

# VanillaJS to VueJS In No Time

VueJS 版本：2.6.11

## 內容

1. 前言 .....	2
2. 新觀念—Component、State 和 Props .....	3
2.1. Component(元件)和 State(狀態) .....	3
2.2. Props .....	5
3. Prerequisites .....	6
3.1. HTML .....	6
3.1.1. Tags(<h2>、<input>...等) .....	6
3.1.2. Attributes(id、class...等) .....	6
3.1.3. Bootstrap (非必要) .....	6
3.2. JavaScript 基本語法及 ES6 新語法 .....	6
3.2.1. Arrow function .....	7
3.2.2. Destructing assignment(解構賦值) .....	7
3.2.3. 陣列操作 .....	8
3.2.4. 字串操作 .....	12
3.2.5. 迴圈操作 .....	13
3.2.6. 布林值判定 .....	14
3.2.7. 表單處理 .....	15
3.2.8. Spread syntax .....	15
3.2.9. Shallow copy(淺複製)和 deep copy(深複製) .....	15
3.2.10. == 和 === 的分別 .....	16
3.2.11. null 和 undefined 的分別 .....	16
3.2.12. Object.keys 方法 .....	16
4. Component .....	17
4.1. Template .....	20

4.2.	Style.....	21
4.3.	Script .....	22
4.4.	Component attributes .....	24
4.4.1.	Name.....	24
4.4.2.	Props .....	25
4.4.3.	data.....	26
4.4.4.	Methods.....	26
4.4.5.	Computed .....	28
5.	Conditional Rendering.....	30
6.	List Rendering.....	31
7.	v-bind Directive .....	33
8.	Input 事件處理 .....	35
9.	元件註冊(Component Registration).....	38
10.	Lifting State Up.....	40
11.	Lifecycle Hooks.....	42
12.	Dynamic component(v-is Directive).....	44
13.	呼叫 API—Axios .....	45
14.	Assignment .....	45
14.1.	基本練習.....	46
14.2.	進階練習.....	51
15.	Vuex .....	53
16.	額外資訊.....	55

## 1. 前言

這本手冊能夠提供您絕大多數 **VueJS** 的基礎知識。若在學習過程中遇到任何問題，歡迎隨時提問，務必在所有問題都得到解答後才繼續進行至下一個章節。

學完之後會有一個實作練習，您將會使用 **VueJS** 來實作一個簡單的 **web app**，看起來就像下面這個樣子：

**Form A0000001**

跳至 A0000001 , 共 5 張

[Log this form](#) [Log all forms](#) [Delete this form](#)

Personal Info [Orders](#) [Overview](#)

First name  Last name

Gender ☐ Secret ☒ Male ☐ Female

Address

☐ 此客戶屬無定所

Job

Note  
0 / 2000 characters

[Previous](#) [1](#) [2](#) [3](#) [4](#) [5](#) [Next](#)

每張表單都有 3 個標籤頁，我們必須將表單的資料顯示在這些標籤頁中，並允許使用者編輯。更詳細的 **assignment** 說明，請參閱**14 Assignment**。

## 2. 新觀念—Component、State 和 Props

### 2.1. Component(元件)和 State(狀態)

在開始之前，您需要先摒棄原生 Javascript(VanillaJS)中對 DOM Manipulation 的觀念，因為在大多數的前端框架中(包含 VueJS)，使用者(就是我們)對於 HTML element 的 CRUD 方式和 VanillaJS 完全不同，我們並不需要(也不該)使用 `document.createElement` 及 `element.appendChild` 這套邏輯來操作 DOM (但是框架的底層依然是使用這套邏輯，畢竟他還是 Javascript)。舉例來說，當我們想在某個`<ol>`上新增三個`<li>`的時候：

1. Item A
2. Item B
3. Item C

(示意畫面)

若是使用 VanillaJS，我們可能會採取以下的作法：  
(方法有很多，僅以一種為例)

1. 先使用 `document.getElementById` 找到 `<ol>`
2. 再使用 `document.createElement` 建立 3 個 `<li>`
3. 最後使用 `element.appendChild` 將第二步建立的 `<li>` 放到第一步的 `<ol>` 上面去

但是在使用前端框架時(包含 VueJS)，您所需要的思維和這個完全不同。**您需要建立的是「State(狀態)」的概念**，而 state 會依附在某個東西上面，就好比是物件導向設計(OOP)當中，class 和他的屬性(properties)之間的關係，例如：

```
1. public class ol
2. {
3.     public li[] Items;
4. }
```

在這個 class 中，`ol` 就是 HTML 中的 `<ol>`，而 `Items` 就是這個 `<ol>` 上面的 `<li>`。因此如果我們想要達成「在 `<ol>` 裡面新增三個 `<li>`」的目標，以 class 的角度來看，我們只要對 `ol` class 中的 `Items` 屬性進行操作就可以了，不需要 `document.createElement`，也不需要 `element.appendChild`；**當這些屬性改變的時候，VueJS 的引擎會幫助我們即時更新頁面**，以確保我們看見的永遠是最新的結果。

然而，這些「class」在前端框架中並不叫做 class，而是叫做「**Component(元件)**」。就和 OOP 的 class 一樣，**component(class)之間的 state(properties)是各自獨立的**，就好比一個頁面可以同時有很多個 `<ol>`，在某個 `<ol>` 裡面新增 `<li>` 並不會影響到其他的 `<ol>`，這就是 component 和 state 的概念。**一個 component 就只有一個 state，但是這個 state 裡面可以包含很多屬性。**

那麼 state 在 VueJS 裡面看起來會是什麼樣子呢？以剛才 `ol` class 的範例來說，他的 state 看起來就會像是這個樣子：

```
1. {
2.     items: []
3. }
```

要特別注意的是，**在 VueJS 中，state 裡面的所有變數都應該是 VanillaJS 的原生變數類別**—Number、String、Boolean、Array、Object、Function 或是 Promise。

當我們需要其他的屬性時，只要直接在 state 裡面加上其他的變數就可以了。舉例來說，如果我們想要加入一個「MaxItemCount」的屬性，結果就會像下面這樣：

```
1. {
2.   items: [],
3.   maxItemCount: 10
4. }
```

**component** 最終會當作一個新的 **HTML tag** 來使用，直接出現在 **HTML** 裡面。例如：

```
1. <div>
2.   <h2>Hello, world</h2>
3.   <div>
4.     <h2>Order list 1</h2>
5.     <MyOrderList></MyOrderList>
6.     <h2>Order list 2</h2>
7.     <MyOrderList></MyOrderList>
8.   </div>
9. </div>
```

在上面的範例中，我們可以很快的看出來 `<MyOrderList>` 就是我們自己做出來的 **component**，因為標準的 **HTML** 並沒有 `<MyOrderList>` 這個 **tag**。我們可以把 `<MyOrderList>` 放在任何頁面的任何地方，即使同一個頁面上有很多個 `<MyOrderList>`，他們的 **state** 仍然各自獨立，不會互相影響。這種「只要寫一次就可以到處用」的特性說明了 **component 是可以被重複使用的**。

在設計頁面的時候，我們應該先決定好要將畫面切成哪些 **component**，之後再將他們一一完成，不該將所有的東西都寫在同一個檔案裡面。

那麼我們到底該怎麼存取 **component** 的 **state** 呢？以先前 `ol` 的例子來說，您的腦中可能會浮現這樣的想法：

```
1. const ol = (以某種方式拿到的 ol component);
2. ol.items.push(...);
3. ol.maxItemCount = 20;
```

這是很直觀的想法，**但這是不對的**，因為這麼做是 **anti-pattern**。在 **VueJS** 中，**component** 不該被當成變數使用，所以絕對不該出現「某個變數的類別為 **component**」這樣的事情(您可以這麼做，但您不該這麼做)。正確的 **state** 存取方式我們會在 **4.4.4 Methods** 做更詳細的說明。

## 2.2. Props

我們知道 **HTML** 裡面有一個叫做 **attribute** 的東西，可以放在 **element** 上面，例如：

```
1. <ol class="my-order-list">
2.   <!-- Put something here -->
3. </ol>
```

在上面的範例中，`class="my-order-list"`就是 attribute—`class`是他的名稱，而 `my-order-list` 就是他的值。在 VueJS 的 component 中，這樣的東西叫做 **props(properties)**。

在 HTML attribute 中，我們能傳遞的 attribute 種類不多，只有字串、數字和 function(例如：input 事件)。但是在 VueJS 中，props 可以是任何型態的數值，包括自訂的變數和方法。我們將會在 **4.4.2 Props** 做更詳細的說明。

總之，在開始學習之前，您需要謹記以下四件事：

1. 我們不會在 VueJS 中用到 VanillaJS 的 DOM Manipulation
2. Component 的概念和 OOP 中的 class 相近，有各自獨立的 state
3. Component 也有 attribute 可以用(在 VueJS 裡面稱為 props)，而且可以傳入任何類型的變數
4. Component 會以 tag 的形式出現在 HTML 中，且不該被當成變數使用

### 3. Prerequisites

雖然以下項目被稱為「Prerequisites」，但是即使您不具備這些知識，要了解 VueJS 的運作原理也不會有太大的問題。**但是要跳過這部分也可以，需要用的時候再查即可。**

#### 3.1. HTML

3.1.1. Tags(<h2>、<input>...等)

3.1.2. Attributes(id、class...等)

3.1.3. Bootstrap (非必要) (<https://getbootstrap.com/docs/4.4/getting-started/introduction/>)

#### 3.2. JavaScript 基本語法及 ES6 新語法

在現代 Javascript 中，宣告式程式設計(Declarative Programming)已經越來越常見，而原有的指令式程式設計(Imperative Programming)雖然效率通常較佳，但是由於程式碼和前者比起來增加不少，可讀性也較差，因此使用的頻率也沒有過去那麼頻繁。以陣列的 for 迴圈為例：

```
const a = [
  { id: 1, name: "A" },
  { id: 2, name: "B" },
  { id: 3, name: "C" },
  { id: 4, name: "D" }
];
```

現在有一個陣列 `a`，我們想要找出 `id > 2` 的 element，並將這些 element 的 `name` 放到新的陣列中。若使用指令式的寫法，看起來大概會是這樣：

```
const b = [];
for(let i=0; i<a.length; i++) {
  const tmp = a[i];
  if (tmp.id > 2) {
    b.push(tmp.name);
  }
}
```

但是若我們使用宣告式的寫法，看起來就會像這樣：

```
const b = a.filter(x => x.id > 2).map(x => x.name);
```

很明顯的，宣告式的寫法比指令式還要簡潔許多，且 `for` 迴圈只是其中一種情況。雖然宣告式寫法的執行效率通常不如指令式寫法好，但在大多數情況下我們仍然可以優先考慮使用宣告式的寫法，當有特殊情況時再換回指令式寫法即可。

### 3.2.1. Arrow function ([https://developer.mozilla.org/zh-TW/docs/Web/JavaScript/Reference/Functions/Arrow\\_functions](https://developer.mozilla.org/zh-TW/docs/Web/JavaScript/Reference/Functions/Arrow_functions))

在 VueJS 裡面為非必要，因為大多數情況下我們會使用一般的 function，而不是 arrow function(因為 `this` context 的問題)；但是 arrow function 在其他地方很常見，看得懂 arrow function 對我們在 google 上找答案或是學習其他框架的時候會很有幫助。

```
1. // 一般 function
2. function sum(a, b) {
3.   return a + b;
4. }
5.
6. // Arrow function
7. const sum = (a, b) => { return a + b; }
8.
9. // 或是去掉大括弧，代表這個 function 會直接 return 某個值，但是這種寫法在
   return 的時候只允許有一個 statement，範例如下：
10. const sum = (a, b) => a + b;
```

### 3.2.2. Destructuring assignment(解構賦值) ([https://developer.mozilla.org/zh-TW/docs/Web/JavaScript/Reference/Operators/Destructuring\\_assignment](https://developer.mozilla.org/zh-TW/docs/Web/JavaScript/Reference/Operators/Destructuring_assignment))

#### 3.2.2.1. 對陣列

```
1. const getValues = () => [1, true, "Hello, world"];
2. const [a, b, c] = getValues();
3. console.log(a, b, c);
```

```
4. // a = 1
5. // b = true
6. // c = "Hello, world"
7.
8. const [d,,e] = getValues();
9. console.log(d, e);
10. // d = 1
11. // e = "Hello, world"
```

### 3.2.2.2. 對物件

```
1. const adam = {
2.   firstName: "Adam",
3.   lastName: "Someone",
4.   age: 5
5. };
6. const { firstName, lastName, age } = adam;
7. // 其實就是把以下三行合成一行
8. // const firstName = adam.firstName;
9. // const lastName = adam.lastName;
10. // const age = adam.age;
```

### 3.2.3. 陣列操作

3.2.3.1. push 方法 ([https://developer.mozilla.org/zh-TW/docs/Web/JavaScript/Reference/Global\\_Objects/Array/push](https://developer.mozilla.org/zh-TW/docs/Web/JavaScript/Reference/Global_Objects/Array/push))

```
1. const a = [3, 4, 5];
2. a.push(6);
3. // a = [3, 4, 5, 6];
```

3.2.3.2. pop 方法 ([https://developer.mozilla.org/zh-TW/docs/Web/JavaScript/Reference/Global\\_Objects/Array/pop](https://developer.mozilla.org/zh-TW/docs/Web/JavaScript/Reference/Global_Objects/Array/pop))

```
1. const a = [3, 4, 5];
2. const b = a.pop();
3. // a = [3, 4];
4. // b = 5; <-- 被移除的值
```

3.2.3.3. map 方法 ([https://developer.mozilla.org/zh-TW/docs/Web/JavaScript/Reference/Global\\_Objects/Array/map](https://developer.mozilla.org/zh-TW/docs/Web/JavaScript/Reference/Global_Objects/Array/map))

```
1. const a = [3, 4, 5];
```



```

2. a.map((value, index) => {
3.     console.log(`a[${index}] = ${value}`);
4. });
5. // a[0] = 3;
6. // a[1] = 4;
7. // a[2] = 5;
8.
9. // 在這個範例中，這兩種寫法效果相同
10. for(let i=0; i<a.length; i++) {
11.     console.log(`a[${i}] = ${a[i]}`);
12. }

```

```

1. const a = [
2.     { name: "Apple", price: 10 },
3.     { name: "Banana", price: 20 },
4.     { name: "Cherry", price: 30 }
5. ];
6. const b = a.map(x => x.price);
7. // b = [10, 20, 30];

```

#### 3.2.3.4. filter 方法 ([https://developer.mozilla.org/zh-TW/docs/Web/JavaScript/Reference/Global\\_Objects/Array/filter](https://developer.mozilla.org/zh-TW/docs/Web/JavaScript/Reference/Global_Objects/Array/filter))

```

1. const a = [3, 4, 5];
2. const b = a.filter(x => x >= 4);
3. // a = [3, 4, 5];
4. // b = [4, 5];

```

```

1. const a = [
2.     { name: "Apple", price: 10 },
3.     { name: "Banana", price: 20 },
4.     { name: "Cherry", price: 30 }
5. ];
6. const b = a.filter(x => x.price > 25);
7. // b = [{ "name": "Cherry", "price": 30 }];

```

#### 3.2.3.5. sort 方法 ([https://developer.mozilla.org/zh-TW/docs/Web/JavaScript/Reference/Global\\_Objects/Array/sort](https://developer.mozilla.org/zh-TW/docs/Web/JavaScript/Reference/Global_Objects/Array/sort))

```

1. const a = [5, 4, 3];

