前言

現今的前端開發已經跟以往不同。網站內容愈來愈豐富也更具互動性，身為前端開發者的我們需要持續往裡頭加入複雜的功能並使用功能更強大的工具。透過 jQuery 來更新網頁中的一小段文字一點都不難，不過，因為我們還需要做更多的事——更新網頁中更多、更具互動性的內容；處理複雜的狀態；運用客戶端路由；以及編寫並組織好更多的程式碼——使用 JavaScript 框架可以讓我們的工作輕鬆許多。

這裡所指的**框架**（*framework*）是一種 JavaScript 工具，讓開發者在建置內容豐富（rich）且互動性高的網站時，能輕鬆一些。框架內含功能，讓我們可以製作出功能完整的網頁應用程式（web application）：操作複雜的資料並將之顯示在網頁上、處理客戶端路由而不需要依賴伺服器，有時甚至可以讓我們建置出，只需要在初始下載階段與伺服器連線一次，就可以完整運作的網站。Vue.js 是最新流行的 JavaScript 框架，而且正迅速普及當中。Evan You，當時還在 Google 服務，在 2014 年初時編寫並釋出第一版的 Vue.js。本書編寫期間，這個專案在 GitHub 上已累積了 75,000 顆星，成為 GitHub 上獲得最多星之專案的第八名，而且數字仍持續在快速累積當中[1]。Vue 有數以百計的協作者（collaborators），而且每天透過 npm 被下載的次數約 40,000 次。她內含對開發網站與應用程式有用的功能：能處理 DOM 與傾聽事件的強大樣版語法、資料更新後無須再更新樣版的高響應性，以及一些讓你能輕鬆地操作資料的功能。

1 在我開始編寫本書時，此專案在 GitHub 上已獲得 65,000 顆星。在你閱讀本書時，此專案所獲得的星星數可能已超過 75,000 顆了！

目標讀者

若你瞭解 HTML 與 JavaScript 並且想要透過框架的使用，將目前所學提升到下一個等級，本書就是為你而寫的。雖然本書並不要求你已能靈活運用 JavaScript，但我並不會對運用 Vue.js 功能的 JavaScript 範例程式碼，再多作解釋，所以對 JavaScript 有基本的概念是必須的。本書的範例程式碼都以 ECMAScript 2015 寫成，這是 JavaScript 最新的版本，涵蓋如 `const`、胖箭號（fat arrow）函式，以及解構（destructuring）等語言功能。若你不熟悉 ES2015，別擔心——有許多文章與資源可以幫你[2]，而且大多數的範例程式碼都容易閱讀與瞭解。

若你有用過 React，本書也適合你，但請先讀附錄 B，其中對一些 Vue.js 概念與你熟悉的 React 寫法，有作比較與說明。

本書的結構

本書有 7 章與 2 個附錄：

第 1 章，Vue.js：基礎

第一章介紹 Vue.js 的基礎，即本書所要討論的主要技術。其中將說明它的安裝與引入網頁，以及如何運用它將資料顯示在網頁上的方法。

第 2 章，Vue.js 中的組件

Vue.js 允許——且鼓勵——你將程式碼分割成組件（components），如此你就能在碼庫（codebase）中重複運用這些組件。本章說明你可以怎麼做，以建立更容易瞭解與維護的碼庫。

第 3 章，以 Vue 處理樣式

本書其他章節所討論的是 HTML 與 JavaScript，但本章的主題則是討論製作網站的視覺元素。其中將說明 Vue 如何與 CSS 及樣式搭配，讓網站與應用程式更有型。其內建的輔助器功能（helper functionality）可幫助你處理樣式。

2 我推薦 Babel 網站上的“Learn ES2015”（*https://babeljs.io/learn-es2015*）。

第 4 章，渲染函式與 *JSX*

若你曾看過許多 Vue 程式碼或讀過入門指南（Getting Started guide），除了你認識的樣版語法（templating syntax）外，Vue 亦支援自定的染渲函式，讓你可以使用 JSX。若曾用過 React 的話，你應該已熟悉這種語法。我會在本章中說明如何在 Vue 應用程式中使用 JSX。

