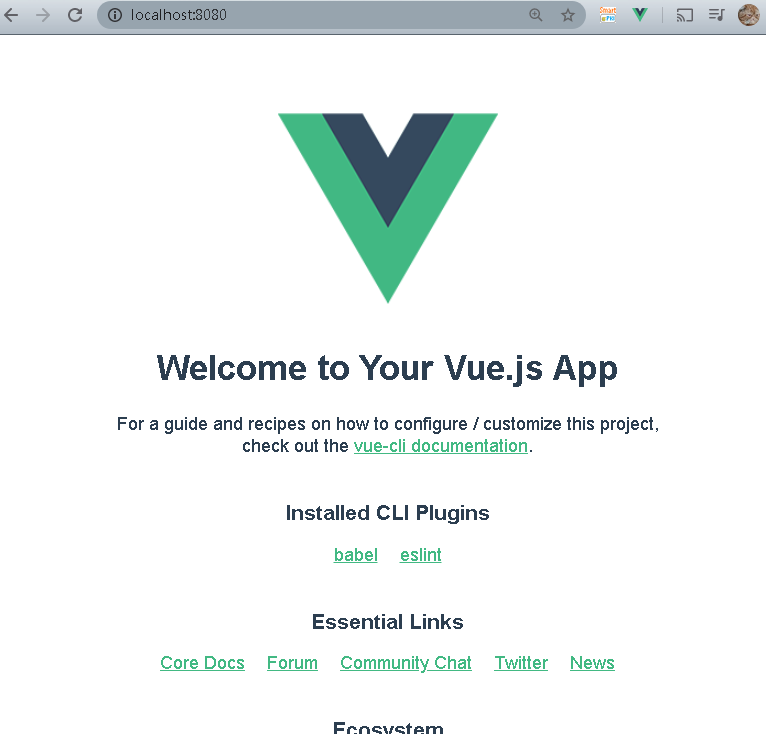
#### 啟動專案開發用的網頁伺服器

C:\\_vue\_misc\cli\_projects>cd hello-world

C:\\_vue\_misc\cli\_projects\hello-world>npm run serve



#### 開啟瀏覽器並輸入網址http://localhost:8080



### 專案佈署(Deployment)

#### 設定專案佈署所在的基礎網址

* 如果你的專案欲佈署在網站根目錄下的子目錄，則你必須進行下列設定(否則可跳過本步驟)：

vue.config.js

module.exports = {

publicPath: '/my-app/'

//設定你的應用程式包(application bundle)的佈署所在的基礎網址(base URL)。

(預設網站根目錄，即 '/' )。

//Vue CLI 3.3 以前版本須使用baseUrl選項。

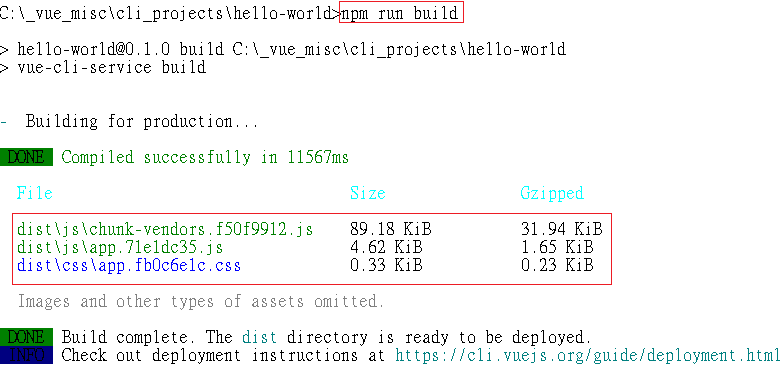
}

* + vue.config.js 是一個選用的配置檔案(必須自行新增，存放在根目錄中)，它能改變Vue CLI 的預設行為。
  + 這個檔案應該匯出(exports)一個包含了選項的物件：

#### 建置(Build)專案

C:\\_vue\_misc\cli\_projects>cd hello-world

C:\\_vue\_misc\cli\_projects\hello-world>npm run build



* vue-cli-service build指令會在dist/ 目錄產生一個可用於生產環境的包(a production-ready bundle)，其中具有JavaScript、CSS等檔案的壓縮檔，和為了更佳的快取(caching)效果而自動做的第三方JavaScript函式庫分割(例如chunk-vendors.f50f9912.js)。
* 上述的檔案清單會被列到HTML的檔案內(預設：index.html).

#### 佈署Builded Project到網頁伺服器

* 將dist\ 整個目錄下的所有檔案與目錄複製到網頁伺服器的文件根目錄即完成佈署。

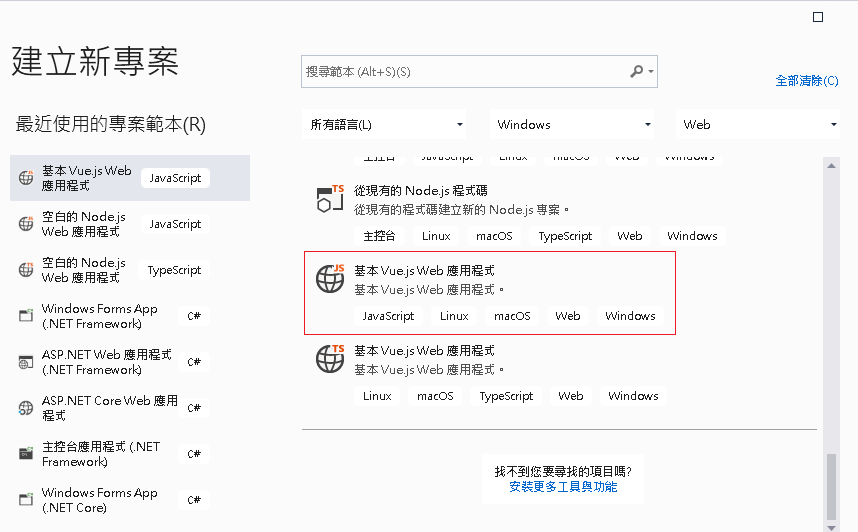
## 使用Visual Studio快速建立SPA專案

### 建立專案的步驟

* 準備進入「建立新專案」視窗

1. 在Visual Studio啟動時的「開始使用」選項中點選 「建立新的專案」
2. 在Visual Studio啟動後的「功能表」選項中點選 「檔案」>「新增」>「專案」