```

```
2. const b = a.sort((x, y) => x - y);
3. // a = [3, 4, 5];
4. // b = [3, 4, 5];
```

```
1. const a = [
2.   { name: "Apple", price: 10 },
3.   { name: "Banana", price: 20 },
4.   { name: "Cherry", price: 30 }
5. ];
6. a.sort((x, y) => y.price - x.price);
7. /*
8.  a = [
9.    { name: "Cherry", price: 30 },
10.   { name: "Banana", price: 20 },
11.   { name: "Apple", price: 10 }
12. ];
13. */
```

3.2.3.6. find 方法 ([https://developer.mozilla.org/zh-TW/docs/Web/JavaScript/Reference/Global\\_Objects/Array/find](https://developer.mozilla.org/zh-TW/docs/Web/JavaScript/Reference/Global_Objects/Array/find))

```
1. const a = [3, 4, 5];
2. const b = a.find(x => x === 3);
3. // b = 3;
```

```
1. const a = [3, 4, 5];
2. const b = a.find(x => x === 9);
3. // b = undefined;
```

```
1. const a = [
2.   { name: "Apple", price: 10 },
3.   { name: "Banana", price: 20 },
4.   { name: "Cherry", price: 30 }
5. ];
6. const b = a.find(x => x.price === 20);
7. // b = { name: "Banana", price: 20 };
```

3.2.3.7. findIndex 方法 ([https://developer.mozilla.org/zh-TW/docs/Web/JavaScript/Reference/Global\\_Objects/Array/findIndex](https://developer.mozilla.org/zh-TW/docs/Web/JavaScript/Reference/Global_Objects/Array/findIndex))

```
1. const a = [3, 4, 5];
2. const b = a.findIndex(x => x === 3);
3. // b = 0;
```

```
1. const a = [3, 4, 5];
2. const b = a.findIndex(x => x === 9);
3. // b = -1;
```

```
1. const a = [
2.   { name: "Apple", price: 10 },
3.   { name: "Banana", price: 20 },
4.   { name: "Cherry", price: 30 }
5. ];
6. const b = a.findIndex(x => x.price === 20);
7. // b = 1;
```

3.2.3.8. indexOf 方法 ([https://developer.mozilla.org/zh-TW/docs/Web/JavaScript/Reference/Global\\_Objects/Array/indexOf](https://developer.mozilla.org/zh-TW/docs/Web/JavaScript/Reference/Global_Objects/Array/indexOf))

```
1. const a = [3, 4, 5];
2. const b = a.indexOf(3);
3. // b = 0;
```

```
1. const a = [3, 4, 5];
2. const b = a.indexOf(9);
3. // b = -1;
```

```
1. const a = [
2.   { name: "Apple", price: 10 },
3.   { name: "Banana", price: 20 },
4.   { name: "Cherry", price: 30 }
5. ];
6. const b = a.indexOf({ name: "Banana", price: 20 });
7. // b = -1; 說明請參閱 3.2.10 == 和 === 的分別
   (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Equality_comparisons_and_sameness)
```

3.2.3.9. splice 方法 ([https://developer.mozilla.org/zh-TW/docs/Web/JavaScript/Reference/Global\\_Objects/Array/splice](https://developer.mozilla.org/zh-TW/docs/Web/JavaScript/Reference/Global_Objects/Array/splice))

請特別注意將元素插入到指定 index 的用法。

3.2.3.10.slice 方法 ([https://developer.mozilla.org/zh-TW/docs/Web/JavaScript/Reference/Global\\_Objects/Array/slice](https://developer.mozilla.org/zh-TW/docs/Web/JavaScript/Reference/Global_Objects/Array/slice))

3.2.3.11.length 屬性 ([https://developer.mozilla.org/zh-TW/docs/Web/JavaScript/Reference/Global\\_Objects/Array/length](https://developer.mozilla.org/zh-TW/docs/Web/JavaScript/Reference/Global_Objects/Array/length))

```
1. const a = [3, 4, 5];
2. const b = a.length;
3. // b = 3;
```

### 3.2.4. 字串操作

3.2.4.1. Template literals ([https://developer.mozilla.org/zh-TW/docs/Web/JavaScript/Reference/Template\\_literals](https://developer.mozilla.org/zh-TW/docs/Web/JavaScript/Reference/Template_literals))

```
1. const firstName = "John";
2. const lastName = "Doe";
3. const fullName = `${firstName} ${lastName}`;
4. // fullName = "John Doe";
```

3.2.4.2. indexOf 方法 ([https://developer.mozilla.org/zh-TW/docs/Web/JavaScript/Reference/Global\\_Objects/String/indexOf](https://developer.mozilla.org/zh-TW/docs/Web/JavaScript/Reference/Global_Objects/String/indexOf))

```
1. const a = "I am John Doe.";
2. const b = a.indexOf("Jo");
3. // b = 5;
```

```
1. const a = "I am John Doe.";
2. const b = a.indexOf("Mary");
3. // b = -1;
```

3.2.4.3. substr 和 substring 方法

```
1. const a = "123456789";
2. const b = a.substring(3, 6);
3. const c = a.substr(3, 6);
4. // b = "456";
5. // c = "456789";
```

3.2.4.4. toLowerCase 方法

```
1. const a = "abCdEfG";
```

```
2. const b = a.toLowerCase();
3. // b = "abcdefg";
```

#### 3.2.4.5. toUpperCase 方法

```
1. const a = "abCdEfG";
2. const b = a.toUpperCase();
3. // b = "ABCDEFGG";
```

#### 3.2.4.6. includes 方法

```
1. const a = "I am John Doe.";
2. const b = a.includes("Mary");
3. // b = false;
```

#### 3.2.4.7. length 屬性

```
1. const a = "I am John Doe.";
2. const b = a.length;
3. // b = 14;
```

### 3.2.5. 迴圈操作

#### 3.2.5.1. 對陣列

```
1. const a = [3, 4, 5];
2. for(const value of a) { // for-of 的時候拿到的是 value
3.     console.log(value);
4. }
5. // 3
6. // 4
7. // 5
```

```
1. const a = [3, 4, 5];
2. for(const index in a) { // for-in 的時候拿到的是 key(陣列的 key 就是
    index)
3.     console.log(index);
4. }
5. // 0
6. // 1
7. // 2
```

#### 3.2.5.2. 對字串

```
1. const a = "ABC";
2. for(const value of a) {
3.     console.log(value);
4. }
5. // A
6. // B
7. // C
```

```
1. const a = "ABC";
2. for(const index in a) {
3.     console.log(index);
4. }
5. // 0
6. // 1
7. // 2
```

### 3.2.6. 布林值判定

#### 3.2.6.1. null 和 undefined

永遠是 `false`。

```
1. const a = Boolean(null); // false
2. const b = Boolean(undefined); // false
```

#### 3.2.6.2. 數字

只有兩種情況：

1. 是 0: `false`
2. 不是 0: `true`

```
1. const a = Boolean(-99); // true
2. const b = Boolean(-1); // true
3. const c = Boolean(0); // false
4. const d = Boolean(1); // true
5. const e = Boolean(99); // true
```

#### 3.2.6.3. 字串

只要不是空字串都會是 `true`。

```
1. const a = Boolean(""); // false
2. const b = Boolean(" "); // true
3. const c = Boolean("-1"); // true
4. const d = Boolean("0"); // true
```

```
5. const e = Boolean("1"); // true
6. const f = Boolean("abc"); // true
```

#### 3.2.6.4. 其他型別

其他型別包含 Date、陣列、物件(object)和 Function，只要不是 null 或 undefined 就會是 **true**。

```
1. const a = Boolean([]); // true
2. const b = Boolean([1, 2, 3]); // true
3. const c = Boolean({}); // true
4. const d = Boolean({ name: "value" }); // true
5. const e = Boolean(function() {}); // true
```

#### 3.2.7. 表單處理

- onInput : [https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement/input\\_event](https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement/input_event)
- onChange : [https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement/change\\_event](https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement/change_event)
- onBlur : [https://developer.mozilla.org/en-US/docs/Web/API/Element/blur\\_event](https://developer.mozilla.org/en-US/docs/Web/API/Element/blur_event)

#### 3.2.8. Spread syntax ([https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Spread\\_syntax](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Spread_syntax))

要特別注意的是 Spread syntax 只能做到淺複製(shallow copy)，無法做到深複製；詳細說明請參閱3.2.9 *Shallow copy*(淺複製)和 *deep copy*(深複製)。

##### 3.2.8.1. 對陣列

```
1. const a = [3, 4, 5];
2. const b = [...a];
3. console.log(b); // [3, 4, 5]
```

##### 3.2.8.2. 對物件

```
1. const a = { name: "Apple" };
2. const b = { ...a };
3. console.log(b); // { name: "Apple" }
```

#### 3.2.9. Shallow copy(淺複製)和 deep copy(深複製)

##### 3.2.9.1. Shallow copy(淺複製)

常見的情況出現在使用 Spread syntax 複製物件或是陣列，修改複製品中的值之後，原物件/陣列裡面的值卻一起被更改了，例如：

```
1. const adam = {
```

```

2.     pet: {
3.         name: "Diego"
4.     }
5. };
6. const ben = { ...adam };
7. ben.pet.name = "SanDiego";
8. console.log(adam.pet.name); // SanDiego
9. console.log(ben.pet.name); // SanDiego

```

或是：

```

1. const adam = {
2.     age: 15,
3.     luckyNumbers: [1, 2, 3, 4, 5]
4. };
5. const ben = { ...adam };
6. ben.luckyNumbers.pop();
7. console.log(adam.luckyNumbers); // [1, 2, 3, 4]
8. console.log(ben.luckyNumbers); // [1, 2, 3, 4]

```

淺複製只會複製第一層的值，第二層之後就會參照到原物件。若您需要一個全新的物件，您應該使用深複製而不是淺複製。

### 3.2.9.2. Deep copy(深複製)

您可以實作自己的深複製方法，或是使用 lodash 套件的 cloneDeep 方法。

### 3.2.10.== 和 === 的分別 ([https://developer.mozilla.org/en-US/docs/Web/JavaScript/Equality\\_comparisons\\_and\\_sameness](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Equality_comparisons_and_sameness))

絕大部分的情況下直接使用嚴格相等(===，三個等於)即可。

### 3.2.11. null 和 undefined 的分別

#### 3.2.11.1. 在一般相等(==)的情況下

```
console.log(null == undefined); // true
```

#### 3.2.11.2. 在一般相等(==)的情況下

```
console.log(null === undefined); // false
```

### 3.2.12. Object.keys 方法

#### 3.2.12.1. 陣列

```

1. const a = ["b", "c", "d"];
2. console.log(Object.keys(a)); // [0, 1, 2]

```



### 3.2.12.2. 物件

```
1. const a = {  
2.   name: "Apple",  
3.   price: 50  
4. };  
5. console.log(Object.keys(a)); // ["name", "price"]
```

### 3.2.12.3. 字串

```
1. const a = "apple";  
2. console.log(Object.keys(a)); // [0, 1, 2, 3, 4]
```

### 3.2.12.4. 其他型別

除了 null 和 undefined 會出錯之外全部都是空字串。

```
1. const boolean = true;  
2. console.log(Object.keys(boolean)); // []  
3.  
4. const number = 5;  
5. console.log(Object.keys(number)); // []  
6.  
7. const date = new Date();  
8. console.log(Object.keys(date)); // []  
9.  
10. const _function = function() {};  
11. console.log(Object.keys(_function)); // []  
12.  
13. console.log(Object.keys(null)); // Error: Cannot convert undefined or null to object  
14. console.log(Object.keys(undefined)); // Error: Cannot convert undefined or null to object
```

## 4. Component (<https://vuejs.org/v2/guide/components.html>)

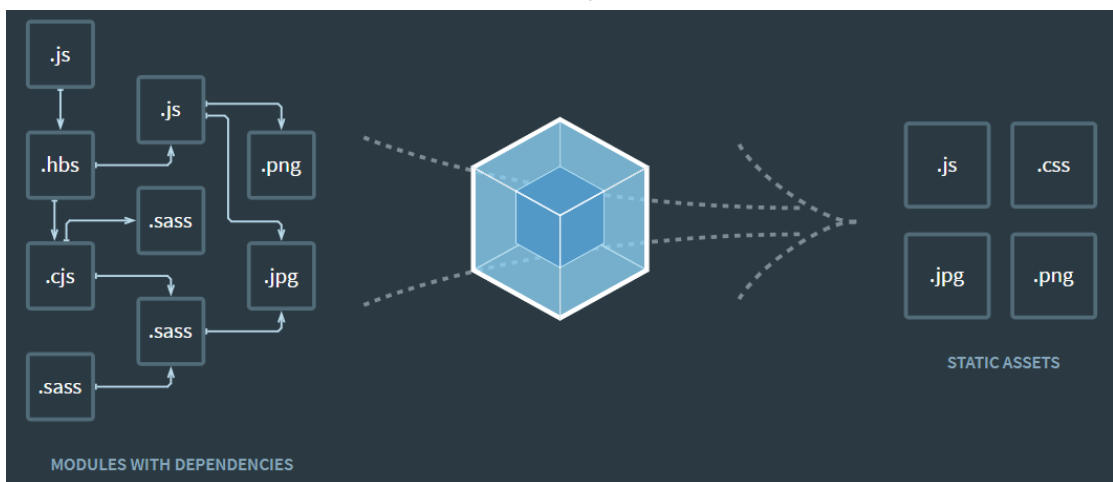
在**2.1 Component(元件)和 State(狀態)** 中，我們已經對 component 有了初步的認識。在這一個章節中，我們將會了解如何從頭創造出一個 component 並將它顯示在頁面上，也會學習如何使用 component 的 props 和 state。

Sandbox 名稱	Sandbox 連結
------------	------------

Component Basic	<a href="https://codesandbox.io/s/component-basic-gzk31?fontsize=14&amp;hiddenavigation=1&amp;module=%2Fsrc%2Fmain.js&amp;theme=dark">https://codesandbox.io/s/component-basic-gzk31?fontsize=14&amp;hiddenavigation=1&amp;module=%2Fsrc%2Fmain.js&amp;theme=dark</a>
-----------------	---

(備註 1：每當您看到 Sandbox 連結，就代表接下來即將進入實作環節。在實作的過程中若遇到任何問題，也歡迎隨時提問；請務必確定自己已經完全了解該章節的內容，而且實作上也沒有問題之後再進行至下一個章節。)

(備註 2：我們在 Sandbox 上編寫的程式碼會經過 webpack 編譯、打包，將程式碼轉換成瀏覽器看得懂的內容(純 HTML, Javascript, CSS)，這也就是為什麼我們寫的 VueJS 程式碼可以被瀏覽器執行，而 CodeSandbox...等 Online IDE 也是如此。)



在開始介紹 component 之前，我們先來看看 `src/main.js`：

- 檔案：`src/main.js`

```

Files
└─ public
  └─ src
    └─ components
      └─ App.vue
        └─ main.js
          └─ package.json

JS main.js x
1  import Vue from "vue";
2  import App from "./App.vue";
3
4  new Vue({
5    render: h => h(App)
6  }).$mount("#app");
7

