



ch0_BeforeVue_Basic JS

概覽

命名法

內建物件(class)	PascalCase	Number FileReader RegExp
屬性和方法	camelCase	length addEventListener()
常數Const	SNAKE_CASE	Math.PI Number.MAX_VALUE
關鍵字 (未來保留字)	all lowercase	switch for
事件	all lowercase	click/onclick readystatechange
屬性的命名	kebab-case	css 屬性 attribute color, font-size HTML標籤的自訂屬性 properties data-*attribute <p sn="A123">XX</p> 不要這樣寫 <p data -sn="A123">XX</p> 要這樣寫

基本語法

variable	types	input/output	operators
iterations	selections	function	...

物件導向

物件	(1)變數,HTML,物件的屬性(2)屬性方法
事件	X
Window物件	屬性: (1)navigator(2)location(3)history(4)document、方法
內建物件	(1)Number(2)String(3)Boolean(4)Array(5)Math(6)Date(7)RegExp
自訂物件	(1)object(2)class

呼叫函數--事件呼叫

呼叫方式

直接呼叫	<code>action()</code>
事件聆聽 物件	<code>.onclick=action</code>
計時器	<code>setInterval(action,1000)</code>

1.建立事件聆聽功能(包含一個物件+一個事件+一個處理函數)

.html	<code><button onclick="action">XX</button></code>
.js	<code>button物件.onclick=action</code> //不要加小括號 等號最後處理沒有按就執行
.js	<code>button物件.addEventListener('click',action,false)</code> <code>element.addEventListener(event, handler, useCapture);</code>

1. `useCapture`：布林值，指定事件是否在捕獲階段觸發。這是第三個參數。
2. `addEventListener` 中，`useCapture` 的作用是決定事件是否應該在捕獲階段還是冒泡階段觸發。
 - **false**：事件將在事件冒泡階段觸發。事件冒泡是指當一個元素觸發事件時，事件會從最內層的元素（事件目標）開始，然後逐層向外傳遞（即父元素，祖先元素），直到 `document`。
使用 `false`（或不提供該參數，因為默認值是 `false`），表示事件處理器會在冒泡階段被觸發。
 - **true**：事件將在事件捕獲階段觸發。事件捕獲是指事件會從最外層的元素（例如 `document`）開始，逐層向內傳遞，直到事件目標。

2. 事件分類 鼠標事件 鍵盤事件 瀏覽器事件 表單事件

click	當鼠標按鈕在元素上按下並釋放時觸發。
mousedown	當鼠標按鈕被按下時觸發。
mouseup	當鼠標按鈕被釋放時觸發。
dblclick	當鼠標雙擊元素時觸發。 注意：不要在同一個物件同時使用 click 和 dblclick。
mousemove	當鼠標在元素上移動時觸發。
contextmenu	當鼠標右鍵點擊時觸發。
mouseover	與 mouseenter 類似，但 mouseover 會冒泡，即當滑鼠進入該元素及其子元素時也會觸發。
mouseout	當鼠標離開元素時觸發。
mouseenter	與 mouseover 類似，但 mouseenter 不會冒泡，即只會在滑鼠進入該元素時觸發，不會觸發子元素。
mouseleave	當鼠標離開元素時觸發。
keypress	當鍵盤上的某個字符鍵被按下並輸入字符時觸發。這個事件與 keydown 類似，但僅對那些會輸入字符的按鍵有效，例如字母和數字鍵。
keydown	當鍵盤上的任何鍵被按下時觸發。
keyup	當鍵盤上的任何鍵被釋放時觸發。
load	當頁面和所有資源（圖像、樣式等）完全加載後觸發。
unload	當頁面被卸載時觸發。
beforeunload	當頁面即將卸載時觸發，用於提示用戶是否確定離開頁面。
resize	當瀏覽器視窗大小變化時觸發。
scroll	當頁面或元素出現滾動時觸發，通常用於特效（例如 window 或 iframe）。
submit	當表單提交時觸發。
reset	當表單被重設時觸發。
onreset	當表單被重設時觸發，可以包含自定義的功能（如提示或確認）。
focus	當元素獲得焦點時觸發。
blur	當元素失去焦點時觸發。
change	當元素的值發生變化時觸發。
select	選擇文本時觸發。要求重複輸入確保輸入正確，鎖定用戶的複製貼上操作。
input	當用戶在輸入框中輸入內容時觸發。

3.事件物件

target	指向觸發事件的 DOM 元素。例如，當用戶點擊某個按鈕時，target 會指向該按鈕元素。
type	事件的類型（例如：click、mousedown、keydown 等），表示該事件是什麼類型的事件。
clientX	事件發生時，鼠標指針相對於視窗的水平位置，單位為像素。
clientY	事件發生時，鼠標指針相對於視窗的垂直位置，單位為像素。
pageX	事件發生時，鼠標指針相對於文檔的水平位置，單位為像素（包括頁面滾動）。
pageY	事件發生時，鼠標指針相對於文檔的垂直位置，單位為像素（包括頁面滾動）。
preventDefault()	取消事件的預設行為。例如，點擊超連結時可以使用此方法阻止頁面跳轉。
stopPropagation()	停止事件的冒泡，即事件不會繼續向父元素傳遞。這對於避免多個事件處理器在事件冒泡過程中相互干擾很有用。