* 在「建立新專案」視窗點選： 「基本Vue.js Web應用程式」



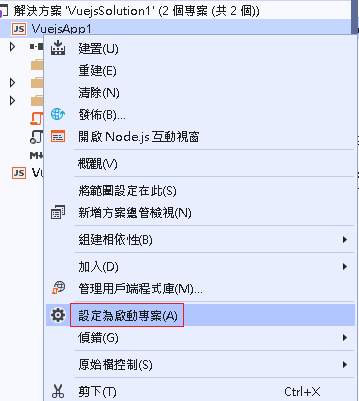
* 在「設定新專案」視窗輸入各項欄位值

輸入「專案名稱」、「位置」、「解決方案名稱」，然後按下「建立」按鈕

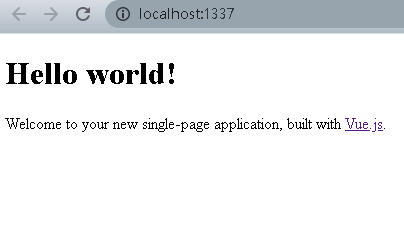


### 執行專案

* 在Visual Studio中**開啟**目標「專案/方案」
* 如果目標方案中包含多個專案時，在「方案總管」中以**滑鼠右鍵**點擊你欲執行的專案後點選 「設定為啟動專案」

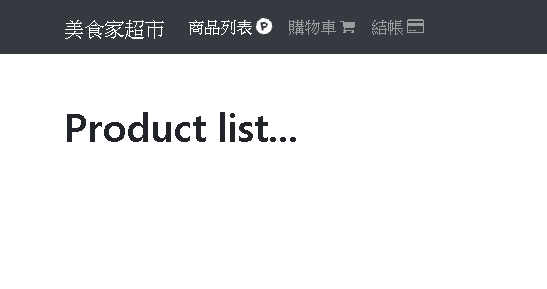


* 按下F5按鍵(功能表「偵錯」>「開始偵錯」) 或 Ctrl+F5按鍵(功能表「偵錯」>「啟動但不偵錯」)。執行結果如下：



# Module 17. 導引實作練習：主題網站 - 元件設立

## 導覽列設置



T17-11\index.html

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<title>導覽列設置</title>

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/css/bootstrap.min.css" />

<link rel="stylesheet"

href="https://stackpath.bootstrapcdn.com/font-awesome/4.7.0/css/font-awesome.min.css">

<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/js/bootstrap.min.js"></script>

<script src="https://unpkg.com/vue/dist/vue.min.js"></script>

</head>

<body>

<div id="app">

<nav class="navbar navbar-expand-lg navbar-dark bg-dark">

<div class="container">

<a class="**navbar-brand**" href="#">美食家超市</a>

<button class="**navbar-toggler**" type="button" data-toggle="collapse"

data-target="#main-menu">

<span class="**navbar-toggler-icon**"></span>

</button>

<div class="collapse **navbar-collapse**" id="main-menu">

<ul class="**navbar-nav** mr-auto">

<li v-for="nav in navs"

:class="['nav-item',{active:nav.id==currentComponentId}]"

v-bind:key="nav.id">

<a class="nav-link" @click="currentComponentId=nav.id">{{nav.title}}

<i :class="['fa', nav.icon,{'fa-spin':nav.id==currentComponentId}]"></i>

</a>

</li>

</ul>

</div>

</div>

</nav>

<div class="container mt-5">

<keep-alive>

<component v-bind:is="currentComponentId"></component>

</keep-alive>

</div>

</div>

<script src="main.js"></script>

</body>

</html>

T17-11\main.js

var app = new Vue({

el: '#app',

data: {

navs: [ { id: '**shop**', title: '商品列表', icon: 'fa-product-hunt' },

{ id: '**cart**', title: '購物車', icon: 'fa-shopping-cart' },

{ id: '**checkout**', title: '結帳', icon: 'fa-credit-card' }],

currentComponentId: ''

},

components: {

**shop**: { template: `<h1>Product list...</h1>` },

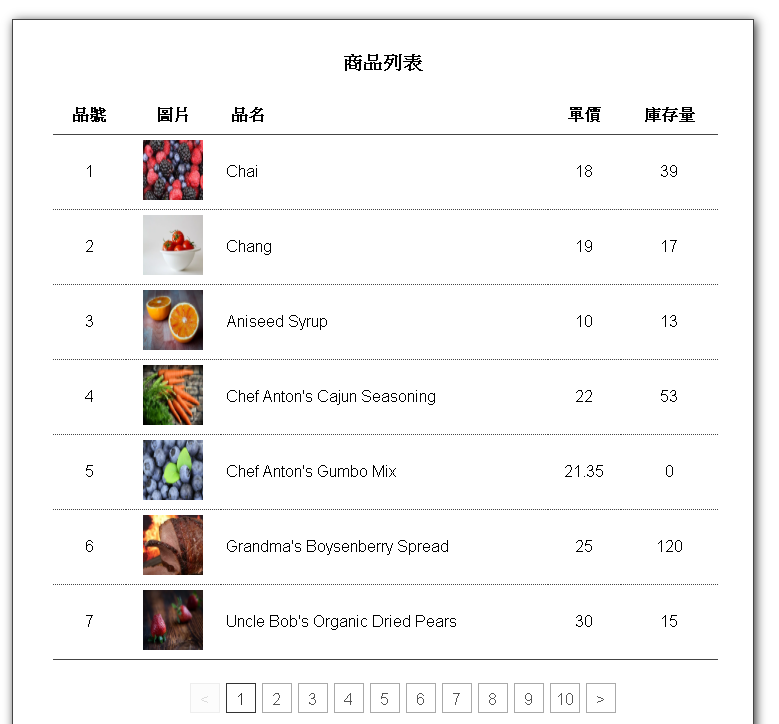
**cart**: { template: `<h1>Shopping Cart...</h1>` },