```

`src/main.js` 是這個程式的 entry point(進入點)，我們主要做了兩件事情：

1. 匯入 `App` (`src/App.vue`)

```
import App from "./App.vue";
```

這裡的 `App` 指的是一個 component，通常 VueJS component 檔案的副檔名會是 `.vue`。

2. 將 `App` 渲染(render)到 `id` 為 `app` 的 HTML element 上面

```

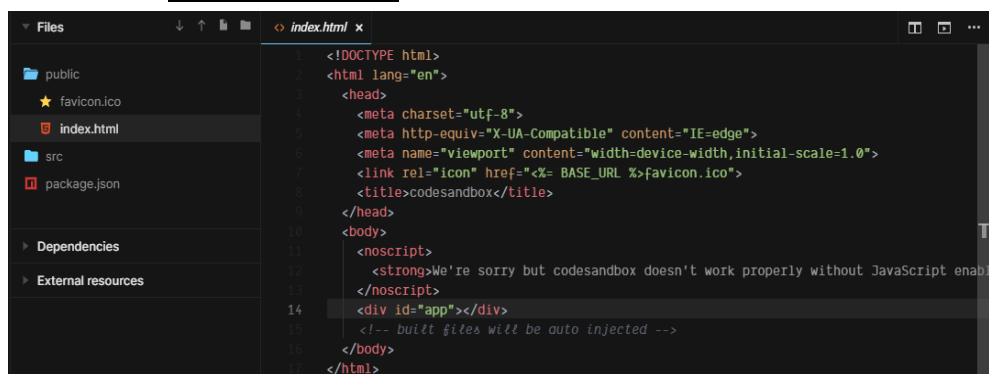
1. new Vue({
2.   render: h => h(App)

```

3. `}).$mount("#app");` // #app = (id 為 app 的 HTML element)

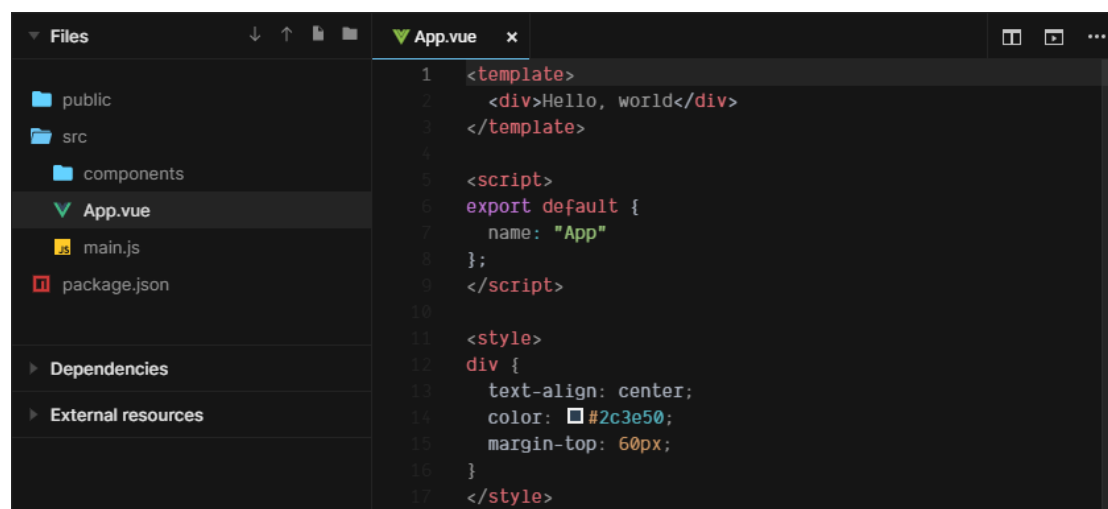
而這個 `id` 為 `app` 的 HTML element 可以在 `public/index.html` 裡面找到。中間的 `render: h => h(App)` 為 VueJS 原生方法(每個 component 都有，通常不寫)，可以不必理會，這個手冊並不會介紹他。

● 檔案：`public/index.html`



如此一來，VueJS 就會幫我們把 `App` 裡面的東西都 render 到 `<div id="app"></div>` 底下去。因此，當我們想要在頁面上顯示任何東西的時候，我們只要修改 `App` 就可以了，不需要再對這個 entry point 的 render function 做任何事情；所以，**App 會是所有 component 共同的根源；同時，App 自己也是一個 component**。接下來，讓我們打開 `src/App.vue`：

● 檔案：`src/App.vue`



一個 component 由 2~3 個區塊組成：

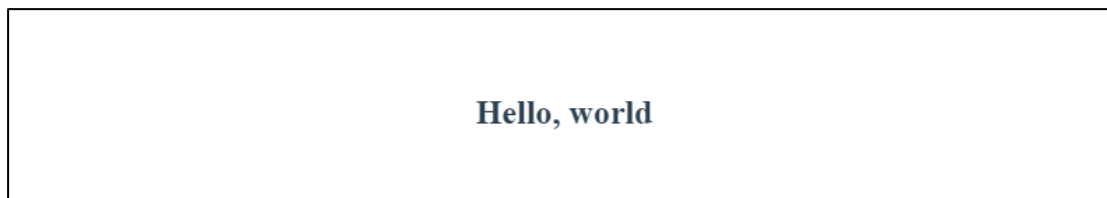
1. Template(整體 layout，就是 HTML)
2. Script(運作邏輯，包含我們在 **2 新觀念-Component、State 和 Props** 中介紹的 state 和 props)
3. Style(CSS，僅 SFC 有)

這種將 `<template>`，`<script>` 和 `<style>` 區分成三塊並放到同一個檔案裡面的 component 稱為「**Single File Component(SFC)**」，而且 **SFC 只有在副檔名為 .vue**

的檔案中才能被解析；同時，一個檔案只能一次宣告一個 **SFC**。以下將分別對這三個區塊進行說明。

#### 4.1. Template

目前的畫面看起來會像是這個樣子：



如果和 `App` 的程式碼對照著看，我們不難發現其實 `<template>` 區塊就是一個 **component** 的 **layout(HTML)**，寫法就和一般的 HTML 一樣。

需要注意的是，一個 **component** 的根元素(**root element**)最多只能有一個。意思就是說，以下的範例是錯誤的：

```
1. <template>
2.   <div>Hello, world</div> <!-- 這是第一個 root element -->
3.   <div>Hello, world, again</div> <!-- 這是第二個 root element -->
4. </template>
```

在這個範例中，`<template>` 的底下出現了兩個 `<div>`，這是不對的，因為一個 **component** 的 **root element** 只能有一個。如果我們將他修改成以下的樣子，程式就能順利執行：

```
1. <template>
2.   <div> <!-- 這是第一個 root element -->
3.     Hello, world
4.   <div>Hello, world, again</div> <!-- 他不是 root element -->
5. </div>
6. </template>
```

更多關於 **root element** 的情況，請參考以下範例：

```
1. // Example A: 正確，只有 <div> 一個 root element
2. <template>
3.   <div>Hello, world</div>
4. </template>
5.
6. // Example B: 正確，只有 <div> 一個 root element
7. <template>
8.   <div>
9.     Hello, world
10.    <label>Hi</label>
```

```

11.     </div>
12. </template>
13.
14. // Example C: 正確，雖然裡面一個 element 都沒有
15. <template>
16. </template>
17.
18. // Example D: 錯誤，雖然裡面有文字，但是文字不是一個 element
19. <template>
20.     Hello, world
21. </template>
22.
23. // Example E: 錯誤，因為有 <div> 和 <span> 兩個 root element
24. <template>
25.     <div>Hello</div>
26.     <span>world</span>
27. </template>
28.
29. // Example F: 錯誤，因為文字不可以直接出現在 template 中，必須包在某個
    element 底下
30. <template>
31.     Hello<span>world</span>
32. </template>

```

## 4.2. Style

在介紹 `<script>` 區塊之前，我們先來介紹較簡單的 `<style>` 區塊。顧名思義，`<style>` 區塊放的就是 component 的 style，就和一般的 CSS 一樣，例如：

```

1. <style>
2. h2 {
3.     color: red;
4. }
5. </style>

```

但是，這樣的寫法會讓整個網頁上所有的 `<h2>` 都套用到這個 `<style>`。在 VueJS 中，我們可以替 `<style>` 加上 `scoped` 屬性，如此一來就能來讓 `<style>` 只作用在其所屬的 component 上，就像這樣：

```

1. <style scoped> <!-- 在這裡加上 scoped -->
2. h2 {
3.     color: red;

```

```
4.  }
5.  </style>
```

在加上 `scoped` 屬性之前，頁面渲染出來的 HTML 會是這樣：

```
1.  <div >
2.    <h2>Hello, world</h2>
3.  </div>
```

當我們在 `<style>` 加上 `scoped` 屬性之後，頁面渲染出來的 HTML 看起來會像是這樣：

```
4.  <div data-v-763db97b="">
5.    <h2 data-v-763db97b="">Hello, world</h2>
6.  </div>
```

我們可以看見，這個 component 中的所有 element(無論他是否有出現在 `<style>` 的規則中)都被加上了 `data-v-763db97b=""` 的屬性，正是因為這個屬性，我們的 `<style>` 才可以套用到正確的 element 上面；而 `data-v-[id]` 的 `[id]` 則是一個唯一的隨機值。

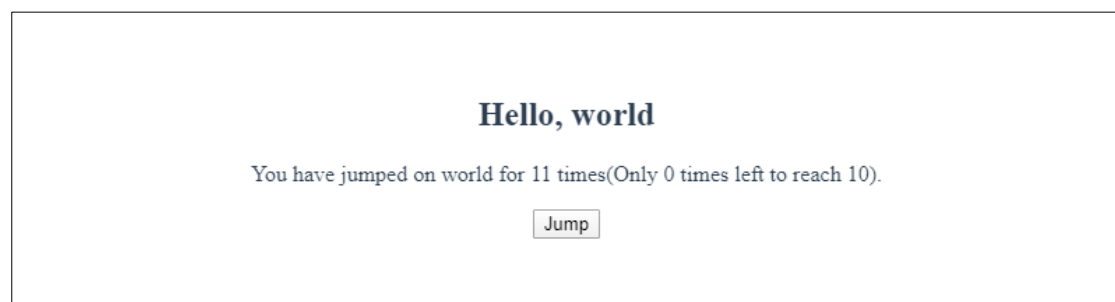
另外，`<style>` 區塊並非必要；可以一個都沒有，也可以有很多個。我們也可以在 `<style>` 上面加上 `lang` 屬性來指定我們要使用什麼語法(less, scss...等)，就像這樣：

```
1.  <style lang="scss" scoped>
2.    h2 {
3.      color: red;
4.    }
5.  </style>
```

這樣一來，我們就可以在 `<style>` 中使用 scss 的語法(前提是您的環境可以解析這些 syntax)。

### 4.3. Script

在開始介紹 `<script>` 區塊之前，我們需要先建立一個新的 component—`HelloWorldComponent`。他看起來就像下面這個樣子：



畫面上有一個 `Jump` 按鈕，每點擊一次，畫面上的計數器就會跟著+1(用到 `state` 和 `methods`)；同時也會顯示還要跳幾次才會到 10 下(用到 `computed`)。

我們希望可以在 `App` 裡面以 HTML tag 的方式顯示這個 component，例如：

```
<HelloWorldComponent worldName="MyWorld" />
```

當我們指定 `worldName` 屬性為 `MyWorld` 的時候，畫面上能要顯示出正確的結果(用到 **props**)。

要達成這幾個目的，我們需要用到四項功能：**state**、**methods**、**computed** 和 **props**。

請打開 `src/components/HelloWorldComponent.vue` 並將以下的程式碼貼到裡面去：

- 檔案：`src/components/ HelloWorldComponent.vue`

```
<template>
  <div class="hello-world-component">
    <h2>Hello, {{ worldName }}</h2>
    <div>You have jumped on {{ worldName }} for {{ jumpCount }} times(Only {{ remainingJumpCount }} times left to reach 10).</div>
    <button type="button" v-on:click="handleJumpClick">Jump</button>
  </div>
</template>

<script>
export default {
  name: "HelloWorldComponent",
  props: {
    worldName: {
      type: String,
      required: false,
      default: () => "world"
    }
  },
  data() {
    return {
      jumpCount: 0
    };
  },
  computed: {
    remainingJumpCount() {
      const count = (10 - this.$data.jumpCount);
      return (count <= 0) ? 0 : count;
    }
  }
}
```

```

    },
    methods: {
      handleJumpClick() {
        this.$data.jumpCount++;
      }
    }
  };
</script>

<style lang="scss" scoped>
.hello-world-component {
  text-align: center;
  & > button {
    display: block;
    margin: 0 auto;
    margin-top: 1rem;
  }
}
</style>

```

在修改完檔案之後，請試著進行以下操作：

1. 打開 `src/App.vue`，在 `<template>` 區塊裡找到 `<HelloWorldComponent />`，並新增一個 `worldName` attribute 給他，例如：  
`<HelloWorldComponent worldName="MyWorld" />`，並觀察頁面的變化。
2. 點擊畫面上的 Jump 按鈕，觀察頁面的變化。

操作完成後請試著閱讀 `HelloWorldComponent` 的程式碼；有些地方不了解也沒關係，只要對這個 component 有個粗略的概觀即可。

#### 4.4. Component attributes

在操作過 `HelloWorldComponent` 之後，我們就來看看這個 component 裡面到底做了哪些事情，了解他是如何運作的。首先，我們來看看 `HelloWorldComponent` 的 `<script>` 區塊中各個屬性。

##### 4.4.1. Name

`name` 屬性代表 component 的名字，沒有這個屬性 component 也可以正常運作，他在全域註冊(global registration)及遞迴情況下會用到，但是目前不需要了解；總之最好每個 component 都有這個屬性(雖然非必要)，而且盡量讓他和檔案名稱相同就對了。



#### 4.4.2. Props

在**2.2 Props** 中，我們已經大概知道 props 的作用為何。Props 的定義有很多種方法，這裡只會用最有利於閱讀/理解的一種來當例子。詳細的定義方法請參閱官方網站 <https://vuejs.org/v2/guide/components-props.html>。

Props 是一個物件，每個 property 除了名字之外還包含三個屬性：type(類別)、required(是否必要)和 default(預設值)，就像下面這個 `worldName` property：

```
1. worldName: {  
2.   type: String,  
3.   default: "world",  
4.   required: false  
5. }
```

1. Type(類別)

設定 property 的類別，可用的值有 `Number`、`Boolean`、`String`、`Array`、`Object`、`Function` 和 `Promise`。

2. Default(預設值)

可省略，預設值為 `undefined`。需要注意的是，當這個 property 的 type 是 `Array` 或是 `Object` 的時候，如果要提供預設值，那麼他就必須是一個 `function` (為了避免多個 component 共用一個物件的情況)，例如：

```
1. something: {  
2.   type: Array,  
3.   default: () => [], // 因為這裡沒有用到 this，所以可以用 arrow  
   function；您也可以寫成 default: function() { return []; }  
4.   required: false  
5. }
```

3. Required(是否必要)

可省略，預設值為 `false`。如果設為必要但卻沒有提供給他，那麼程式就會報錯。

每個 property 的格式都相同，我們可以依照上述的規則在 props 裡面依我們的需求加入 property。

那麼我們是如何將這個 `worldName` property 顯示在頁面上的呢？若您回頭看看 `<template>` 的內容，您會看見像是這樣的語法：

```
<h2>Hello, {{ worldName }}</h2>
```

在 VueJS 的 `<template>` 中，兩個大括號裡面的文字(例如：`{{ worldName }}`)都會被當成是 **Javascript** 來處理；所以如果我們在裡面寫下這樣的文字：

```
<h2>{{ 10 * 10 }}</h2>
```