第 5 章，以 *vue-router* 處理客戶端路由

Vue 本身只是一個視版層（view layer）。要建立不需要重新要求就能存取多重網頁的應用程式（用行話來說：即單網頁應用程式），你需要在網站中加入 vue-router，你可以透過它來處理路由——也就是說，它可以在特定路徑被要求時，將頁面導到需要被執行的程式碼與需要被顯示的內容上。本章將說明如何運用它的方法。

第 6 章，以 *Vuex* 進行狀態管理

在內含多層組件的複雜應用程式中，在各組件間傳遞資料可能會有點麻煩。Vuex 可讓你將管理應用程式狀態的工作，集中在一個地方處理。在本章中，我會說明如何運用它以方便地管理複雜應用程式狀態的方法。

第 7 章，測試 *Vue* 組件

至此，你已經學到要讓網頁運行所需要瞭解的內容了。不過，在未來維護網站時，你還需要能為網站編寫測試。本章說明如何運用 vue-test-utils 為你的 Vue 組件編寫單元測試，以確保它們不會在未來出問題。

附錄 *A*，架設 *Vue*

vue-cli 能讓你運用現成的樣版快速地架設出 Vue 應用程式。這份簡短的附錄說明它的運作方式並呈現一些範例作為示範。

附錄 *B*，從 *React* 轉 *Vue*

若你曾使用過 React，或許你已熟悉許多 Vue 的概念。這份附錄列出 Vue 與 React 二者間的某些異同。

第一章

Vue.js：基礎

如同前言所說的，Vue.js 是一套程式庫，其位於一個生態體系的核心，而這個體系可以讓我們做出強而有力的客戶端應用。我們並不需使用此生態體系所提供的全部功能才能建造網站，所以我們先從 Vue 本身開始。

為何要用 Vue.js ？

若不運用框架（framework），我們的應用程式最終會變成一堆無法維護的程式碼。其實大部分所要處理的事，框架都可以幫我們處理。以下列二個範例來作說明。這二支範例程式的功用都是從一個 Ajax 資料來源下載一份列表中的資料項目，然後在網頁中將之顯示出來。第一個範例所使用的是 jQuery，而第二個範例則使用 Vue。

透過 jQuery，我們下載資料項目、選取 `ul` 元素，然後，若有下載到資料項目則逐項循環，接著手動建立一個列表元素，需要的話，加進 `is-blue` 樣式類別，然後將項目資料設定成其中的文字。最後加進 `ul` 元素中：

```
<ul class="js-items"></ul>

<script>
  $(function () {
    $.get('https://example.com/items.json')
      .then(function (data) {
        var $itemsUl = $('.js-items');

        if (!data.items.length) {
          var $noItems = $('li');
          $noItems.text('Sorry, there are no items.');
          $itemsUl.append($noItems);
```

```
        } else {
          data.items.forEach(function (item) {
            var $newItem = $('li');
            $newItem.text(item);

            if (item.includes('blue')) {
              $newItem.addClass('is-blue');
            }

            $itemsUl.append($newItem);
          });
        }
      });
  });
</script>
```

這段程式碼做了下列工作：

1. 使用 `$.get()` 發出 Ajax 要求。
2. 選取符合 `.js-items` 的元素，並將之存進 `$itemsUl` 物件中。
3. 若下載的列表中沒有任何項目，則建立 `li` 元素，將 `li` 元素中的文字設定成沒有項目的說明文字，再將之加進文件（document）中。

   若列表中有項目存在，則在迴圈中逐一進行循環。
4. 為列表中的每一個項目，建立一個 `li` 元素，並將其中的文字設定成項目中的資料。接著，若項目中含有 `blue` 字串，則將該元素的樣式類別設定成 `is-blue`。最後將這個元素加進文件中。