**checkout**: { template: `<h1>Checkout...</h1>` }

}

});

## 商品展示



T17-21\index.html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>商品展示</title>

<link rel="stylesheet" href="style.css">

<link rel="stylesheet" href="myPaginator.css" />

<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>

<script src="https://unpkg.com/vue/dist/vue.min.js"></script>

<script src="myPaginator.js"></script>

</head>

<body>

<div id="app">

<div id="container">

<div class="page-title">商品列表</div>

<table>

<thead>

<tr>

<th class="width-small">品號</th>

<th>圖片</th>

<th class="width-large justify-left">品名</th>

<th>單價</th>

<th>庫存量</th>

</tr>

</thead>

<tbody>

<tr v-for="(item, index) in itemList" :key="item.productid">

<td class="width-small">**{{**item.productid**}}**</td>

<td><img :src="item.photo" width="60" height="60" /></td>

<td class="width-large justify-left">**{{**item.productname**}}**</td>

<td>**{{**item.unitprice**}}**</td>

<td>**{{**item.unitsinstock**}}**</td>

</tr>

</tbody>

</table>

<div id="paginator1"></div>

</div>

</div>

<script src="main.js"></script>

</body>

</html>

T17-21\main.js

var app = new Vue({

el: '#app',

data: {

itemList: null

},

methods: {

doMyPagination: function (lengthPerPage, pageAnchorsPerBatch ) {

//以jQuery處理每個<table>的<tr>的分頁

var rows = $("table:eq(0) tr:has(td)");

var paginator = $("#paginator1");

tableRowsPagination(rows, paginator, lengthPerPage, pageAnchorsPerBatch);

}

},

created: function () {

var vm = this;

fetch("products.json")

.then(function (response) {

return response.json();

})

.then(function (arr) {

vm.itemList = arr;

});

},

updated: function () {//Data變更->DOM變更->jQuery.ready()內的Task才能正常運作

this.doMyPagination(7,10);

}

})

T17-21\products.json

[

{

"productid": 1,

"productname": "Chai",

"unitprice": 18.00,

"unitsinstock": 39,

"photo": "https://cdn.pixabay.com/photo/2010/12/13/10/05/background-2277\_\_340.jpg"

},

{

"productid": 2,

"productname": "Chang",

"unitprice": 19.00,

"unitsinstock": 17,

"photo": "https://cdn.pixabay.com/photo/2014/04/10/11/06/tomatoes-320860\_\_340.jpg"

},

(略)

]

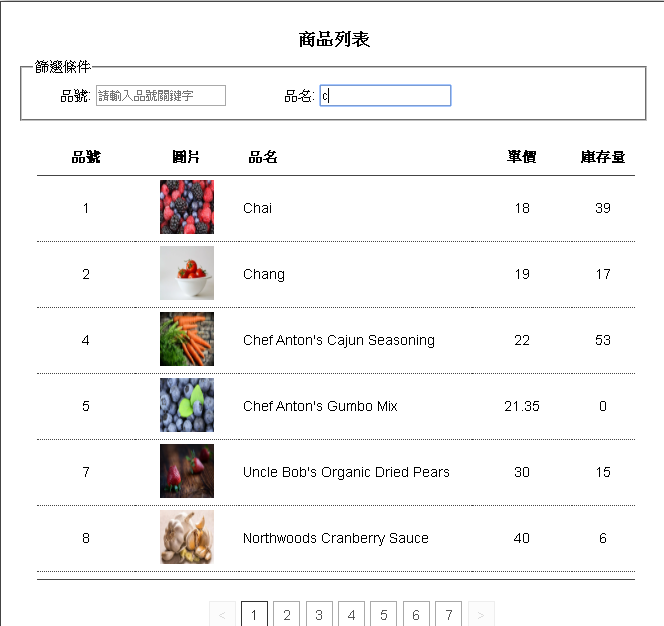
T17-21\style.css

T17-21\myPaginator.css

T17-21\myPaginator.js

(略)

## 條件篩選



T17-31\index.html

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>條件篩選</title>

<link rel="stylesheet" href="style.css">

<link rel="stylesheet" href="myPaginator.css" />

<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>

<script src="https://unpkg.com/vue/dist/vue.min.js"></script>

<script src="myPaginator.js"></script>

</head>

<body>

<div id="app">

<div id="container">

<div class="page-title">商品列表</div>

<fieldset>

<legend>篩選條件</legend>

<form class="filter">

<label>品號: <input type="text" v-model="productid" size="20"

placeholder="請輸入品號關鍵字"/></label>

<label>品名: <input type="text" v-model="productname" size="20"

placeholder="請輸入品名關鍵字" /></label>

</form>

</fieldset>

<table>

<thead>

<tr>

<th class="width-small">品號</th>

<th>圖片</th>

<th class="width-large justify-left">品名</th>

<th>單價</th>

<th>庫存量</th>

</tr>

</thead>

<tbody>

<tr v-for="(item, index) in itemList" :key="item.productid">

<td class="width-small">**{{**item.productid**}}**</td>

<td><img :src="item.photo" width="60" height="60" /></td>

<td class="width-large justify-left">**{{**item.productname**}}**</td>

<td>**{{**item.unitprice**}}**</td>

<td>**{{**item.unitsinstock**}}**</td>

</tr>

</tbody>

</table>

<div id="paginator1"></div>

</div>

</div>

<script src="main.js"></script>

</body>

</html>

T17-31\main.js

var app = new Vue({

el: '#app',

data: {

prodList: null,//用於保存完整products資料

productid: '',

productname: ''