那麼頁面上自然就會出現 100，這種語法被稱為「Mustache syntax(Double curly braces)」。不僅 props 可以用這種方法來顯示，**所有在 component 裡面的變數都可以這樣使用**。所有類型的變數在兩個大括弧內都會使用 `.toString()` 方法，以字串的形式顯示出來，並且會在他的數值更新時即時刷新頁面，讓我們看到的永遠是最新的結果；**只要 props 和 data 裡面的屬性發生變化，就會觸發該 component 的畫面刷新機制**。

需要注意的是，**這個雙括弧區域只能顯示變數或是呼叫 function，無法宣告新變數**。雖然 VueJS 在功能上允許我們可以使用雙括弧語法直接在 `<template>` 裡面進行一些簡單的運算，但是這麼做除了程式碼不一致之外，也可能會有效能問題(因為隨著頁面不斷刷新，這些數值也會不斷地被重複計算)，因此還是建議將需要運算的東西都移到 `<script>` 裡面，除非這些運算真的很簡單(例如：僅判斷 `true` 或 `false`)。我們將會在 7 v-bind Directive 針對 props 的使用方式做更詳細的介紹。

#### 4.4.3. data

在 2.1 Component(元件)和 State(狀態) 中，我們介紹了 state 的概念，而 data 就是 VueJS component 中的 state。Data 的定義比 props 簡單許多，只要直接以物件形式把各個屬性的值放上去就可以了，例如：

```
1. data() {  
2.     return {  
3.         jumpCount: 0  
4.     };  
5. }
```

需要注意的是，`data` 他自己並不是一個物件，而是一個 **function**；他回傳的東西才是一個物件(`data` 屬性是 function 的原因和 props 相同，就是為了避免多個 component 共用一個物件的情況出現)。就和 props 一樣，我們也可以使用雙括弧語法將 `jumpCount` 這個變數顯示在頁面上，例如：

```
<div>You have jumped {{ jumpCount }} times</div>
```

#### 4.4.4. Methods

顧名思義，**methods 包含一個 component 所有的方法**。舉例來說，如果在 OOP 中我們有這樣一個 class：

```
1. public class Person  
2. {  
3.     public int Age = 0;  
4.  
5.     public void GrowUp()  
6.     {
```

```
7.     this.Age++;
8.   }
9. }
```

如果我們把上面的 class 轉成 component 的樣子，看起來就會像這樣：

```
1. export default {
2.   data() {
3.     return {
4.       age: 0
5.     };
6.   },
7.   methods: {
8.     growUp() {
9.       this.$data.age++;
10.    }
11.  }
12. }
```

**Methods** 會是一個物件，裡面放了所有這個 **component** 會使用的方法(絕大多數和修改 **data** 有關)。您可能已經注意到，如果需要存取的屬性出現在 **data** 裡面，我們就可以使用 **this.\$data.變數名稱** 的方式來存取他；如果需要存取的屬性出現在 **props** 裡面，我們就可以使用 **this.\$props.變數名稱** 的方式來存取他。

但是 **this** 後方的 **\$data** 和 **\$props** 其實並非必要。在上面的例子中，如果我們想要存取 **data** 裡面的 **age** 屬性，除了 **this.\$data.age** 之外，我們也可以用 **this.age** 來存取他，而 **props** 裡面的屬性也一樣。

那麼如果 **props** 和 **data** 同時都有 **age** 屬性，這時候 **this.age** 會拿到哪一個呢？在 **VueJS** 中，這種情況是不被允許的。在 **component** 裡面，**props**、**data**、**computed** 和 **methods** 四個區塊屬性的 **key** 都是不可以重覆的，例如：

```
1. props: {
2.   jumpCount: {
3.     type: Number
4.   }
5. },
6. data() {
7.   return {
8.     jumpCount: 0
9.   };
10. },
11. methods: {
```

```

12.   jumpCount() {
13.       return 5;
14.   }
15. }

```

在上面的範例中，`props`、`data` 和 `methods` 都各有一個 `jumpCount` 屬性，這將會導致程式報錯。

讓我們回到上面的 `HelloWorldComponent`。我們的目的是「在按下 Jump 按鈕的時候，就讓 `data` 裡面的 `jumpCount+1`」。所以我們可以寫出這樣一個方法：

```

1.   handleJumpClick() {
2.       this.$data.jumpCount++;
3.   }

```

這個方法做的事情很簡單，就是讓 `data` 裡面的 `jumpCount+1`。在 VueJS 中，**state(就是 data)的值是可以直接修改的，但是 props 不行(其實可以，但是您不該這麼做)**。請千萬要記得，**props 的值不該由 child component 來修改；只該由來源 component 自己來修改**。意思就是說，以下的程式碼是不該出現在您的程式中的：

```
this.$props.something = value;
```

因為 `props` 通常是 parent component 的 `data`，為了要保持 single source of truth，應該只有來源 component 可以修改他們。

在 `methods` 寫好了之後，我們還需要把他綁到我們的按鈕上。在 VueJS 中，事件(event)的綁定語法為 `v-on:事件名稱="方法名稱"`，例如：

```
<button type="button" v-on:click="handleJumpClick">Jump</button>
```

VueJS 也提供了事件綁定的縮寫，使用 `@` 取代 `v-on:`，例如：

```
<button type="button" @click="handleJumpClick">Jump</button>
```

綁定了之後，當我們按下 Jump 按鈕時，他就會來呼叫 `methods` 裡的 `handleJumpClick` 方法，在 `data` 裡面的 `jumpCount` 屬性也就會因此+1。

另外，我們當然也可以在 component 裡的任何一個地方呼叫這些 `methods`；只要使用 `this.方法名稱` 就可以呼叫到這些 `methods` 了。針對 VueJS 的事件，我們將會在**錯誤! 找不到參照來源。 錯誤! 找不到參照來源**。做更詳細的說明。

#### 4.4.5. Computed

**Computed 的功用是將 state 和 props 做運算，完成之後放到一個新的變數裡面**。以上面 `HelloWorldComponent` 例子來說，我們現在需要一個 `computed` 屬性，他回傳的是「還要跳幾次才會到 10 下」。因此，我們可以寫出這樣一個方法：

```
1.   remainingJumpCount() {
```

```
2.     const count = (10 - this.$data.jumpCount);
3.     return (count <= 0) ? 0 : count;
4. }
```

就和 methods 一樣，**computed** 也是一個物件，裡面放了很多屬性，這些 **computed** 屬性都會是 **function**(因為他們是 **getter**)；而且在運算的過程中，我們絕對不該修改任何一個 **state** 或是 **props**(因為 **computed** 屬性是 **getter**)。

雖然他在 **computed** 裡面是以 **function** 的形式出現，但是使用時必須直接當成變數來使用；我們沒有辦法把它當成 **function** 來呼叫(因為他是 **getter**)。如果要在頁面上顯示 **computed** 屬性，我們同樣可以在 `<template>` 裡面使用兩個大括號來達成，例如：

```
<div> Only {{ remainingJumpCount }} times left to reach 100</div>
```

另外，如果我們要在 component 的其他地方(例如：methods 裡面)存取這些 **computed** 屬性，我們只要使用 `this.屬性名稱` 就可以了。

---

### ※額外資訊—善用 **computed** 屬性可以改善效能

**computed** 屬性在計算完成之後會被暫存起來，直到裡面的 **dependencies**(就是指裡面用到的 **props** 和 **data**)改變之後才會再重新計算一次。以上面 `remainingJumpCount` 的例子來說，直到 `this.$data.jumpCount` 的值發生變化之前(使用嚴格相等來判斷)，`remainingJumpCount` 都不會重新計算，所以善用 **computed** 屬性的確是可以改善程式效能的。

---

### ※額外資訊 2—methods 和 **computed** 在某些情況下似乎可以做到一樣的事情？

其實 methods 和 **computed** 在某些情況下可以做到一樣的事情。以剛才的 `remainingJumpCount` 來說，我也可以在 methods 裡面自訂一個 `getRemainingJumpCount` 方法，同樣讓他回傳「還要跳幾次才會到 100 下」，例如：

```
1. methods: {
2.   getRemainingJumpCount() {
3.     const count = (100 - this.$data.jumpCount);
4.     return (count <= 0) ? 0 : count;
5.   }
6. }
```

然後在 `<template>` 裡面，我可以這麼做：

```
<div> Only {{ getRemainingJumpCount() }} times left to reach 100</div>
```

在這種情況下，methods 和 **computed** 的確可以做到一樣的事情。那麼我們到底該用哪一個呢？如果使用 methods 的作法，那麼在每次頁面刷新時，無論

`this.$data.jumpCount` 有沒有變更，`getRemainingJumpCount` 都會被重新執行一次。因此針對這種需要自動計算的數值，應以 **computed** 為優先做法(在絕大部分的情況下是這樣)。

## 5. Conditional Rendering

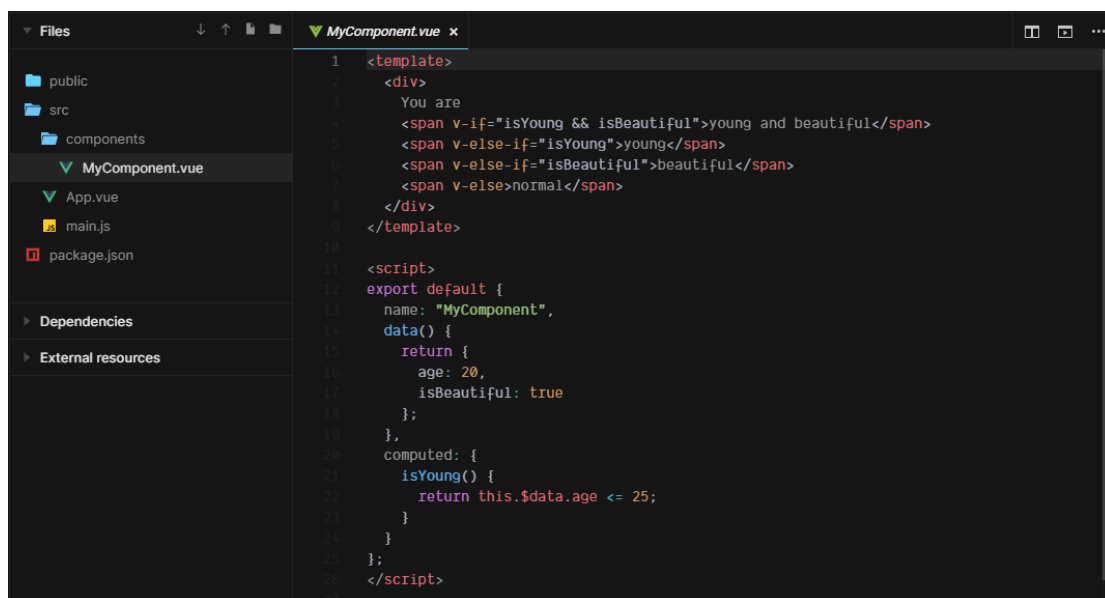
在這個章節中，我們將會學習如何在 `<template>` 裡面使用 if-else，就像我們在寫程式的時候那樣。

Sandbox 名稱	Sandbox 連結
Conditional Rendering	<a href="https://codesandbox.io/s/conditional-rendering-khw2t?fontsize=14&amp;hidenavigation=1&amp;module=%2Fsrc%2Fcomponents%2FMyComponent.vue&amp;theme=dark">https://codesandbox.io/s/conditional-rendering-khw2t?fontsize=14&amp;hidenavigation=1&amp;module=%2Fsrc%2Fcomponents%2FMyComponent.vue&amp;theme=dark</a>

(備註：每當您看到 Sandbox 連結，就代表接下來即將進入實作環節。在實作的過程中若遇到任何問題，也歡迎隨時提問；請務必確定自己已經完全了解該章節的內容，而且實作上也沒有問題之後再進行至下一個章節。)

讓我們直接看看 `src/components/MyComponent.vue`。

- 檔案：`src/components/MyComponent.vue`



```
1 <template>
2   <div>
3     You are
4     <span v-if="isYoung && isBeautiful">young and beautiful</span>
5     <span v-else-if="isYoung">young</span>
6     <span v-else-if="isBeautiful">beautiful</span>
7     <span v-else>normal</span>
8   </div>
9 </template>
10
11 <script>
12 export default {
13   name: "MyComponent",
14   data() {
15     return {
16       age: 20,
17       isBeautiful: true
18     };
19   },
20   computed: {
21     isYoung() {
22       return this.$data.age <= 25;
23     }
24   }
25 };
26 </script>
```

相信要讀懂這個 component 不是太難。在 `<template>` 區塊中，我們在 `<span>` 上面看到了 `v-if`、`v-else-if` 和 `v-else` 這些屬性，他們的功能就跟程式語言的 if-else 一樣。這些東西被稱為「**Directive(指令)**」——就是只能用在 `<template>` 的 **element** 上的自訂語法(**HTML element 及 component 皆可使用**)，`v-if`、`v-else-if` 和 `v-else` 只是 VueJS 預設 directive 中的其中幾個。

除了 `v-if` 之外，還有另一個 `v-show` directive，它的功能和 `v-if` 一樣，可以依照提供的布林值顯示/隱藏 element(但是 `v-show` 沒有 if-else 的邏輯可以用)。和 `v-if` 不同的是，`v-show` 在隱藏 element 時是使用 `css` 的方法 (`display: none`)，所以 `DOM element` 會一直存在於頁面上；而 `v-if` 的作法則是移除整個 `DOM element`。因此，對於常切換的內容，我們建議使用 `v-show`；至於不常切換(或是無法切換)的內容，我們則推薦使用 `v-if`。

更多關於 conditional rendering 的相關資訊請參閱官方文件 <https://vuejs.org/v2/guide/conditional.html>。

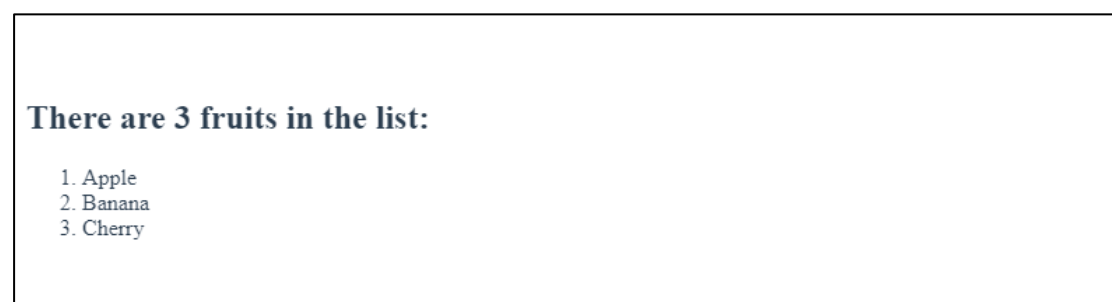
## 6. List Rendering

在這個章節中，我們將會學習如何將陣列變數顯示在畫面上。

Sandbox 名稱	Sandbox 連結
List Rendering	<a href="https://codesandbox.io/s/list-rendering-t3cw4?fontsize=14&amp;hidnavigation=1&amp;module=%2Fsrc%2Fcomponents%2FMyComponent.vue&amp;theme=dark">https://codesandbox.io/s/list-rendering-t3cw4?fontsize=14&amp;hidnavigation=1&amp;module=%2Fsrc%2Fcomponents%2FMyComponent.vue&amp;theme=dark</a>

(備註：每當您看到 Sandbox 連結，就代表接下來即將進入實作環節。在實作的過程中若遇到任何問題，也歡迎隨時提問；請務必確定自己已經完全了解該章節的內容，而且實作上也沒有問題之後再進行至下一個章節。)