每一個步驟都要手動來完成——每一個元素都要個別建立出來並加到文件中。我們要看過所有程式碼，才能弄懂它到底做了些什麼，它的功用沒辦法一眼就能看出來。

透過 Vue，具相同功能的程式碼則更容易閱讀與瞭解——既使你現在還不熟悉 Vue：

```
<ul class="js-items">
  <li v-if="!items.length">Sorry, there are no items.</li>
  <li v-for="item in items" :class="{ 'is-blue': item.includes('blue') }">
    {{ item }}</li>
</ul>
<script>
  new Vue({
    el: '.js-items',
    data: {
```

```
        items: []
      },
      created() {
        fetch('https://example.com/items.json')
          .then((res) => res.json())
          .then((data) => {
            this.items = data.items;
          });
      }
    });
  </script>
```

這段程式碼做了下列工作：

1. 用 `fetch()` 發出 Ajax 要求。
2. 將 JSON 格式的回應資料解析成 JavaScript 的物件。
3. 將下載的項目存在 `items` 資料屬性中。

這就是這段程式碼實際上的邏輯。現在這些資料項目已被下載並儲存好了，我們可以透過 Vue 的樣版功能，將這些元素寫進 *Document Object Model*（*DOM*）中，這個模型就是 HTML 網頁顯示資料的模型。我們跟 Vue 說，每一個項目需要一個 `li` 元素，其值應為 `item`。Vue 就會建立元素並為我們設定樣式類別。

先別擔心還無法完全瞭解這段範例程式。我會把步調放慢，並一一在本書中介紹這些概念。

透過將程式邏輯與視版（view）邏輯完全分開，第二段範例程式不僅明顯較短，讀起來也容易許多。與其要瞭解 jQuery 在何時加了什麼東西進來，我們可以直接看樣版：若沒有資料項目需要顯示，則呈現警告訊息；否則各資料項目就會以列表元素的方式呈現在網頁中。這種差別在較大型的程式中更容易被看出來。試想我們要在網頁中加進一個可傳送要求給伺服器的重新整理（reload）按鈕，以 Vue 的範例來說，只要加進幾行程式碼就可以了，但若要以 jQuery 來實作，則會複雜許多。

除了 Vue 框架的核心外，有許多程式庫可以與 Vue 搭配，這些程式庫也是由維護 Vue 的同一組人在維護。在路由（routing）——依據應用程式的 URL 顯示不同的內容——方面，有 vue-router 可用。在狀態管理——透過一全域儲存區分享資料給不同組件——方面，有 vuex 可用。在 Vue 組件的單元測試方面，則有 vue-test-utils。稍後我會在本書中介紹並說明這三個程式庫：vue-router 在第 5 章介紹，vuex 在第 6 章，而 vue-test-utils 則在第 7 章說明。

安裝與設定

安裝 Vue 並不需要任何特別的工具。照底下這樣子寫就行了：

```
<div id="app"></div>
<script src="https://unpkg.com/vue"></script>
<script>
  new Vue({
    el: '#app',
    created() {
      // 這裡的程式碼會在啟動時執行
    }
  });
</script>
```

上列範例包含三個重要的東西。首先，有一個 ID 為 app 的 div，這是為 Vue 進行初始化的地方——有一些因素，我們不能在 body 元素上進行初始化。接著將 Vue 的 CDN[1] 版下載進網頁中。你可以使用 Vue 的本地端複本，但為了簡單化，現在我們先用這種方式。最後則是執行我們自己的 JavaScript，也就是建立一個 Vue 的實例，並設定其中的 el 屬性，讓它指到前述的 div 上。

這種方式很適合簡單的網頁，不過若網頁開始變複雜，或許你想要透過像 webpack 這類的打包器（bundler）來做。先不說別的，用打包器來做可讓你用 ECMAScript 2015（及以上）來編寫 JavaScript，在一個檔案中寫好一個組件並將組件匯入到其他組件中使用，也可以編寫特定組件專屬的 CSS 樣式（第 2 章會談到更多關於這方面的細節）。

vue-loader 與 webpack

vue-loader 是一個供 webpack 使用的載入器（loader），它可讓你將一個組件所需的所有 HTML、JavaScript 與 CSS 編寫進一個檔案中。我們會在第 2 章中探索它的功能，現在你只需知道如何把它安裝好就行了。若你已經有設定好的 webpack 或有偏好的 webpack 樣版，則可透過 npm 安裝 vue-loader 的方式將它安裝好。接著將底下的程式碼加進 webpack 載入器組態（configuration）中：