4.引用'事件物件'

傳統函數 (function) 事件處理器	箭頭函數 (arrow function) 事件處理器
<pre>button物件.addEventListener('click',function(e){ alert(e.target.nodeName)----->button this alert(e.type) ----->click e.preventDefault() //在此函數內，如果有使用事件物件的屬性或方法，就要引用事件物件 })</pre>	<pre>button物件.addEventListener('click', e =>{ alert(e.target) ----->button 物件 this alert(this) ----->window 物件 //arrow func 會因為沒有this而指向更外層的物件 //不要寫傳統函數，要用arrow function })</pre>

this(JS)

this 在 JS代表執行當下的外層物件，所以this在執行的時候才有意義。至於this代表哪一個物件跟當下的情境有關。

this 是 JavaScript 中一個特殊的關鍵字，它指向執行上下文中的當前物件或範圍。在不同的情境下，this 的值是動態的，並且會根據函數的調用方式而改變。

1. this 的定義

在 JavaScript 中，this 代表當前函數或方法被調用時，**執行上下文中的物件**。這個物件會根據函數的呼叫方式而有所不同。

- 在**全局作用域**下，this 通常指向全局物件（在瀏覽器中是 **window**，在 Node.js 中是 **global**）。
- 在**物件方法**中，this 指向調用該方法的物件。
- 在**建構函數**中，this 指向新創建的物件。
- 在**事件處理函數**中，this 指向觸發事件的 DOM 元素。

2. this 的優點

- **2.1 動態綁定**:this 的一個重要優點是它的**動態綁定**特性。根據函數的執行上下文，this 可以指向不同的物件或範圍，使得同一段代碼可以在不同的情境下運行，而不需要硬編碼具體的物件參考。
- **2.2 支持物件導向編程**:this 是物件導向編程中非常核心的一部分。它允許你在物件的方法中引用物件的屬性，並在建構函數中引用新創建的物件，這樣就可以創建具有封裝性、可重用性的物件。
- **2.3 高效的事件處理**:在事件驅動編程中，this 可以讓你快速訪問觸發事件的 DOM 元素，而不需要額外的參數。這對於處理用戶交互非常方便，尤其是在處理大量事件時。
- **2.4 顯式控制 this**:通過 **call**、**apply** 和 **bind** 方法，開發者可以顯式控制 this 的指向，這樣可以在多個物件之間共享方法，或者確保某些方法能夠在特定的上下文中執行。

3. 箭頭函數內並沒有定義this，所以this會往外層找。

- 箭頭函數 中的 this 行為和傳統函數有所不同，箭頭函數的 **this** 是從外層上下文繼承的，而不是根據函數的呼叫來動態綁定。
- 當你創建一個箭頭函數時，this 的綁定是在**函數定義時確定的**，而不是在函數被調用時決定的。

- 在傳統函數中，`this` 的值會根據函數被調用的方式來決定（例如是作為方法呼叫、作為建構函數呼叫，或是綁定 `this`）。
 - 箭頭函數的設計特點之一就是**沒有自己的 `this`**，它會“捕獲”外層作用域中的 `this`，並且保持不變。這是由於箭頭函數的 **靜態綁定** 行為所致。
1. **簡化 `this` 的使用**：尤其是在需要傳遞回調函數或處理異步操作時，箭頭函數可以避免由於 `this` 被重新綁定而造成的錯誤或混亂。
 2. **提高代碼可讀性**：箭頭函數可以讓代碼更簡潔，並且讓 `this` 更容易理解和控制。

物件解構 Object Destructuring

```
const {property, method, variable} = object
```

在 Vue 中，解構賦值（Destructuring Assignment）是一種從數組或物件中提取值並賦給變數的語法。它讓你能夠更方便地從物件或數組中提取資料，特別是在 Vue 組件中，經常會使用解構賦值來簡化程式碼。

為什麼要使用物件解構？

1. **減少冗長的代碼**：物件解構能讓你避免寫多行冗長的賦值語句，讓代碼更加簡潔且易於維護。
2. **提高程式碼可讀性與可維護性**：解構提供了清晰的視覺結構，能夠迅速理解哪些物件屬性被提取出來並且賦值給了變數。
3. **方便重構和擴展**：解構語法非常靈活，尤其是在處理複雜的物件結構時，你可以輕鬆重複命名變數、設置默認值，甚至對嵌套物件進行解構，這使得代碼更具靈活性。