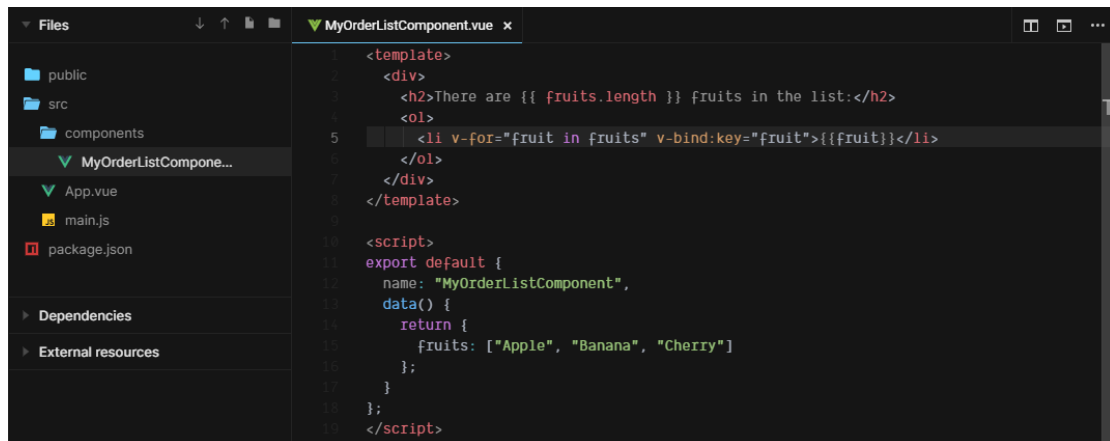
在畫面中，我們看見了一個 `<ol>`，裡面有三個 `<li>`。



我們的目的是把 `data` 裡的字串陣列——`fruits` 轉換成 `<li>` 顯示在頁面上，這裡使用了另一個 directive——`v-for`。

- 檔案：`src/components/ MyComponent.vue`





```
1 <template>
2   <div>
3     <h2>There are {{ fruits.length }} fruits in the list:</h2>
4     <ol>
5       <li v-for="fruit in fruits" v-bind:key="fruit">{{fruit}}</li>
6     </ol>
7   </div>
8 </template>
9
10 <script>
11 export default {
12   name: "MyOrderListComponent",
13   data() {
14     return {
15       fruits: ["Apple", "Banana", "Cherry"]
16     };
17   }
18 };
19 </script>
```

除了 `v-for` 之外，我們還看見了一個特別的屬性——`v-bind:key`。

```
<li v-for="fruit of fruits" v-bind:key="fruit">{{ fruit }}</li>
```

`v-for` 的功能就像是 Javascript 中的 `for`。如果我們把範例中的 `v-for` 轉換成 Javascript，那麼看起來就會像是這個樣子：

```
1. for(const fruit of this.$data.fruits) {
2.   // Do something
3. }
```

由於 `v-for` 是套用在 HTML element 或 component 上的，他代表的意思自然就是「針對提供的變數做 **for 迴圈**，並產生出相對應個數的 **HTML element 或 component**」。因此，以上面的範例來說，如果 `fruits` 陣列中有 10 個 element，那麼畫面上就會出現 10 個 `<li>`。另外，`v-for="fruit of fruits"` 中的 `of` 也可以換成 `in`，兩者的效果相同(但是在 Javascript 迴圈中的 `of` 和 `in` 就不同了)。

至於 `v-bind:key` 則是用來讓 VueJS 辨別 DOM nodes 用的值。`key` 的運作原理目前不需要知道，不過一定要記得**只要使用 `v-for` 就一定要配上 `v-bind:key`**，而且每個 **element 的 key 都不可重覆**(請盡量使用簡單的數字或是字串來當 **key**)，否則頁面 render 出來的東西就會不正常。而 `v-bind` 也是一個 **directive**，我們會在**錯誤! 找不到參照來源**。**錯誤! 找不到參照來源**。做更詳細的說明。

另外，**雖然使用元素的 index 來當作 key 看起來沒有什麼問題，但這其實不是個好作法，因為他在某些情況下仍然會導致 render 結果不如預期**；所以若非必要，我們應該盡量避免這種情況。

當我們的陣列從字串陣列變成較複雜的物件陣列時，相同的邏輯也能運作。假設目前的 `fruits` 變成下面這個樣子：

```
1. fruits: [
2.   { id: 0, name: "Apple" },
3.   { id: 1, name: "Banana" },
4.   { id: 2, name: "Cherry" },
```



5. ]

通常我們會以 element 的 `id` 來當作 key；但是在這個情況中，由於三個 element 的 `name` 都沒有重覆，所以挑選哪一個屬性來當 key 都可以。因此，以下範例同樣會 render 出所有的 `fruit`：

```
<li v-for="fruit of fruits" v-bind:key="fruit.id">{{ fruit.name }}</li>
```

此外，`v-for` 也提供 `index` 讓我們存取，例如：

```
<li v-for="(item, index) in items" v-bind:key="item">{{ item }}</li>
```

最後，使用 `v-for` 時有個常見的錯誤——將 `v-for` 用在已經含有 `v-if` 或 `v-show` 的 element 上，例如：

```
1. <li
2.     v-for="fruit in fruits"
3.     v-bind:key="fruit"
4.     v-if="fruit.id > 2"> <!-- id 大於 2 的 fruit 才顯示 -->
5.     {{ fruit.text }}
6. </li>
```

雖然乍看之下沒什麼問題，但是這是不對的。我們應該把 `id` 大於 2 的 `fruits` 移到 computed 屬性當中，然後用那一個 computed value 來使用 `v-for`。關於更多 `v-for` 的錯誤使用案例，請參閱官方文件 <https://vuejs.org/v2/style-guide/#Avoid-v-if-with-v-for-essential>。

## 7. v-bind Directive

在這個章節中，我們將會學習如何使用 `v-bind directive` 來將屬性綁定到 HTML element 或是 component 上。

Sandbox 名稱	Sandbox 連結
v-bind Directive	<a href="https://codesandbox.io/s/v-bind-directive-tunpr?fontsize=14&amp;hiddenavigation=1&amp;module=%2Fsrc%2F%2Fcomponents%2FMyComponent.vue&amp;theme=dark">https://codesandbox.io/s/v-bind-directive-tunpr?fontsize=14&amp;hiddenavigation=1&amp;module=%2Fsrc%2F%2Fcomponents%2FMyComponent.vue&amp;theme=dark</a>

(備註：每當您看到 Sandbox 連結，就代表接下來即將進入實作環節。在實作的過程中若遇到任何問題，也歡迎隨時提問；請務必確定自己已經完全了解該章節的內容，而且實作上也沒有問題之後再進行至下一個章節。)

開啟 Sandbox 後，您應該會看見以下畫面：

There are 4 items

1. Apple
2. Banana
3. Cherry
4. Durian

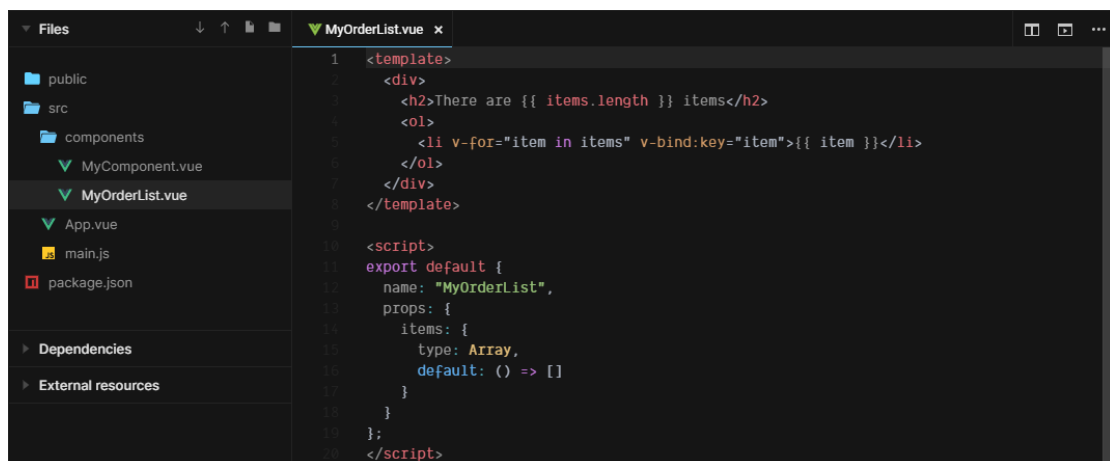
First fruit:

請試著點擊畫面上的按鈕，並觀察以下兩點：

1. `<ol>`之中項目的變化
2. `<input>`內數值的變化

在進入 `MyComponent` 之前，讓我們先來看看 `MyOrderList`。

- 檔案：`src/components/ MyOrderList.vue`



```
1 <template>
2   <div>
3     <h2>There are {{ items.length }} items</h2>
4     <ol>
5       <li v-for="item in items" v-bind:key="item">{{ item }}</li>
6     </ol>
7   </div>
8 </template>
9
10 <script>
11 export default {
12   name: "MyOrderList",
13   props: {
14     items: {
15       type: Array,
16       default: () => []
17     }
18   }
19 };
20 </script>
```

這個 component 的很簡單，他的 props 中接受一個叫做 `items` 的陣列，除此之外沒有別的 props 或 data 了。所以我們只要設法把 `items` 以 props 的形式傳下來，應該就能看見理想的結果。

接下來讓我們回到 `MyComponent`，看看範例是如何將 `items` 傳下去的。我們可以看見 `MyComponent` 的 `<template>` 區塊內容如下：

- 檔案：`src/components/ MyComponent.vue`

```
1. <div>
2.   <MyOrderList v-bind:items="fruits"/>
3.   <div class="input-container">
4.     <label>First fruit:</label>
5.     <input type="text" v-bind:value="firstFruit" disabled>
6.   </div>
```

```

7.     <button type="button" v-
      on:click="handleClick">Reverse data source</button>
8. </div>

```

在 `<MyOrderList>` 上出現了一個新的 directive——`v-bind`。`v-bind` directive 做的事情就是將指定的變數以 **props** 的方式傳到特定的 **HTML element 或 component** 中，語法為 `v-bind:Property 名稱="數值"` 例如：

```
<MyOrderList v-bind:items="fruits"/>
```

在以上的範例中，`<MyOrderList>` 就會收到一個名為 `items` 的 props，並且他的值為 `fruits`。

由於一般的 **HTML attribute** (`id`、`class`、`value`、`disabled`...等) 也是 **props** 的一種，所以我們也可以在這些屬性上使用 `v-bind directive`。

另外，VueJS 也提供 `v-bind` 屬性綁定的縮寫，允許我們可以省略冒號前方的 `v-bind` (冒號依然要留著)；因此，底下兩行的程式碼會執行出一樣的結果：原來的樣子：

```
<button type="submit" v-bind:disabled="!hasValue">Submit</button>
```

縮寫：

```
<button type="submit" :disabled="!hasValue">Submit</button>
```

## 8. Input 事件處理

在這個章節中，我們將會學習如何處理 input 事件，並使用這些事件的結果來更新 data。

Sandbox 名稱	Sandbox 連結
Input Handling	<a href="https://codesandbox.io/s/input-handling-3m25n?fontsize=14&amp;hiddenavigation=1&amp;module=%2Fsrc%2F%2Fcomponents%2FMyComponent.vue&amp;theme=dark">https://codesandbox.io/s/input-handling-3m25n?fontsize=14&amp;hiddenavigation=1&amp;module=%2Fsrc%2F%2Fcomponents%2FMyComponent.vue&amp;theme=dark</a>

(備註：每當您看到 Sandbox 連結，就代表接下來即將進入實作環節。在實作的過程中若遇到任何問題，也歡迎隨時提問；請務必確定自己已經完全了解該章節的內容，而且實作上也沒有問題之後再進行至下一個章節。)

經常被使用的 input 事件有三種：

1. onInput (`v-on:input`)
2. onChange (`v-on:change`)
3. onBlur (`v-on:blur`)

通常的作法是在控制項(control)上面指定 `name` 屬性，對應的屬性會事先定義在 data 中，之後將 input 事件綁定到某個 method，最後更新 data，例如：

```
1. <template>
```

```

2.     <div>
3.         <input type="text" name="firstName" v-
         on:input="handleChange">
4.     </div>
5. </template>
6.
7. <script>
8. export default {
9.     name: "MyComponent",
10.    data() {
11.        return {
12.            firstName: ""
13.        };
14.    },
15.    methods: {
16.        handleChange(e) {
17.            const { type, name, value, checked } = e.target;
18.            this.$data[name] = (type === "checkbox" ? checked :
value);
19.        }
20.    }
21. };
22. </script>

```

在這個範例中，`<input>`上有一個 `name` 屬性(值為 `"firstName"`)，我們在 `data` 中也有事先定義好這個屬性。當 `input` 的 `onInput` 事件觸發時，就會將 `event` 傳到綁定的方法中(在這裡是 `handleChange`)，藉由判斷事件 `target` 的 `type`，我們可以分辨出這個 `input` 事件該使用 `value`(控制項不是 `checkbox`)還是 `checked`(控制項為 `checkbox`)的值，搭配 `name` 來決定要更新哪一個屬性，就**不需要每個事件都做一個 handler**。也請注意範例中如何控制送出按鈕是否為 `disabled`。

另外，事件觸發的頻率越高，所需要消耗的資源也就越多；所以**若情況允許，我們應該盡可能的使用 `onBlur` 事件，接下來是 `onChange` 事件，最後才是 `onInput` 事件**。

然而，這種方式的資訊傳遞是**單向**的，意思就是說：**控制項的 `input` 事件會改變 `data`，但是 `data` 的改變卻不會影響到控制項的值**。如果想要達成雙向綁定，我們有兩種做法：

1. 使用 `v-bind:value` 搭配 `input` 事件
2. 使用下方介紹的 `v-model` directive

Sandbox 名稱	Sandbox 連結
v-model directive	<a href="https://codesandbox.io/s/v-model-directive-8i0rk?fontsize=14&amp;hidenavigation=1&amp;module=%2Fsrc%2F%2Fcomponents%2FMyComponent.vue&amp;theme=dark">https://codesandbox.io/s/v-model-directive-8i0rk?fontsize=14&amp;hidenavigation=1&amp;module=%2Fsrc%2F%2Fcomponents%2FMyComponent.vue&amp;theme=dark</a>