1 內容傳遞網路（content delivery network，CDN）架設在世界各地的伺服器上，讓內容可以迅速地傳遞。CDN 有助於開發與快速雛型製作，不過在產品版上使用 unpkg 前，你應該要再研究看看它是否適合於你的應用。

```
module: {
  loaders: [
    {
      test: /\.vue$/,
      loader: 'vue',
    },
    // ... 其他的載入器 ...
  ]
}
```

若你還沒有設定好的 webpack 或者還在掙扎怎麼加進 vue-loader，別擔心！我也沒有從頭開始設置 webpack 過。你可以透過一個已安裝有 vue-loader 的 webpack 現成樣版，來建置你的 vue 專案。用 vue-cli 來進行：

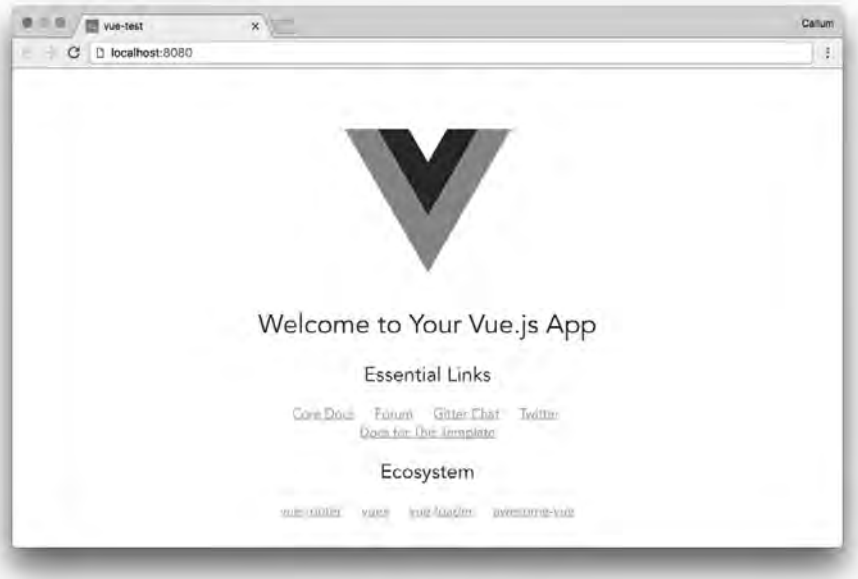
```
$ npm install --global vue-cli
$ vue init webpack
```

接著你需要回答一串問題，如專案名稱與需要哪些依賴模組包等，回答這些問題之後，vue-cli 就會為你建置好一個基本的專案：

```
› vue init webpack

? Generate project in current directory? Yes
? Project name vue-test
? Project description A Vue.js project
? Author Callum Macrae <callum@macr.ae>
? Vue build standalone
? Install vue-router? No
? Use ESLint to lint your code? No
? Setup unit tests with Karma + Mocha? No
? Setup e2e tests with Nightwatch? No

   vue-cli · Generated "vue-test".

   To get started:

     npm install
     npm run dev

   Documentation can be found at https://vuejs-templates.github.io/webpack
```

現在就試試看，然後按照它顯示的說明啟動伺服器。

恭喜——你已設置好第一個 Vue 專案了！



你可以看看其所產生的檔案，瞭解它幫你做了什麼事。大部分重要的東西都產生在 *src* 檔案夾下的 *.vue* 檔案中。

樣版、資料與指令

Vue 的核心是如何將資料呈現在網頁上的一種方法，而這項工作要透過**樣版**（*templates*）來完成。一般的 HTML 可透過特殊的屬性——稱為**指令**（*directives*）——來編寫，我們用這些指令告訴 Vue 要做哪些工作，以及要對我們所提供的資料作哪些處理。