},

methods: {

doMyPagination: function (lengthPerPage, pageAnchorsPerBatch) {

//以jQuery處理每個<table>的<tr>的分頁

var rows = $("table:eq(0) tr:has(td)");

var paginator = $("#paginator1");

tableRowsPagination(rows, paginator, lengthPerPage, pageAnchorsPerBatch);

}

},

computed: {

//itemList用於存儲使用者篩選的products資料

itemList: function () {

if (!this.prodList) //Ajax fetch products 尚未完成

return null;

else {

var vm = this;

return this.prodList.filter(

function (prod) {

return prod.productid.toString().indexOf(vm.productid) >= 0 &&

prod.productname.toUpperCase().indexOf(vm.productname.toUpperCase()) >= 0;

//註: JavaScript's 字串沒有contains()方法

});

}

}

},

created: function () {

var vm = this;

fetch("products.json")

.then(function (response) {

return response.json();

})

.then(function (arr) {

vm.prodList = arr;

//vm.itemList = vm.prodList;

});

},

updated: function () {//Data變更->DOM變更->jQuery.ready()內的Task才能正常運作

this.doMyPagination(6, 10);

}

})

T17-31\style.css

T17-31\myPaginator.css

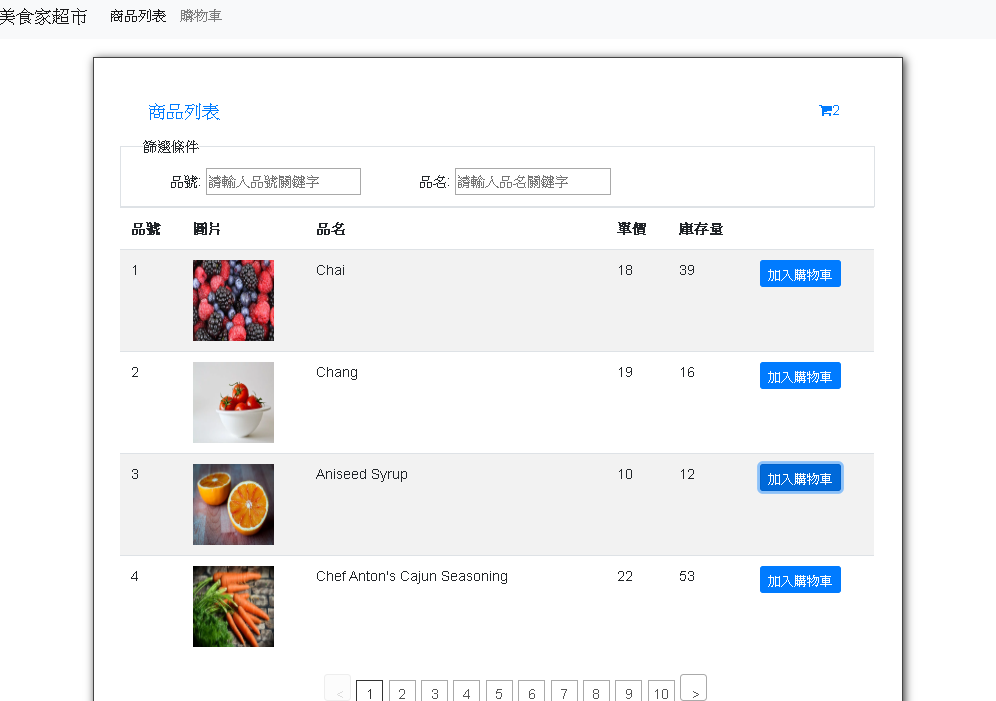
T17-31\myPaginator.js

(略)

# Module 18. 導引實作練習：主題網站 - 資料傳遞連動

## 加入購物車(SPA)

* 本範例為Visual Studio「基本Vue.js Web應用程式」之專案
* 專案名稱：C:\\_vue\vue\_projects\VuejsSolution1\VuejsAppT18-11



### index.html(網站入口)

public\index.html (網站入口)

* public/index.html 文件是一個會被 html-webpack-plugin 處理的模板。
* 在Vue Cli建置(build)過程中，Vue CLI會自動注入resource preload hint以及建置過程中所產生的JavaScript 和CSS檔案的資源鏈結(asset links)。
* 例如：

<!DOCTYPE html>

<html>

<head>

<meta charset=utf-8>

<link rel=icon href=/favicon.ico>

<title>購物車與結帳</title>

<link rel=stylesheet href=https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/css/**bootstrap.min.css**>

<link rel=stylesheet href=https://stackpath.bootstrapcdn.com/font-awesome/4.7.0/css/**font-awesome.min.css**>

<script src=https://code.jquery.com/**jquery-3.5.1.min.js**></script>

<script src=https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/js/**bootstrap.min.js**></script>

<link href=/css/app.fb42da70.css rel=preload as= style>

<link href=/js/app.9ee5af57.js rel=preload as=script>

<link href=/js/chunk-vendors.f2011b25.js rel=preload as=script>

<link href=/css/app.fb42da70.css rel=stylesheet>

</head>

<body>

<div id=app></div>

<script src=/js/chunk-vendors.f2011b25.js></script>

<script src=/js/app.9ee5af57.js></script>

</body>

</html>

【建置前原始碼】

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<link rel="icon" href="<%= BASE\_URL %>favicon.ico"> <!-- Vue Cli文字插值(BASE\_URL) -->

<title>購物車與結帳</title>

<link rel="stylesheet"

href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/css/**bootstrap.min.css**">

<link rel="stylesheet"

href="https://stackpath.bootstrapcdn.com/font-awesome/4.7.0/css/**font-awesome.min.css**">

<script src="https://code.jquery.com/**jquery-3.5.1.min.js**"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/js/**bootstrap.min.js**"></script>

</head>

<body>

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

<!-- built files will be auto injected -->

</body>

</html>

### main.js(JS主程式)

src\main.js (主程式：建立Vuex.Store/VueRouter/根Vue實例)

import Vue from 'vue';

//匯入/安裝 Vuex 與 VueRouter

import Vuex from 'vuex'; //vuex套件須先安裝(npm install vuex)

import VueRouter from 'vue-router'; //vue-router套件須先安裝(npm install vue-router)

Vue.use(Vuex); //Install a Vue.js plugin：Vuex.