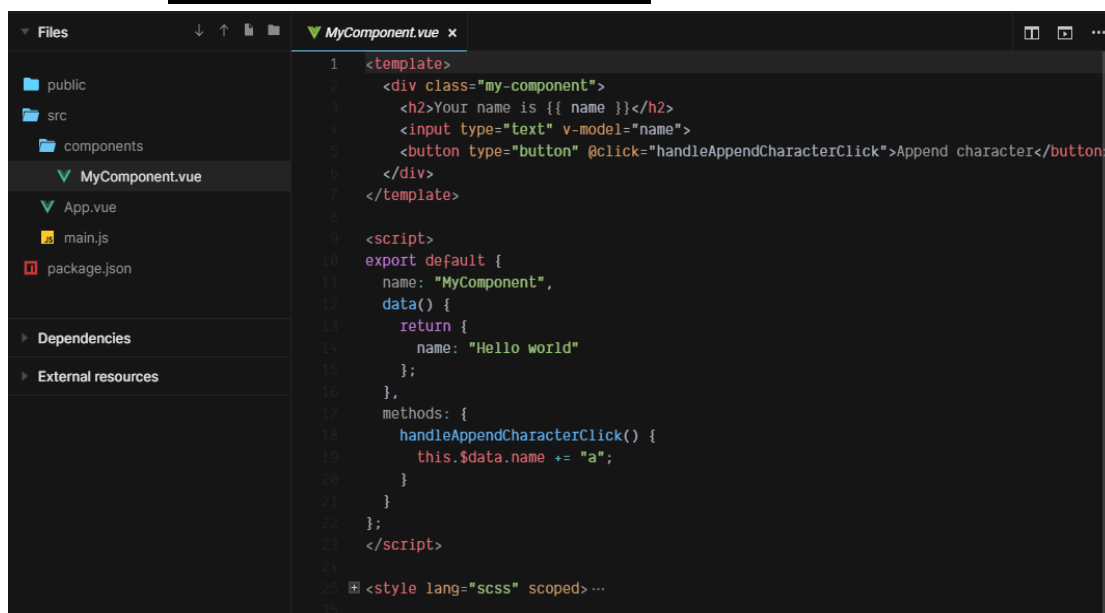
(備註：每當您看到 **Sandbox 連結**，就代表接下來即將進入實作環節。在實作的過程中若遇到任何問題，也歡迎隨時提問；請務必確定自己已經完全了解該章節的內容，而且實作上也沒有問題之後再進行至下一個章節。)

打開 **sandbox** 後您應該會看見以下畫面：



請試著修改 **input** 中的值，並點擊下方的按鈕，觀察頁面的變化。這個 **component** 的程式碼如下：

- 檔案：**src/components/ MyComponent.vue**



我們可以看見，當我們使用 **v-model** directive 綁定變數時，**input** 事件會修改該變數(**name**)的值，而變數 **name** 的修改也會影響到 **input** 內的文字。這種控制項和 **data** 互相影響的綁定方式被稱為雙向綁定(**Two-way data bindings**)。

那麼這兩種的綁定方法我們到底哪該用哪一個呢？我們建議，**如果需要更多客製化的邏輯，就使用 `v-bind` directive**；如果只是單純想要讓 **state** 和 **input** 事件綁在一起，那麼就可以使用 **`v-model` directive**。

更多關於 input binding 的詳細資訊，請參閱官方文件  
<https://vue.js.org/v2/guide/forms.html>。

更多關於 input 事件的資訊，請參閱官方文件  
<https://vue.js.org/v2/guide/events.html>。

## 9. 元件註冊(Component Registration)

在這個章節中，我們將會學習如何將自製的 component 匯入其他的 component 中，讓我們可以藉由 HTML tag 的方式將我們的 component 顯示在網頁上。

Sandbox 名稱	Sandbox 連結
Component Registration	<a href="https://codesandbox.io/s/component-registration-69x15?fontsize=14&amp;hidenavigation=1&amp;module=%2Fsrc%2FApp.vue&amp;theme=dark">https://codesandbox.io/s/component-registration-69x15?fontsize=14&amp;hidenavigation=1&amp;module=%2Fsrc%2FApp.vue&amp;theme=dark</a>

(備註：每當您看到 Sandbox 連結，就代表接下來即將進入實作環節。在實作的過程中若遇到任何問題，也歡迎隨時提問；請務必確定自己已經完全了解該章節的內容，而且實作上也沒有問題之後再進行至下一個章節。)

開啟 Sandbox 後，您應該會看見一片空白。我們要將 `src/components` 底下的 component 匯入 `App`，並將裡面的 component 顯示在頁面上。

### 9.1. Local Registration(本地註冊)

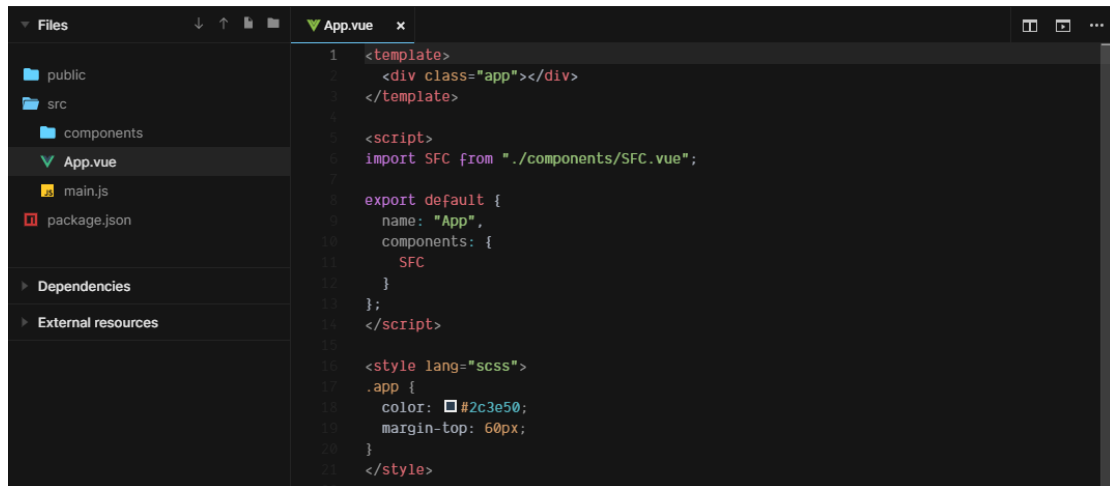
最常見的 component 註冊方式為**本地註冊(Local Registration)**，通常 SFC 都會使用這種方式註冊。執行的步驟為：

1. 打開需要使用該 SFC 的 component(假設他叫做 parent，這裡為 `App`，檔案路徑為 `src/App.vue`)
2. 在 parent 的 `<script>` 區塊將該 component 匯入  

```
import SFC from './components/SFC.vue';
```
3. 在 parent 的 `<script>` 定義中新增 `components` 屬性，並將匯入的 `SFC` 放到裡面去

完成的結果就會像下面這個樣子：

- 檔案：`src/App.vue`



完成之後我們就可以在 `App` 的 `<template>` 區塊裡面使用 `SFC`。所有需要使用 `SFC` 的 `component` 都必須照著這種方式將 `SFC` 註冊到自己的 `components` 屬性裡面。本地註冊的好處是我們知道每個 `component` 的來源是哪裡；壞處是需要不斷的註冊。所以使用頻率特別高的 `component`(例如：`loading-spinner`)，建議使用底下介紹的全域註冊。

## 9.2. Global Registration(全域註冊)

另外一種註冊方式稱為 **Global Registration(全域註冊)**，以這種方式註冊的 `component` 在整個程式裡面都可以用，不需要在不同的檔案裡面分別 `import`。但是，如果您正在使用 `webpack` 等打包工具，那麼就算這個 `component` 沒有在任何一個地方使用，全域註冊的 `component` 最後還是會出現在 `.js` 檔案中；代表最後產出的 `.js` 檔中會包含完全不需要的程式碼，因此建議只對極常使用的 `component` 做全域註冊。

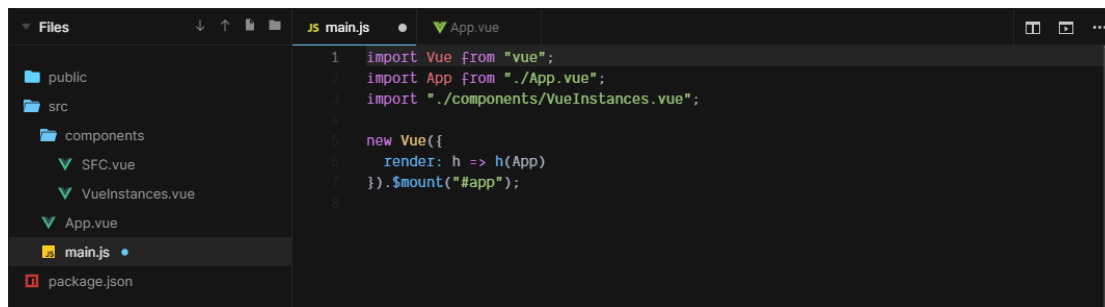
通常我們會在 `src/main.js` 中(就是任何畫面 `render` 之前)做全域註冊。執行的步驟為：

1. 以 `Vue.component` 的方式宣告新的 `component`(將會在底下說明)，這種方式宣告出來的 `component` 稱為 **Vue instance**
2. 視情況將 `Vue instances` 依類別放在不同檔案裡面，就像範例 `Sandbox` 中的 `src/components/VueInstances.vue`
3. 打開 `src/main.js`，在上方 `import` 所有的 `Vue instance` 檔案，範例中只有一個

```
import './components/VueInstances.vue';
```

完成的結果就會像下面這個樣子：

- 檔案：`src/main.js`



完成之後我們就可以在任何一個頁面上使用已經完成全域註冊的 **component**。要特別注意的是，因為 **Vue instance** 並不是使用 `export default` 的方式匯出，所以匯入時只要指定檔案路徑即可，不需要給名字；既然沒有名字，那麼也就沒有辦法像註冊 **SFC** 那樣把他加到 `<script>` 定義的 **components** 屬性中。

`Vue.component` 是一個函數(function)，第一個參數為 **component** 的名稱(就是 **HTML tag**)；而第二個參數就是 **SFC** 中的 **script** 區塊，只是少了 `export default` 而已。不同的地方是 **Vue instance** 的 **template** 是以字串的方式定義在第二個參數中，例如：

```
1. Vue.component("HelloWorldComponent", {
2.   template: `<div>Hello, world</div>`
3. });
```

**SFC** 的規則及限制在這裡也都能套用，但是不包含 `<style>` 區塊——**Vue instance** 是沒有 `<style>` 區塊的。如果想要替 **Vue instance** 加上 **style**，常見的做法有兩種：

1. 將這些 **style** 放在 **css** 檔案中(`.css`、`.scss`...等)並匯入(通常會在 `src/main.js`)

```
import "./styles/some-components.scss";
```

2. 使用 `Vue.component` 註冊 **SFC**，詳細步驟為：

- A. 做出一個 **SFC**(範例為 `SFC`)

- B. 在想要進行全域註冊的地方匯入 **SFC**(範例為 `src/main.js`)

```
import SFC from "./components/SFC.vue";
```

- C. 把做好的 **SFC** 放到 `Vue.component` 的第二個參數中，就像這樣：

```
Vue.component("SFC", SFC);
```

如此一來這個 **SFC** 就完成了全域註冊，可以在任何地方使用。

另外，**Vue instance** 的名稱會以第一個參數為準，所以在第二個參數裡面加上 `name` 屬性並不會影響 **Vue instance** 的運作。

## 10. Lifting State Up

在這個章節中，我們將會學習如何解決兩種問題：

1. 相同層級的 **component** 需要存取彼此的 **state**
2. 父元件需要存取子元件的 **state**



Sandbox 名稱	Sandbox 連結
Lifting State Up	<a href="https://codesandbox.io/s/lifting-state-up-diyqx?fontsize=14&amp;hidenavigation=1&amp;module=%2Fsrc%2Fcomponents%2FMyComponent.vue&amp;theme=dark">https://codesandbox.io/s/lifting-state-up-diyqx?fontsize=14&amp;hidenavigation=1&amp;module=%2Fsrc%2Fcomponents%2FMyComponent.vue&amp;theme=dark</a>

(備註：每當您看到 Sandbox 連結，就代表接下來即將進入實作環節。在實作的過程中若遇到任何問題，也歡迎隨時提問；請務必確定自己已經完全了解該章節的內容，而且實作上也沒有問題之後再進行至下一個章節。)

打開 Sandbox 之後，您應該會看見以下的畫面：

Name:   
Nobody is a good kid.

`<MyComponent>` 的階層看起來就像下面這個樣子：

```
<form>
  <NameInput />
  <GoodKidComponent />
  <button type="submit">Submit</button>
</form>
```

其中，`<NameInput>` 裡面有一個 input。`<GoodKidComponent>` 的作用就是將 input 中的值(假設叫做 `name`)顯示出來。最後，我們可以點擊 `<MyComponent>` 底下的按鈕來印出 `name`。

Name:   
Banana is a good kid.

在 input 中輸入 "Banana" 之後，`<GoodKidComponent>` 裡面就顯示了 "Banana is a good kid. "。

```
► Object {name: "Banana"}
```

點擊傳送按鈕之後也可以在 console 中看見這個屬性。

請試著用您目前擁有的知識想想這 3 個 component—`<MyComponent>`、`<NameInput>`和`<GoodKidComponent>`該如何實作，並試著做出上述的功能；如此一來您會更了解問題的所在。

我們到底該如何解決這些問題呢？首先我們知道，既然`<GoodKidComponent>`可以動態顯示 `name`，代表在`<GoodKidComponent>`的`<template>`中一定使用了雙括弧的語法來綁定 `name`，才能做到動態顯示的效果。但是仔細想想，如果 `name` 存在於`<GoodKidComponent>`的 `data` 中，那麼和他同個層級的 component—`<NameInput>`又是怎麼更新 `name` 的？位於 parent 層級的`<MyComponent>`又是怎麼拿到這個 `name` 將他印出來的？

要解決這種問題，我們必須先觀察在用到 `name` 屬性的 component 中，最上層的 component(root)是哪一個。以上面的例子來說，`<MyComponent>`是 root，所以 `name` 應該儲存在`<MyComponent>`的 `data` 中。

但是負責顯示 `name` 的是`<GoodKidComponent>`，所以我們可以將 `name` 以 props 的形式傳給`<GoodKidComponent>`。最後就剩下更新 `name` 這件事情了。在這裡我們需要把 input handler 以 props 的形式傳給`<NameInput>`，這在 component 的設計模式裡非常常見。如此一來，我們就能解決在本章開頭提到的兩種問題。

請特別注意範例中`<NameInput>`的 `onInput` 事件是定義在 props 裡面的，我們使用 `v-bind` 而不是 `v-on`。

另外，雖然在範例中 parent 的 state 和 children 的 props 名稱都叫做 `name`，但是這些東西的名字(key)不見得要相同，不同名稱也不會影響 component 的運作。

## 11.Lifecycle Hooks

VueJS 的 component 的生命週期分為數個階段，我們可以在這些階段中加入一些程式邏輯，用以解決我們的問題。例如：某個 component 一出現就必須做某些事情(例如：呼叫 api 來取資料)，這種時候我們就需要用到 lifecycle hooks。在這個章節中，我們將會學習如何使用最常使用的 lifecycle hook—`mounted`。

Sandbox 名稱	Sandbox 連結
Lifecycle hook-mounted	<a href="https://codesandbox.io/s/lifecycle-hook-mounted-5ippb?fontsize=14&amp;hidnavigation=1&amp;module=%2Fsrc%2Fcomponents%2FMyComponent.vue&amp;theme=dark">https://codesandbox.io/s/lifecycle-hook-mounted-5ippb?fontsize=14&amp;hidnavigation=1&amp;module=%2Fsrc%2Fcomponents%2FMyComponent.vue&amp;theme=dark</a>

(備註：每當您看到 Sandbox 連結，就代表接下來即將進入實作環節。在實作的過程中若遇到任何問題，也歡迎隨時提問；請務必確定自己已經完全了解該章節的內容，而且實作上也沒有問題之後再進行至下一個章節。)