我們直接看範例。底下的範例會在上午顯示“Good morning!”，在下午 6 點前顯示“Good afternoon!”，之後則顯示“Good evening!”：

```
<div id="app">
  <p v-if="isMorning">Good morning!</p>
  <p v-if="isAfternoon">Good afternoon!</p>
  <p v-if="isEvening">Good evening!</p>
```

```
</div>
<script>
  var hours = new Date().getHours();

  new Vue({
    el: '#app',
    data: {
      isMorning: hours < 12,
      isAfternoon: hours >= 12 && hours < 18,
      isEvening: hours >= 18
    }
  });
</script>
```

我們先看最後的部分：即 data 物件。它是我們要 Vue 在樣版中顯示的資料。我們設定了它的三個屬性——`isMorning`、`isAfternoon` 以及 `isEvening`，視所處時間不同，其中有一項條件為真，另外二項為假。

接著，在樣版中，依據變數值，我們使用 `v-if` 指令來判斷該顯示三句問候語的哪一句。只有在傳進 `v-if` 的變數值為真時，該項問候語才會被顯示；否則該元素不會被寫進網頁。若時間是 2:30 p.m.，輸出的網頁內容如下：

```
<div id="app">
  <p>Good afternoon!</p>
</div>
```

雖然 Vue 具有響應式功能 (reactive functionality)，上述範例並不具響應性，網頁內容也不會隨著時間的改變而更新。稍後我們會就響應性進行更詳細的說明。

雖然有許多跟上例重複的碼，我們來試著將程式改寫：將時間設定給 data 變數，然後在樣版中運用比較邏輯。運氣還不錯，我們可以這樣子來改！ Vue 會對 `v-if` 中的運算式求值：

```
<div id="app">
  <p v-if="hours < 12">Good morning!</p>
  <p v-if="hours >= 12 && hours < 18">Good afternoon!</p>
  <p v-if="hours >= 18">Good evening!</p>
</div>
<script>
  new Vue({
    el: '#app',
```

```
      data: {
        hours: new Date().getHours()
      }
    });
  </script>
```

以這種方式來寫碼，將商業邏輯放在 JavaScript 中，將視版邏輯放在樣版中，我們一眼就可以看出網頁中將顯示的內容會是什麼。比起讓程式中的一些元素跟負責決定什麼該顯示什麼該隱藏的 JavaScript 遠離，這樣的寫法要好很多。

待會兒，我們會介紹計算屬性（computed properties），透過計算屬性，我們可以讓上列的程式碼再更簡潔一些——它有不少重複的部分。

除了使用指令之外，我們也可以透過插值（interpolation）的方式，將資料傳遞給樣版，如下：

```
<div id="app">
  <p>Hello, {{ greetee }}!</p>
</div>
<script>
  new Vue({
    el: '#app',
    data: {
      greetee: 'world'
    }
  });
</script>
```

輸出的網頁內容如下：

```
<div id="app">
  <p>Hello, world!</p>
</div>
```

我們也可以結合二者，使用指令與插值顯示那些已定義或有用的文字。試試看能否猜出下列程式碼會在網頁上輸出什麼：

```
<div id="app">
  <p v-if="path === '/'">You are on the home page</p>
  <p v-else>You're on {{ path }}</p>
</div>
<script>
  new Vue({
    el: '#app',
    data: {
```

```
        path: location.pathname
      }
    });
  </script>
```

location.pathname 是網頁 URL 中的路徑部分，因此，使用者處於網站首頁時，它會是“/”，處於網站的其他位置時，它可能會是“/post/1635”。上例說明了它會用 v-if 指令來判斷你目前的位置是否是在首頁，它會檢測 path 是否等於“/”（若是，使用者目前瀏覽的就是網站的首頁），接著我們引入新的指令，v-else。它很簡單：當它被用在帶有 v-if 的元素之後時，其功用就會跟 if-else 敘述中的 else 一樣。第二個元素會在第一個元素的判斷條件不為真時顯示在網頁上。