Vue.use(VueRouter); //Install a Vue.js plugin： Vue-Router.

//匯入 單檔元件(Single File Ccomponents)

/\*

之前的全域/區域元件可能僅適合中小型的專案，對於較複雜的專案，一般會使用單檔元件(.vue) 。

單檔元件可解決全域/區域元件所遭遇的問題，例如:

1. 字串模板無法提供syntax highlighting功能且不易排版

2. 元件的模板不支援CSS

等等

\*/

import **App** from './App.vue';

import **shop** from './components/Shop.vue';

import **cart** from './components/Cart.vue';

//匯入 jQuery

import $ from 'jquery'; //jquery套件需先安裝(npm install jquery) (註: 因index.html會匯入jQuery，左列

在真正佈署到Web Server上時是不需的)

//匯入 自定義樣式表

import './stylesheets/style.css';

Vue.config.productionTip = true; //設定在瀏覽器之Console上顯示「生產模式提示」訊息

//---------------Vuex.Store---------------//

var store = new Vuex.Store({

state: {

prodList: [],

cart: []

},

getters: {

getProdList: function (state) {

return state.prodList;

},

getUnitsInStockForProd(state) {

return function (prodid) {//元件呼叫本getter時傳遞一個引數(prodid)進來->需回傳一個function

var prod = state.prodList.find(p => p.productid == prodid);

return prod.unitsinstock;

}

},

getNumOfItemsInCart: function (state) {

return state.cart.length;

},

getItemsInCart: function (state) {

return state.cart;

},

getTotalPriceForCart: function (state) {

var total = 0;

for (var i = 0; i < state.cart.length; i++) {

var unitprice = state.cart[i].unitprice;

var quantity = state.cart[i].quantity;

total += unitprice \* quantity;

}

return total;

}

},

mutations: {

loadProdList: function (state, prods) {

state.prodList = prods;

},

insertProdIntoCart: function (state, prod) {

//將商品項目加入購物車

var cartItem = {};

cartItem.productid = prod.productid;

cartItem.productname = prod.productname;

cartItem.photo = prod.photo;

cartItem.unitprice = prod.unitprice;

cartItem.quantity = 1; //\*\*\*

state.cart.push(cartItem);

//該商品的庫存量扣1

prod.unitsinstock--;

},

minusQtyForItemInCart: function (state, itemInCart) {

if (itemInCart.quantity <= 0)

return;

//扣購物車商品項目之購買數量

itemInCart.quantity--;

//加該商品的庫存量

var prod = state.prodList.find(p => p.productid == itemInCart.productid);

prod.unitsinstock++;

},

plusQtyForItemInCart: function (state, itemInCart) {

var prod = state.prodList.find(p => p.productid == itemInCart.productid)

if (prod.unitsinstock > 0) {//庫存量>0

//加購物車商品項目之購買數量

itemInCart.quantity++;

//減該商品的庫存量

prod.unitsinstock--;

}

}

},

actions: {

//從Web伺服器載入商品列表(products.json)

loadProdList: function (context) {

$.ajax({

method: 'get',

url: './products.json',

dataType: 'json',

success: function (data) {

context.commit('loadProdList', data); //commit mutation with payload: data

}

});

}

}

});

//---------------VueRouter---------------//

var routes = [

{

path: "/shop",

name: "shop",

component: **shop**

},

{

path: "/cart",

name: "cart",

component: **cart**

},

{

path: "/\*",

redirect: "/shop"

}

];

var router = new VueRouter({

routes: routes

});

//---------------root Vue instance---------------//

new Vue({

router: router,

store: store,

render: h => h(**App**) //字串模板的代替方案。

//該渲染函數接收一個createElement方法作為第一個參數來建立VNode(Virtual Node)。

//也可寫成render: function(h){ return h(App); }

//App為經vue-loader處理解析過的單檔元件(Single File Component)

created: function () {

this.$store.dispatch('loadProdList'); //從Web伺服器載入商品列表(products.json)

}

}).$mount('#app');

### App.vue(根Vue實例模板)

src\App.vue (根Vue實例之模板)

<!--single-file component (App.vue)-->

<template>

<div id="app">

<div>

<nav class="navbar navbar-expand-lg navbar-light bg-light">

<div class="container">

<a class="navbar-brand" href="#">美食家超市</a>

<button class="navbar-toggler" type="button" data-toggle="collapse"

data-target="#main-menu">

<span class="navbar-toggler-icon"></span>

</button>

<div class="collapse navbar-collapse" id="main-menu">

<ul class="navbar-nav mr-auto">

<li v-for="nav in navs" :key="nav.id"

:class="['nav-item', {active: currentRouteName==nav.id }]">

<router-link class="nav-link" :to="{name: nav.id}"

@click="currentRouteName=nav.id">{{nav.title}}</router-link>

</li>

</ul>

</div>

</div>

</nav>

<keep-alive>

<router-view></router-view>

</keep-alive>

</div>

</div>

</template>

<script>

export default {

data: function () {

return {

**navs:** [

{ id: 'shop', title: '商品列表' },

{ id: 'cart', title: '購物車' }

],

currentRouteName: ''

};