打開 sandbox 後，您應該會看見以下畫面：

### There are 3 fruits

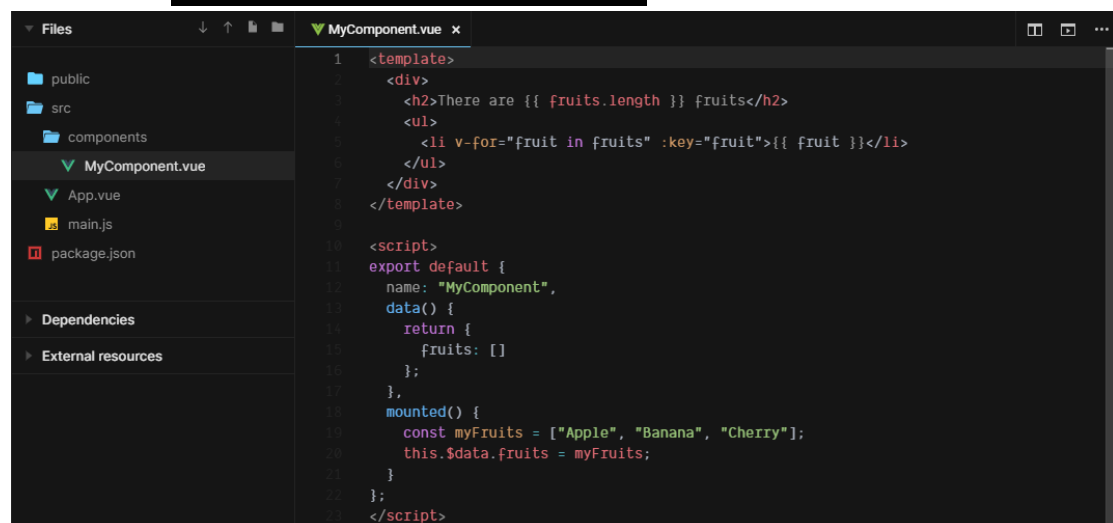
- Apple
- Banana
- Cherry

在先前學到的章節中，`component` 中所有的資料都是事先輸入好的。但是在現實生活中，頁面上的資料大多是藉由呼叫 `api` 後得到，不太可能事先輸入好。因此我們就有了一項新的需求：「**使用者在進入頁面的同時，我們就要呼叫 `api` 來拿資料**」。代表只要這個 `component` 一出現在頁面上，他就應該自動呼叫，不需要使用者進行任何額外的操作。因此我們就有了一項新的需求：「**使用者在進入頁面的同時，我們就要呼叫 `api` 來拿資料**」。代表只要這個 `component` 一出現在頁面上，他就應該自動呼叫 `api`，不需要使用者進行任何額外的操作。

因此就算我們把呼叫 `api` 的方法寫在 `methods` 裡面，只要使用者沒有用某種方式來觸發這個方法(例如：按下某個按鈕)，我們的 `api` 就永遠不會被呼叫。所以我們才需要 `lifecycle hooks` 來幫助我們達成這個目的。

VueJS 的 `component` 中有許多的 `lifecycle hooks`，最適合上述需求的就是 **`mounted`**。使用的方法也很簡單，只要在 `component` 裡面加上對應名稱的屬性就可以了。**所有的 `lifecycle hooks` 都是 `function`，而屬性名稱就和 `lifecycle hook` 的名稱一樣**。當 `component` 進入該階段時，他就會執行 `function` 裡面的程式碼，完成我們想要達到的目標。以 `mounted` 為例，我們可以在 `component` 的 `<script>` 區塊加上 `mounted` `function`，就像下面這個樣子：

- 檔案：`src/components/MyComponent.vue`



```
1 <template>
2   <div>
3     <h2>There are {{ fruits.length }} fruits</h2>
4     <ul>
5       <li v-for="fruit in fruits" :key="fruit">{{ fruit }}</li>
6     </ul>
7   </div>
8 </template>
9
10 <script>
11 export default {
12   name: "MyComponent",
13   data() {
14     return {
15       fruits: []
16     };
17   },
18   mounted() {
19     const myFruits = ["Apple", "Banana", "Cherry"];
20     this.$data.fruits = myFruits;
21   }
22 };
23 </script>
```

我們可以看見，在 `data` 中的 `fruits` 是一個空陣列，他的值會在 `mounted` 階段被放入，之後顯示在頁面上。在這些 **lifecycle hooks** 中，我們也可以使用 **props** 和 **data** 來配合 **api** 的呼叫。至於該如何呼叫 `api`，我們會在**13** 呼叫 **API—Axios** 做更詳細的說明。

關於更多 `lifecycle hooks` 的介紹，請參閱官方文件 <https://vue.js.org/v2/guide/instance.html#Instance-Lifecycle-Hooks>。

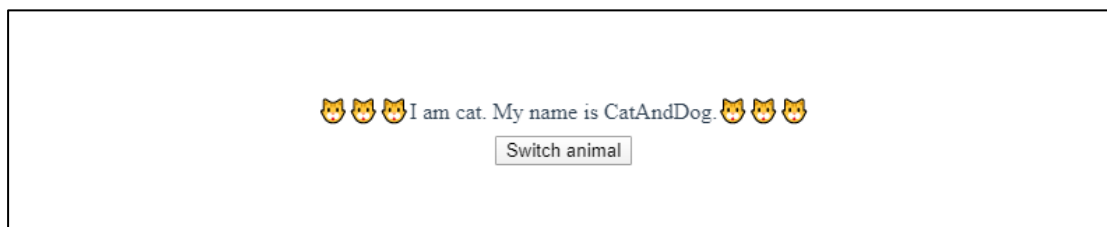
## 12. Dynamic component(v-is Directive)

在這個章節中，我們將會學習如何使用 `<component>` tag 和 `v-is directive` 來動態顯示 `component`。

Sandbox 名稱	Sandbox 連結
Dynamic component	<a href="https://codesandbox.io/s/dynamic-component-cc2pk?fontsize=14&amp;hidenavigation=1&amp;module=%2Fsrc%2F%2Fcomponents%2FMyComponent.vue&amp;theme=dark">https://codesandbox.io/s/dynamic-component-cc2pk?fontsize=14&amp;hidenavigation=1&amp;module=%2Fsrc%2F%2Fcomponents%2FMyComponent.vue&amp;theme=dark</a>

(備註：每當您看到 **Sandbox 連結**，就代表接下來即將進入實作環節。在實作的過程中若遇到任何問題，也歡迎隨時提問；請務必確定自己已經完全了解該章節的內容，而且實作上也沒有問題之後再進行至下一個章節。)

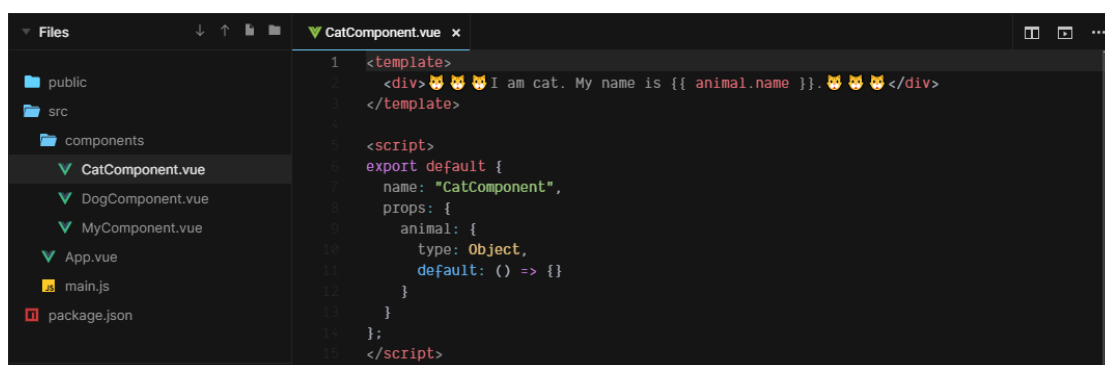
開啟 **Sandbox** 後，您應該會看見以下畫面：



請試著點擊畫面上的按鈕，並觀察頁面的變化。

在進入 `MyComponent` 之前，讓我們先來看看 `CatComponent`。

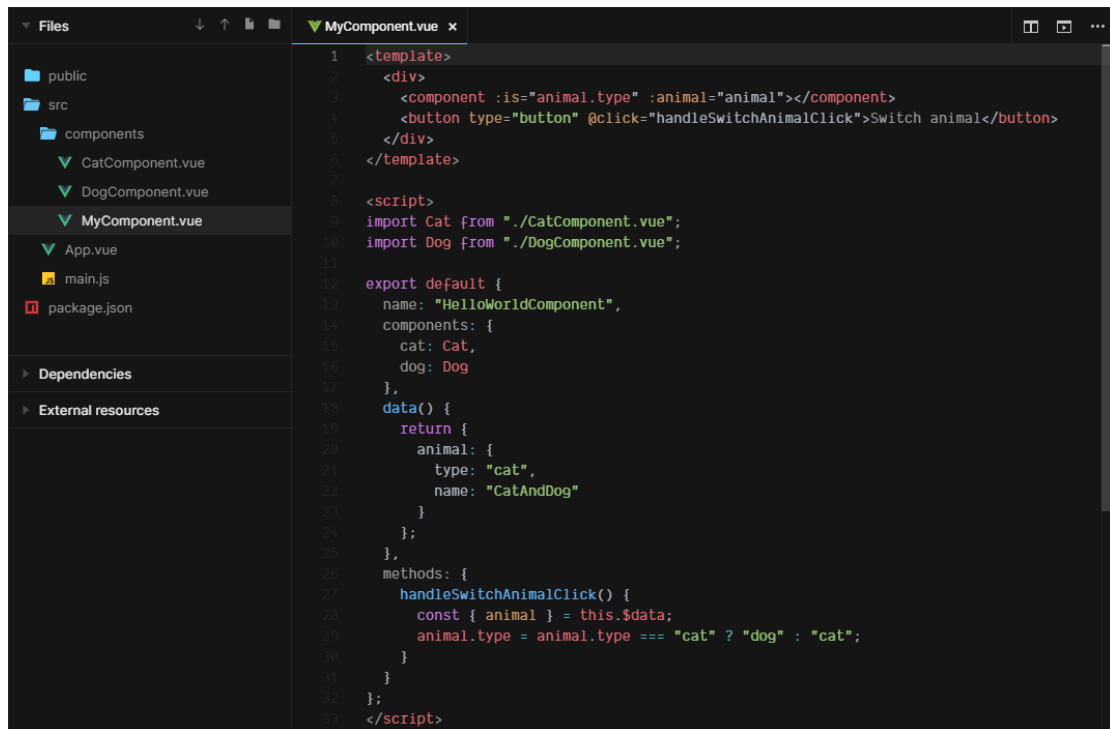
- 檔案：`src/components/ CatComponent.vue`



這個 component 的很簡單，他的 props 中接受一個叫做 `animal` 的物件，然後把他的 `name` 屬性顯示在頁面上。`DogComponent` 的邏輯一樣，只是把 `cat` 換成 `dog` 而已。

現在讓我們回到 `MyComponent`，看看他如何做到切換 component 的功能。

- 檔案：`src/components/ MyComponent.vue`



```
1 <template>
2   <div>
3     <component :is="animal.type" :animal="animal"></component>
4     <button type="button" @click="handleSwitchAnimalClick">Switch animal</button>
5   </div>
6 </template>
7
8 <script>
9   import Cat from "../CatComponent.vue";
10  import Dog from "../DogComponent.vue";
11
12  export default {
13    name: "HelloWorldComponent",
14    components: {
15      cat: Cat,
16      dog: Dog
17    },
18    data() {
19      return {
20        animal: {
21          type: "cat",
22          name: "CatAndDog"
23        }
24      };
25    },
26    methods: {
27      handleSwitchAnimalClick() {
28        const { animal } = this.$data;
29        animal.type = animal.type === "cat" ? "dog" : "cat";
30      }
31    }
32  };
33 </script>
```

在`<template>`區塊中我們看見了一個新的 tag—`<component>`，他的功用是根據我們提供的名稱來動態載入 component。我們可以在`<component>` tag 上提供一個 `is` 屬性來告知 VueJS 應該要讀取哪一個 component，而且這個 `is` 屬性是必須使用 `v-bind directive` 綁定上去。也請注意 `MyComponent` 在註冊 `CatComponent` 及 `DogComponent` 時給了自訂的名字(key)，讓他符合 `animal` 中的 `type`，好讓 `<component>` tag 可以對應到這兩個 component。

更多關於 dynamic component 的介紹，請參閱官方文件 <https://vuejs.org/v2/guide/components-dynamic-async.html>。

## 13. 呼叫 API—Axios

TBD。

## 14. Assignment

給定一個資料陣列，您的 **Vue App** 必須將這些資料顯示出來，並允許使用者進行編輯。

您可以選擇使用以下的 **sandbox** 作為起始專案(已經加上了 **bootstrap**)，或是自己創建一個。

Sandbox 名稱	Sandbox 連結
Blank project with bootstrap	<a href="https://codesandbox.io/s/blank-project-with-bootstrap-qk9y2">https://codesandbox.io/s/blank-project-with-bootstrap-qk9y2</a>

(備註：每當您看到 **Sandbox** 連結，就代表接下來即將進入實作環節。在實作的過程中若遇到任何問題，也歡迎隨時提問；請務必確定自己已經完全了解該章節的內容，而且實作上也沒有問題之後再進行至下一個章節。)

在實作的過程中也請注意以下幾點：

1. 使用 **VueJS** 實作頁面
2. 不一定要有 **CSS**，純 **HTML** 亦可；若要使用 **CSS** 框架，推薦但不限於 **Bootstrap**
3. 不需要 **RWD**，能在電腦螢幕上正常檢視即可
4. 您不該使用 **jQuery** 等 **library**
5. 介面的顯示語言不限，中文、英文、任何語言皆可

## 14.1. 基本練習

**Form A0000001**

跳至 A0000001 ▾ , 共 5 張

[Log this form](#) [Log all forms](#) [Delete this form](#)

Personal Info [Orders](#) [Overview](#)

First name  Last name

Gender ☐ Secret ☒ Male ☐ Female

Address

☐ 此客戶居無定所

Job

Note  
0 / 2000 characters

[Previous](#) [1](#) [2](#) [3](#) [4](#) [5](#) [Next](#)

您的 Web app 必須包含下列幾項功能：

1. 頁面最上方會顯示當前表單的編號

**Form A0000001**

2. 上方有一個選單，可快速跳到其他表單；旁邊會顯示共有幾張表單

跳至 A0000001 ▾ , 共 5 張

[Log this form](#) [Log all forms](#) [Delete this form](#)

Personal Info [Orders](#) [Overview](#)

First name  Last name

Gender ☐ Secret ☒ Male ☐ Female

Address

☐ 此客戶居無定所

Job

Note  
0 / 2000 characters

[Previous](#) [1](#) [2](#) [3](#) [4](#) [5](#) [Next](#)

3. 上方有三個按鈕，功能分別為

[Log this form](#) [Log all forms](#) [Delete this form](#)

- A. 將當前的表單以 `console.log` 印出來(Log this form)
- B. 將所有表單使用 `console.log` 印出來(Log all forms)

- C. 將當前表單刪除(Delete this form)，刪除後自動切換至下一張表單。如果表單剩下最後一張，那麼就不會顯示這個按鈕
4. 表單共有 3 個標籤頁：
- 請特別注意，如果該標籤頁裡面的內容允許使用者修改，那麼他應該要能保存修改後的結果。(意即當我切換到其他表單再切換回來，我所看到的結果應該要是剛才修改後的結果)

A. Personal Info

Personal Info   Orders   Overview

First name   Last name

Fishman   ILike

Gender

☐ Secret

☒ Male

☐ Female

Address

ABCDEFG

☐ 此客戶居無定所

Job

保密

Note

0 / 2000 characters

Previous   1   2   3   4   5   Next

請特別注意以下規範/邏輯：

- i. Gender 共有三個 radio buttons
1. Secret(在 json 中是數字的 0)
  2. Male(在 json 中是數字的 1)
  3. Female(在 json 中是數字的 2)
- Gender

☐ Secret

☒ Male

☐ Female
- ii. 當「此客戶居無定所」的 checkbox 被勾選時，Address 輸入框中的內容會被清除，同時該輸入框會變成不可以編輯的狀態；取消勾選時，輸入框的內容不會恢復(依然被清除)，但是會恢復成可編輯的狀態



Address

Address

☒ 此客戶居無定所

- iii. Job 選單共有四個選項
1. 保密(在 json 中是空字串或是 null)
  2. 調查員(在 json 中是 "agent")
  3. 秘密調查員(在 json 中是 "secret\_agent")
  4. 秘密調查員的調查員(在 json 中是 "agent\_of\_secret\_agent")

Job

保密

保密

調查員

秘密調查員

秘密調查員的調查員

- iv. Note 輸入框(textarea)最多輸入 2000 個字元，輸入框上方會顯示目前已經輸入了幾個字元

Note

20 / 2000 characters

He lives in the sea.