如你所看到的，除了可以傳遞字串與數值之外，也可以將其他型別的資料傳進樣版中。因為我們可以在樣版中執行簡單的運算式，故可以將陣列或物件傳進樣版中，然後再搜尋個別的屬性或項目：

```
<div id="app">
  <p>The second dog is {{ dogs[1] }}</p>
  <p>All the dogs are {{ dogs }}</p>
</div>
<script>
  new Vue({
    el: '#app',
    data: {
      dogs: ['Rex', 'Rover', 'Henrietta', 'Alan']
    }
  });
</script>
```

底下是其所輸出的網頁內容：

```
The second dog is Rover

All the dogs are [ "Rex", "Rover", "henrietta", "Alan" ]
```

如你所看到的，若你將整個陣列或物件輸出到網頁上，Vue 會輸出 JSON 編碼過的值到網頁上。進行除錯時，與其用紀錄（logging）的方式將訊息輸出到主控台，直接輸出到網頁上可能更好用，因為顯示在網頁上內容會跟著該值的改變而改變。

v-if 對 v-show

你已經看到 v-if 可以用來顯示或隱藏一個網頁元素了，但它是怎麼運作的，而它跟相近的 v-show 又有何不同？

若 v-if 指令所計算出的值為假[2]，則該元素不會輸出到 DOM 中。

底下的 Vue 樣版

```
<div v-if="true">one</div>
<div v-if="false">two</div>
```

會產生如下的輸出：

```
<div>one</div>
```

比較運用 CSS 來顯示或隱藏元素的 v-show。

底下的 Vue 樣版

```
<div v-show="true">one</div>
<div v-show="false">two</div>
```

會產生如下的輸出：

```
<div>one</div>
<div style="display: none">one</div>
```

你的使用者會（或許）看到同樣的結果，但二者間還存有其他的影響與差異。

首先，因為透過 v-if 而被隱藏起來的元素並不需要顯示，Vue 並不會為它產生 HTML；但若使用的是 v-show 就不是如此。也就是說 v-if 比較適合用來隱藏還未被下載的資料。

底下的範例會拋出錯誤：

```
<div id="app">
  <div v-show="user">
    <p>User name: {{ user.name }}</p>
  </div>
</div>

<script>
  new Vue({
```

2 假值（falsy value）表其值為 false、undefined、null、0、"" 或 NaN 的值。

```
    el: '#app',
    data: {
      user: undefined
    }
  });
</script>
```

會產生錯誤的原因是 Vue 試著計算 user.name——一個不存在之物件的屬性。若在同樣的範例中改用 v-if，則程式運行正常，因為在 v-if 敘述為真前，Vue 不會試著去產生元素內的資料。

此外，還有二個條件判斷指令與 v-if 相關：即 v-else-if 與 v-else。它們的行為跟你所預期的很類似：

```
<div v-if="state === 'loading'">Loading…/div>
<div v-else-if="state === 'error'">An error occurred</div>
<div v-else>…our content!</div>
```

第一個 div 會在 state 等於 loading 時顯示，第二個則會在 state 等於 error 時顯示，而第三個會在其他情況下顯示。同一個時間點只有一個元素會被顯示出來。

好，看完這些之後，還有人會要用 v-show 嗎？

使用 v-if 會增加效能的成本。每次有元素被新增或移除時，潛藏於下的 DOM 樹都需要重新再被產生，這項工作有時會很繁重。v-show 除了初始化的成本外，並不需要再負擔這種成本。如果你預期到某些東西會頻繁地變動，則 v-show 會是比較好的選擇。

還有，若元素裡頭包含影像，則僅使用 CSS 來隱藏父層級元素可以讓瀏覽器事先下載之後將顯示的影像，也就是說，當 v-show 的值變成真時，影像立刻可以被顯示出來。若不這樣處理，影像在要被顯示之前，才會開始下載。