}

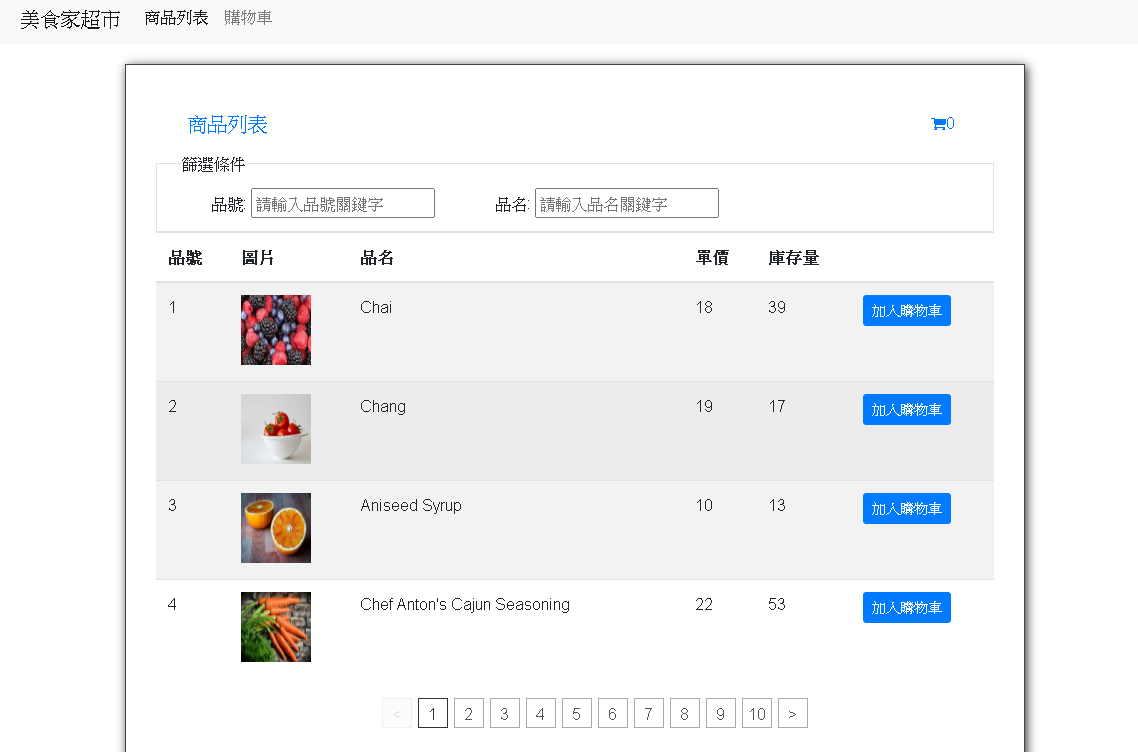
}

</script>

<style>

</style>

### Shop.vue(商品列表元件)



src\components\Shop.vue

<!--single-file component (Shop.vue)-->

<template>

<div id="app">

<div id="container">

<nav class="navbar navbar-expand-lg">

<div class="container">

<a class="navbar-brand" href="#">商品列表</a>

<button class="navbar-toggler" type="button">

<span class="navbar-toggler-icon"></span>

</button>

<div class="collapse navbar-collapse">

<ul class="navbar-nav ml-auto">

<li class="nav-item">

<router-link class="nav-link" :to="{name: 'cart'}">

<i class="fa fa-shopping-cart"></i>

{{numOfItemsInCart}}</router-link>

</li>

</ul>

</div>

</div>

</nav>

<fieldset class="border pl-4">

<legend class="w-auto" style="font-size:1em;">篩選條件</legend>

<form class="filter">

<label>品號: <input type="text" v-model="productid" size="20"

placeholder="請輸入品號關鍵字" /></label>

<label>品名: <input type="text" v-model="productname" size="20"

placeholder="請輸入品名關鍵字" /></label>

</form>

</fieldset>

<table class="table table-striped table-hover" id="shop-table">

<thead>

<tr>

<th>品號</th>

<th>圖片</th>

<th class="width-large justify-left">品名</th>

<th>單價</th>

<th>庫存量</th>

<th></th>

</tr>

</thead>

<tbody>

<tr v-for="(item) in itemList" :key="item.productid">

<td>{{item.productid}}</td>

<td><img :src="item.photo" width="70" height="70" /></td>

<td class="width-large justify-left">{{item.productname}}</td>

<td>{{item.unitprice}}</td>

<td>{{item.unitsinstock}}</td>

<td>

<button class="btn btn-sm"

:class="{'btn-primary':item.unitsinstock>0,

'btn-light':item.unitsinstock<=0}"

:disabled="!item.unitsinstock"

@click="insertProdIntoCart(item)">

加入購物車

</button>

</td>

</tr>

</tbody>

</table>

<div id="paginator1"></div>

</div>

</div>

</template>

<script>

import Vue from 'vue'

//匯入/安裝 Vuex

import Vuex from 'vuex'

Vue.use(Vuex) //Install a Vue.js plugin: Vuex.

//匯入 jQuery

import $ from 'jquery'; //jquery套件需先安裝(npm install jquery)

//匯入 自定義函數

import **tableRowsPagination** from '../javascripts/myPaginator.js';

//匯入 自定義樣式表

import '../stylesheets/myPaginator.css';

export default {

name: 'Shop',

data: function () {

return {

productid: '',

productname: ''

}

},

methods: {

...Vuex.mapMutations(['insertProdIntoCart'])

// map `this.insertProdIntoCart(prod)` to `this.$store.commit('insertProdIntoCart', prod)`

},

computed: {

...Vuex.mapGetters({ //Mapping Vuex.Stroe's getters to local computed properties

prodList: 'getProdList',

numOfItemsInCart: 'getNumOfItemsInCart',

}),

//itemList用於儲存使用者篩選的products資料

itemList: function () {

if (!this.prodList) //Ajax fetch products.json 尚未完成

return [];

else {

var vm = this;

return this.prodList.filter(

function (prod) {

return prod.productid.toString().indexOf(vm.productid) >= 0 &&

prod.productname.toUpperCase().indexOf(

vm.productname.toUpperCase()) >= 0;