## B. Orders

Personal Info Orders Overview

Apple -5 -1 1 +1 +5

Banana

配料

☒ 巧克力醬 ☐ 草莓醬 ☐ 胡麻醬 ☐ 味噌 ☒ 辣椒 ☒ 大蒜 ☒ 醬油 ☐ 醬油膏 ☐ 百草膏

Previous 1 2 3 4 5 Next

請特別注意以下規範/邏輯：

- i. Apple 欄位為數字輸入框，最少為 1 個，最多為 100 個；左右兩邊分別有-5, -1 和+1, +5 的按鈕，點擊之後會改變輸入框裡面的值，輸入框依然可以手動輸入數字，且需為整數。
- ii. Banana 配料欄位允許使用者進行多選(也可以一個都不選)，可選擇的選項有：
  1. 巧克力醬(在 json 中是 "chocolate")

2. 草莓醬(在 json 中是"strawberry")
3. 胡麻醬(在 json 中是"flax")
4. 味噌(在 json 中是"miso")
5. 辣椒(在 json 中是"chili")
6. 大蒜(在 json 中是"garlic")
7. 醬油(在 json 中是"soy\_sauce")
8. 醬油膏(在 json 中是"thick\_soy\_sauce")
9. 百草膏(在 json 中是"herbal\_cream")

### C. Overview

Personal Info	Orders	Overview
<p>以下是 ILike Fishman 的訂單：</p> <ul style="list-style-type: none"> <li>• 1 顆蘋果</li> <li>• 香蕉配料               <ul style="list-style-type: none"> <li>◦ 巧克力醬</li> <li>◦ 辣椒</li> <li>◦ 大蒜</li> <li>◦ 醬油</li> </ul> </li> </ul>		
<div> <span>Previous</span> <span>1</span> <span>2</span> <span>3</span> <span>4</span> <span>5</span> <span>Next</span> </div>		

請特別注意以下規範/邏輯：

- i. 上方會顯示"以下是 {{ 某個人 }} 的訂單"。{{ 某個人 }}指的是 Personal Info 標籤頁裡面的人名(last\_name 和 first\_name 欄位)。如果兩個欄位都有值，那麼就將 last\_name 放在前方；如果只有其中一個欄位有值，那麼就顯示該欄位的值；如果兩個欄位都沒有值，那麼就顯示"unknown"
  - ii. 人名下方會顯示 Orders 標籤頁中的資料；除了顯示這個人購買了幾顆蘋果之外，也會顯示他的香蕉放了什麼配料；如果一個配料都沒有，那麼就只顯示蘋果數量，不顯示香蕉配料
- D. 特別注意每個標籤頁底下都會有可切換的分頁，會隨著表單數量改變；當滑鼠移到指定的分頁上時，會顯示該表單的編號(上一頁/下一頁也有這樣的功能；如果沒有上一頁/下一頁，那麼當滑鼠移動到上一頁/下一頁時就不會顯示表單編號)

Previous	1	2	3	4	5	Next
A0000001						

以下為範例 json：

```
[{
  "id": "A0000001",
```

```

    "personal_info": {
        "first_name": "Fishman",
        "last_name": "ILike",
        "gender": 1,
        "address": "ABCDEFGF",
        "is_homeless": false,
        "job": null,
        "note": null
    },
    "orders": {
        "apple_count": 1,
        "banana_condiments": ["chocolate", "chili", "garlic", "soy_sauce"]
    }
}
]]

```

以下為測試資料：

```

[{"id": "A0000001", "personal_info": {"first_name": "Fishman", "last_name": "ILike", "gender": 1, "address": "ABCDEFGF", "is_homeless": false, "job": null, "note": null}, "orders": {"apple_count": 1, "banana_condiments": ["chocolate", "chili", "garlic", "soy_sauce"]}}, {"id": "A0000002", "personal_info": {"first_name": null, "last_name": "Somebody", "gender": 2, "address": "QWERTY", "is_homeless": false, "job": "secret_agent", "note": "Hello, world"}, "orders": {"apple_count": 15, "banana_condiments": []}}, {"id": "A0000003", "personal_info": {"first_name": "Painter", "last_name": null, "gender": 0, "address": null, "is_homeless": true, "job": "agent_of_secret_agent", "note": "Strange"}, "orders": {"apple_count": 1, "banana_condiments": ["herbal_cream"]}}, {"id": "A0000004", "personal_info": {"first_name": "President", "last_name": "Mr.", "gender": 1, "address": "America", "is_homeless": false, "job": "secret_agent", "note": "He is the president!"}, "orders": {"apple_count": 100, "banana_condiments": ["chocolate", "strawberry", "flax", "miso", "chili", "garlic", "soy_sauce", "thick_soy_sauce"]}}, {"id": "A0000005", "personal_info": {"first_name": "Pepper", "last_name": "Dr.", "gender": 0, "address": "Farm", "is_homeless": false, "job": "agent", "note": "胡椒博士現身"}, "orders": {"apple_count": 5, "banana_condiments": ["chili", "garlic"]}}]]

```

## 14.2. 進階練習

在**14.1 基本練習**中，每張表單的標籤頁順序都是固定的(Personal Info -> Orders -> Overview)，但是在實務上表單標籤頁的順序及數量有可能不一樣。給定新型的資料格式如下：

```
[{
  "id": "A0000001",
  "records": [
    {
      "tab_name": "personal_info",
      "tab_sequence": 0,
      "data": {
        "first_name": "Fishman",
        "last_name": "ILike",
        "gender": 1,
        "address": "ABCDEFGF",
        "is_homeless": false,
        "job": null,
        "note": null
      }
    },
    {
      "tab_name": "orders",
      "tab_sequence": 1,
      "data": {
        "apple_count": 1,
        "banana_condiments": ["chocolate", "chili", "garlic", "soy_sauce"]
      }
    },
    {
      "tab_name": "overview",
      "tab_sequence": 2
    }
  ]
}]
```

上述的 json 定義代表這張表單的 ID 為 **"A0000001"**，**records** 屬性(陣列)裡面放的則是這張表單裡面的標籤頁及其內容，一個物件就代表一個標籤頁。

此表單目前共有 3 個標籤頁—**personal\_info**、**orders** 和 **overview**。這些標籤頁的欄位、邏輯都和**14.1 基本練習**中所描述的一模一樣，但是標籤頁的順序

是由 `tab_sequence` 屬性來決定的，數值越小的排在越前方(即畫面左方)。請使用您在**14.1 基本練習**中所完成的 web app 作為基底，實作出符合新型資料格式的 web app。

實作時請注意以下幾點：

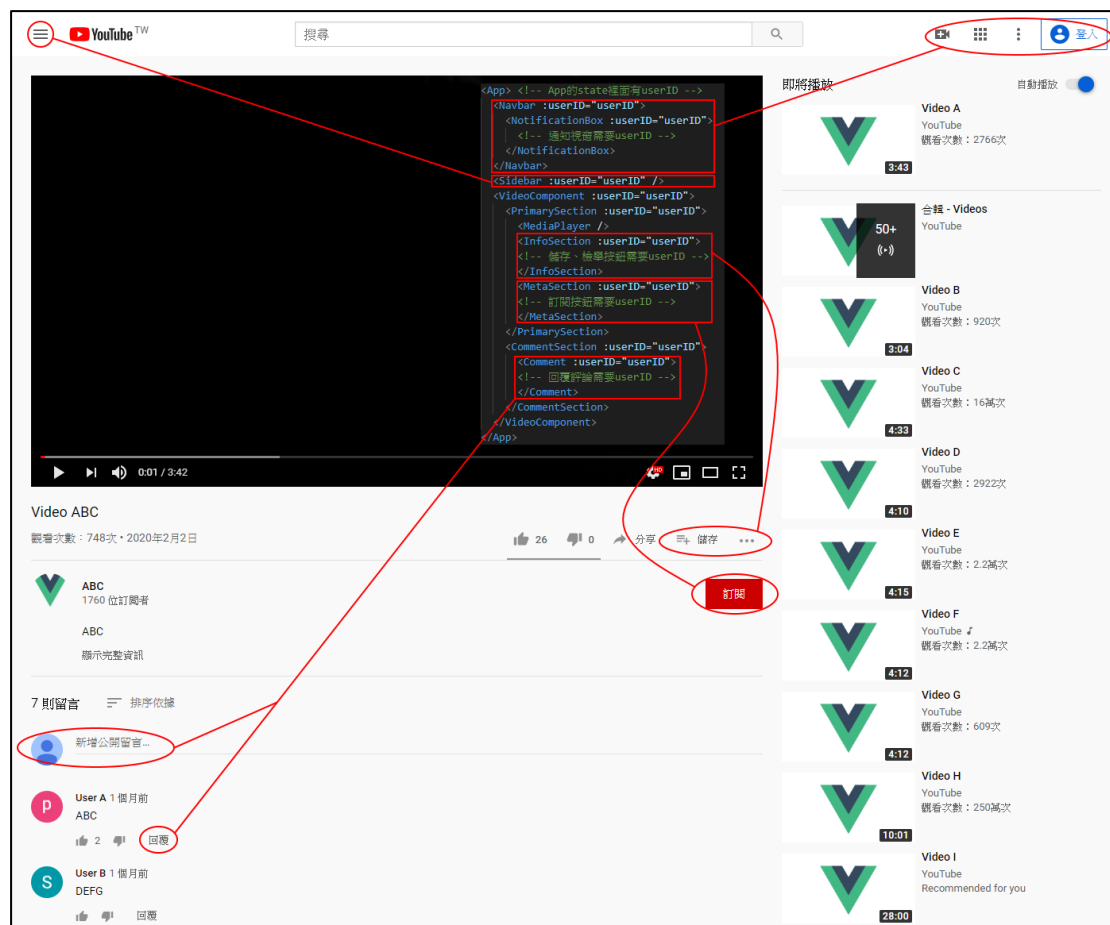
1. 每張表單的標籤頁順序可能不同
2. 不是所有的表單都有 3 個標籤頁(但是至少會有一個)
3. 切換表單時請開啟該表單的第一個標籤頁
4. 假定所有表單 records 中的標籤頁已經依照 `tab_sequence` 屬性由小排到大，且同個表單標籤頁的 `tab_sequence` 都不重覆，所以不需要對測試資料做任何處理
5. 在 Overview 標籤頁中，如果該表單沒有 PersonalInfo 可供顯示，那麼就在原先顯示人名的位置顯示 `"unknown"`；若該表單沒有 Orders 可供顯示，那麼就在 Overview 中顯示 `"No data found"`
6. 請使用 `v-is` directive 來動態顯示表單

以下為測試資料：

```
[{"id": "A0000001", "records": [{"tab_name": "personal_info", "tab_sequence": 0, "data": {"first_name": "Fishman", "last_name": "ILike", "gender": 1, "address": "ABCDEFGH", "is_homeless": false, "job": null, "note": null}}, {"tab_name": "orders", "tab_sequence": 1, "data": {"apple_count": 1, "banana_condiments": ["chocolate", "chili", "garlic", "soy_sauce"]}}, {"tab_name": "overview", "tab_sequence": 2}], {"id": "A0000002", "records": [{"tab_name": "orders", "tab_sequence": 0, "data": {"apple_count": 15, "banana_condiments": []}}, {"tab_name": "personal_info", "tab_sequence": 1, "data": {"first_name": null, "last_name": "Somebody", "gender": 2, "address": "QWERTY", "is_homeless": false, "job": "secret_agent", "note": "Hello, world"}}, {"tab_name": "overview", "tab_sequence": 2}], {"id": "A0000003", "records": [{"tab_name": "personal_info", "tab_sequence": 0, "data": {"first_name": "Painter", "last_name": null, "gender": 0, "address": null, "is_homeless": true, "job": "agent_of_secret_agent", "note": "Strange"}}, {"tab_name": "orders", "tab_sequence": 1, "data": {"apple_count": 1, "banana_condiments": ["herbal_cream"]}}], {"id": "A0000004", "records": [{"tab_name": "orders", "tab_sequence": 0, "data": {"apple_count": 100, "banana_condiments": ["chocolate", "strawberry", "flax", "miso", "chili", "garlic", "soy_sauce", "thick_soy_sauce"]}}], {"id": "A0000005", "records": [{"tab_name": "overview", "tab_sequence": 0}]}]
```

## 15. Vuex

在某些情況下，有些 `state` 必須在很多個 `component` 之間共用(例如：目前使用者的 `id`)，而最直觀的做法(不靠 `window["variable_name"]`、`localStorage`...等作弊的方法)就是把這些 `state` 放在最上層的 `component` 中，然後再用 `props` 的方式一層一層傳下去，以 YouTube 來說，`component` 的階層可能就會像下面這個樣子：



我們將 `userID` 放在最上層的 `component`—`App` 裡面，之後將他一路向下傳。我們可以看出，有些 `component` 自己並不需要 `userID` 這個 `props`，但是他的 `children` 需要，所以他也必須擁有這個 `props`。雖然這種方式的確可以達到我們的目的，但是在維護時會非常麻煩，也許這個 `props` 一傳就是 10 層，非常不理想；`Vuex` 就是為了解決這種問題而出現的 `library`。

`Vuex` 提供一個 `global state`(全域狀態)，讓每個 `component` 都可以跳過層層的 `props` 直接存取這些屬性(其實不是直接)。實作邏輯目前不需要知道，只需要會用就可以了。`Vuex` 預設將 `global state` 儲存在 `this.$store.state` 裡面，如果我們想要在某個 `component` 裡面獲取全域的 `userID`，看起來大概會像這樣：

```
const { userID } = this.$store.state;
```

`global state` 的存取方式看起來就和原來的 `data` 和 `props` 差不多；然而，想要更新 `currentUser` 就沒有這麼容易了。`Vuex` 採用 `dispatch` 機制，代表我們沒

有辦法使用 `this.$store.state.userID = value;` 這種方式來修改 global state 。  
更詳細的介紹，請參閱 Vuex 官方網站 <http://vuex.vuejs.org/> 。

## 16. 額外資訊

### 16.1. VueJS 程式編寫風格指南(<https://vuejs.org/v2/style-guide/>)

包含 component 命名方式、專案目錄結構、常常漏寫的屬性…等，非常建議把他讀完。

### 16.2. Class and Style Bindings(<https://vuejs.org/v2/guide/class-and-style.html>)

實用的技巧，可以做到像是 classnames 套件那個樣子的功能。