樣版中的迴圈

另一個我自己常用的指令是 v-for，它會對陣列或物件的元素進行循環，將元素多次輸出。請見下列範例：

```
<div id="app">
  <ul>
    <li v-for="dog in dogs">{{ dog }}</li>
  </ul>
</div>
```

```
<script>
  new Vue({
    el: '#app',
    data: {
      dogs: ['Rex', 'Rover', 'Henrietta', 'Alan']
    }
  });
</script>
```

上列程式會將陣列中的每一個項目輸出到網頁中的 list 元素中，如下：

```
<div id="app">
  <ul>
    <li>Rex</li>
    <li>Rover</li>
    <li>Henrietta</li>
    <li>Alan</li>
  </ul>
</div>
```

v-for 指令也能與物件搭配。考慮下列範例，它會接收一個內含幾個城市平均租金的物件，並將它們輸出到網頁上：

```
<div id="app">
  <ul>
    <li v-for="(rent, city) in averageRent">
      The average rent in {{ city }} is ${{ rent }}</li>
  </ul>
</div>
<script>
  new Vue({
    el: '#app',
    data: {
      averageRent: {
        london: 1650,
        paris: 1730,
        NYC: 3680
      }
    }
  });
</script>
```

因為我們也要取得鍵（key）的緣故，這裡的語法有些不同——只在網頁上顯示租金但不顯示城市的話，並沒有太大用途。不過，若我們不需要鍵，則可使用與之前相同的語法，`v-for="rent in averageRent"`。這種寫法也適用於陣列：若我們需要陣列的索引，則可使用你剛剛在陣列部分看過的括號與逗號語法：`v-for="(dog, i) in dogs"`。

要注意參數的順序：是 `(value, key)`，其他的寫法是不對的。

最後，若你只是需要一個簡單的計數器，你可以傳進一個數值作為參數。下列範例會輸出數字 1 到 10。

```
<div id="app">
  <ul>
    <li v-for="n in 10">{{ n }}</li>
  </ul>
</div>
<script>
  new Vue({
    el: '#app'
  });
</script>
```

你也許會認為輸出會是 0 到 9，但實際上並不是。要從列表的 0 開始，要寫成 `n-1` 而不是 `n`。

綁定參數

某些指令，如 `v-bind`，需要參數。`v-bind` 指令用來將一個數值綁定到一個 HTML 屬性上。比方說，底下的範例將 `submit` 這個值綁定到按鈕的類型（type）上：

```
<div id="app">
  <button v-bind:type="buttonType">Test button</button>
</div>
<script>
  new Vue({
    el: '#app',
    data: {
      buttonType: 'submit'
    }
  });
</script>
```

底下是其輸出：

```
<button type="submit">Test button</button>
```

v-bind 是指令的名稱，而 type 則是其參數：在此例中，它是我們要綁定到給定變數上的屬性（attribute）名稱。而 buttonType 是其值。

就屬性（properties）而言，這也適用，如 disabled 與 checked：若傳進來的運算式值為真，則輸出元素就會帶有屬性（property），若為否，則元素就不帶有屬性：

```
<div id="app">
  <button v-bind:disabled="buttonDisabled">Test button</button>
</div>
<script>
  new Vue({
    el: '#app',
    data: {
      buttonDisabled: true
    }
  });
</script>
```

搭配許多屬性（attributes）使用 v-bind 時，可能會需要重複寫很多次。我們可以用一種簡便的方法來寫：即可以省略指令的 v-bind，只保留冒號。比方說，你可以用短語法來重寫上述範例：

```
<div id="app">
  <button :disabled="buttonDisabled">Test button</button>
</div>
<script>
  new Vue({
    el: '#app',
    data: {
      buttonDisabled: true
    }
  });
</script>
```

這種寫法純粹是個人的偏好，我很喜歡用這種短語法來寫，很少在程式碼中寫到 v-bind。

不管你選用的是 v-bind 還是短語法，請持續下去。將 v-bind 與短語法混用會造成混淆。