});

}

}

},

updated: function () {//Data變更->DOM變更->將<tbody>之<tr>重新分頁

//以jQuery處理每個<table>的<tr>的分頁

var rows = $("table:eq(0) tr:has(td)");

var paginator = $("#paginator1");

**tableRowsPagination**(rows, paginator, 4, 10);

//tableRowsPagination(rows, paginator, lengthPerPage, pageAnchorsPerBatch);

}

}

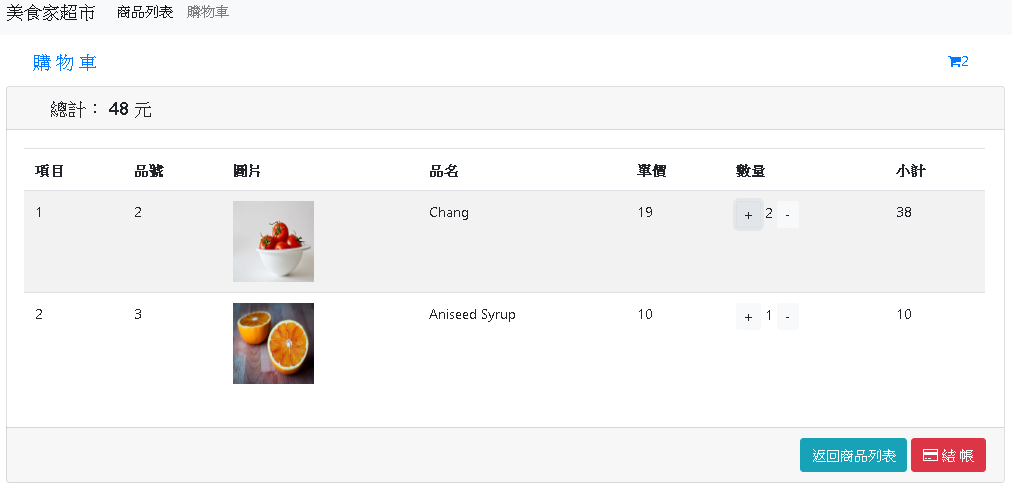
</script>

<!-- Add "scoped" attribute to limit CSS to this component only -->

<style scoped>

</style>

### Cart.vue(購物車元件)



src\components\Cart.vue

<!--single-file component (Cart.vue)-->

<template>

<div class="container">

<nav class="navbar navbar-expand-lg">

<div class="container">

<a class="navbar-brand" href="#">購 物 車</a>

<button class="navbar-toggler" type="button">

<span class="navbar-toggler-icon"></span>

</button>

<div class="collapse navbar-collapse">

<ul class="navbar-nav ml-auto">

<li class="nav-item">

<router-link class="nav-link" :to="{name: 'cart'}">

<i class="fa fa-shopping-cart"></i>

{{numOfItemsInCart}}</router-link>

</li>

</ul>

</div>

</div>

</nav>

<div class="card">

<div class="card-header pl-5 h5">

總計： <span>{{totalPriceForCart}}</span> 元

</div>

<div class="card-body">

<table class="table table-striped">

<thead>

<tr>

<th>項目</th>

<th>品號</th>

<th>圖片</th>

<th>品名</th>

<th>單價</th>

<th>數量</th>

<th>小計</th>

</tr>

</thead>

<tbody>

<tr v-for="(item,index) in itemsInCart" v-bind:key="item.productid">

<td>{{index+1}}</td>

<td>{{item.productid}}</td>

<td><img :src="item.photo" width="90" height="90" /></td>

<td>{{item.productname}}</td>

<td>{{item.unitprice}}</td>

<td>

<button class="btn btn-sm"

:class="{'btn-danger': !unitsInStockForProd(item.productid),

'btn-light': unitsInStockForProd(item.productid)>0}"

:disabled="!unitsInStockForProd(item.productid)"

@click="plusQtyForItemInCart(item)">

+

</button>

{{item.quantity}}

<button class="btn btn-sm"

:class="{'btn-danger': item.quantity<=0,

'btn-light': item.quantity>0}"

:disabled="!item.quantity"

@click="minusQtyForItemInCart(item)">

-

</button>

</td>

<td>{{item.unitprice\*item.quantity}}</td>

</tr>

</tbody>

</table>

</div>

<div class="card-footer pl-5 text-right">

<router-link :to="{name: 'shop'}" class="btn btn-info">返回商品列表</router-link>

<button class="btn btn-danger"><i class="fa fa-credit-card"></i>結 帳</button>

</div>

</div>

</div>

</template>

<script>

<script>

import Vue from 'vue'

//匯入/安裝Vuex

import Vuex from 'vuex'

Vue.use(Vuex) //Install a Vue.js plugin: Vuex.

export default {

computed: {

...Vuex.mapGetters({ //Mapping Vuex.Stroe's getters to local computed properties

numOfItemsInCart: 'getNumOfItemsInCart',

itemsInCart: 'getItemsInCart',

totalPriceForCart: 'getTotalPriceForCart',

unitsInStockForProd: 'getUnitsInStockForProd'

})

},

methods: {

...Vuex.mapMutations(['plusQtyForItemInCart', 'minusQtyForItemInCart'])

// map `this.plusQtyForItemInCart(itemInCart)` to

`this.$store.commit('plusQtyForItemInCart', itemInCart)`. (以此類推)

}

};

</script>

<!-- Add "scoped" attribute to limit CSS to this component only -->

<style scoped>

</style>

## 結帳頁面(SPA)

* 範例程式：請參考18.1範例

## 表單驗證



T18-31\index.html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>表單驗證</title>

<style type="text/css">

.main-box {

margin: 50px auto;

width: 500px;

height: 380px;

padding: 20px;

box-shadow: 5px 5px 10px #999;

border: 1px solid #fff;

text-align: center;

}

.main-box form {

padding: 10px 10px 10px 10px;

text-align: left;

}

.main-box label {

display: inline-block;

width: 100px;

text-align: right;

margin: 0 15px 0 0;

color: black;

}

.main-box #button1 {

margin-left: 120px;

}

.main-box .fail {

color: red;

}

.main-box .ok {

color: blue;

}

#error-msg {

margin: 10px auto;

width: 300px;

height: 100px;

padding: 10px;

box-shadow: 2px 2px 5px #999;

border: 1px solid #fff;

text-align: center;

color:darkred;

}

</style>

<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>

<script src="https://unpkg.com/vue/dist/vue.min.js"></script>

</head>

<body>

<div id="app">

<div class="main-box">

<h2>Login</h2>

<form v-on:submit="checkForm">

<p>

<label for="username">使用者名稱:</label>

<input type="text" name="username" id="username" v-model="username">

</p>

<p>

<label for="password">密碼:</label>

<input type="text" name="password" id="password" v-model="password">

</p>

<p>

<button id="button1">送出</button> <!--button 預設type='submit'-->

</p>

<div v-if="errors.length" id="error-msg">

<b>請修正下列錯誤:</b>

<ul>

<li v-for="error in errors">{{ error }}</li>

</ul>

</div>

</form>

<p id="response-msg"></p>

</div>

</div>

<script src="main.js"></script>

</body>

</html>

T18-31\main.js

var vm = new Vue({

el: '#app',

data: {

errors: [],

username: null,

password: null

},

methods: {

checkForm: function (e) {

this.errors = [];

if (!this.username) {

this.errors.push('「使用者名稱」是須填欄位...');

}

if (!this.password) {

this.errors.push('「密碼」是須填欄位...');

}

else if (this.password.length < 5) {

this.errors.push('「密碼」必須具5個(含)字元以上..');

}

//表單通過驗證時將送出Ajax請求

if (!this.errors.length) {

$.ajax({

method: 'post',

url: 'AjaxServer.aspx',

data: $(e.currentTarget).serialize(),

dataType: 'json',

success: function (data) {

if(data.status=='ok')

$("#response-msg").html(`<span class='ok'>${data.message}</span>`);

else

$("#response-msg").html(`<span class='fail'>${data.message}</span>`);

setTimeout(function () {

$("#response-msg").html('');

}, 2000);

},

error: function (jqXHR, textStatus, errorThrown) {

$("#response-msg").html(`${textStatus}: ${errorThrown}`);

}

});

}

//取消同步請求

e.preventDefault();

}

}

})

T18-31\AjaxServer.aspx

<%@ Page Language="C#" ContentType="text/html" %>

<%

if (Request["username"].Equals("hello") && Request["password"].Equals("kitty"))

Response.Write("{\"status\": \"ok\", \"message\":\"歡迎"+ Request["username"] + "光臨!\"}");

//{"status": "ok", "message":"歡迎hello光臨!"}

else

Response.Write("{\"status\": \"fail\", \"message\":\"您的登入資訊不正確!\"}");

//{"status": "fail", "message":"您的登入資訊不正確!"}

%>

# 附錄

## 過濾器(Filters)

* Vue.js 允許你定義過濾器()，可被用於一些常見的文字格式化。
* 過濾器可以用在兩個地方：
  + 小鬍子標籤文字插值(mustache interpolations)
  + v-bind運算式
* 過濾器應該被附加在JavaScript運算式的尾部，以“管道/pipe”符號指示。例如：

{{ message | upperCase }}

<div v-bind:id="rawId | formatId"></div>

* 定義過濾器的方式
  + 你可以在一個元件的選項中定義區域(local)過濾器：

var vm = new Vue({

el: "#app",

//區域過濾器(local filter)

filters: {

toUpperCase: function (val) {

if (!val)

return '';

return val.toString().toUpperCase();

}

}

});

* + 或者在建立Vue實例之前定義全域(global)過濾器：

//全域過濾器(global filter)

Vue.filter('toUpperCase', function (val) {

if (!val)

return '';

return val.toString().toUpperCase();

});

new Vue({

// ...

})

* 當全域過濾器和區域過濾器同名時，會採用區域過濾器。

### 範例 T19-11.html



<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<title>Filters</title>

<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>

</head>

<body>

<div id="app">

<h1>{{ message | toUpperCase | appendPostfix('...')}}</h1>

</div>

<script>

new Vue({

el: '#app',

data: {

message: 'Hello Vue'

},

filters: {

toUpperCase: function (val) { //過濾器函式第一個參數(val)會被傳入管道左邊的

運算式的值 (Corelab 17-3)

if (!val)

return '';

return val.toString().toUpperCase();

},

appendPostfix: function (val, postfix) {

//過濾器函式第一個參數(val)會被傳入管道左邊的運算式的值

//第二個參數(postfix)會被傳入過濾器呼叫時的引數值

if (!val)

return '';

else if (!postfix)

return val;

else

return val.toString() + postfix;

}

}

});

</script>

</body>

</html>

## 使用v-clock指令

* 這個指令會在相關Vue實例完成其模板編譯之後被移除。 (Corelab 18-1)
* 和CSS規則如 [v-cloak] { display: none } 一起使用時，這個指令可以隱藏未編譯的 Mustache標籤直到Vue實例準備完畢。

### 範例 T19-21.html



(2秒後)



<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<title>v-cloak指令</title>

<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>

<style>

[v-cloak] { display: none; }

</style>

</head>

<body>

<div id="app">

<h1> Hello! <span v-cloak>{{ message }}</span></h1>

</div>

<script>

var vm = new Vue({

data: {

message: 'Vue...'

}

});

setTimeout(function() {

vm.$mount("#app"); //手動起始"模板編譯"

}, 2000);

</script>

</body>

</html>