與函數參數配合使用：物件解構也常常與函數參數一起使用，當函數接受一個物件作為參數時，可以直接在參數列表中使用解構，這樣就不需要在函數內部再次手動提取物件屬性。

箭頭函數 Arrow Function

優

1. **簡潔的語法**：箭頭函數的語法非常簡潔，尤其是在處理簡單的回調函數時，不需要使用 `function` 關鍵字和大括號。這使得代碼更加簡潔，易讀性提高。
2. **不會改變 `this` 的綁定**：箭頭函數的 `this` 是靜態綁定的，它會繼承外層函數的 `this`，不會在調用時改變。這在處理回調函數或事件處理器時特別有用，因為你不需要擔心 `this` 被重新綁定到不希望的物件上。

3. **可以作為簡單的回調函數**:在處理簡單的回調函數時，箭頭函數提供了一個非常簡潔且直觀的語法，使得代碼更具可讀性。例如，數組的 **map**、**filter** 和 **reduce** 等方法中的回調函數，使用箭頭函數會更簡單。

缺

1. **不能作為建構函數**：箭頭函數不能用 **new** 關鍵字來創建實例。如果試圖這麼做，會拋出錯誤。
2. **沒有 this、arguments、super 和 new.target**：箭頭函數沒有自己的 this、arguments、super 和 new.target，它們會從外層作用域繼承。
3. **不能使用yield**：箭頭函數不能用作生成器函數，因此不能使用 yield。

立即執行函數表達式 IIFE

傳統函數	<code>function action () {}</code>
IIFE	<code>const action = function() {}</code>
IIFE+箭頭函數	<code>const action = () => {}</code>

- Vue Options API 的this 應該都要指向Vue物件本身，所以**不要使用傳統函數**。因為 this會指向上層的傳統函數，而不會只到Vue物件本身。
- IIFE（**Immediately Invoked Function Expression**）即**立即執行函數表達式**，是一種在定義後立即執行的 Java式。這種模式的主要好處包括：
 1. **避免全局命名空間污染**:將代碼封裝在一個函數內部，使得變數和函數不會暴露在全局範圍內。只有 IIFE 函數內部的代碼能夠訪問這些變數，避免了污染全局命名空間。
 2. **創建私有作用域**:IIFE 可以用來創建一個私有的作用域，將不需要暴露給外部的變數或方法封裝在其中，保護這些數據不受外部干擾。
 3. **立即執行代碼**:IIFE 提供了將函數定義和執行合二為一的方式，使得函數能夠在**定義後立即執行**，而不需要額外調用，這對於初始化代碼或者設置初始化狀態非常有用。
 4. **模組化和封裝**:IIFE 可以用來創建模組化代碼，將不同的功能區域化，並且能夠提供公開的接口同時隱藏內部的實現細節。

Spread | Rest Operator

在 JavaScript 中，**Spread Operator**（擴展運算符）和 **Rest Operator**（剩餘運算符）看起來是相同的語法（...），但它們有不同的功能和用途。它們都是 ES6 引入的特性，讓我們能夠更靈活地處理數據結構（如陣列和物件）。

1. 擴展運算符 Spread Operator 展開&合併

定義： Spread Operator 是用來「展開」一個數組或物件的元素。它會將數組或物件的內容展開成獨立的元素，通常用於函數呼叫、陣列或物件的複製、合併等。

功能：


- **展開陣列或物件：**將一個陣列或物件的元素展開，使其能夠被單獨處理。
- **合併陣列或物件：**可以簡單地將多個陣列或物件合併成一個。
- **複製陣列或物件：**創建陣列或物件的淺拷貝。

優點：

- **簡化語法：**比傳統的 `concat()` 或 `Object.assign()` 等方法更簡潔。
- **避免深拷貝問題：**對於物件或陣列的淺拷貝操作，可以防止不必要的引用複製。
- **合併資料結構：**可以非常方便地合併多個陣列或物件。


用法：

1. 展開陣列：

```
javascript  複製程式碼

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

2. 展開物件：

```
javascript  複製程式碼

const obj = { a: 1, b: 2 };
const newObj = { ...obj, c: 3 };
console.log(newObj); // { a: 1, b: 2, c: 3 }
```



2. 剩餘運算符Rest Operator 擷取與不定量參數

定義： Rest Operator 是用來將函數參數、數組或物件的元素「收集」到一個數組或物件中，通常用於處理不確定數量的參數或從數組或物件中擷取其餘部分。

功能：

- **收集不定數量的參數：**在函數中使用，將多個參數收集成一個數組。
- **從數組或物件中提取剩餘部分：**在解構賦值時，將多餘的元素或屬性收集起來。

2. 陣列解構中使用 Rest Operator：


```
javascript  複製程式碼

const arr = [1, 2, 3, 4];
const [first, ...rest] = arr;

console.log(first); // 1
console.log(rest);  // [2, 3, 4]
```

在這個例子中，`...rest` 收集了 `arr` 中其餘的元素。

3. 物件解構中使用 Rest Operator：

```
javascript  複製程式碼

const obj = { a: 1, b: 2, c: 3 };
const { a, ...rest } = obj;

console.log(a);    // 1
console.log(rest); // { b: 2, c: 3 }
```

在物件解構中，`...rest` 收集了 `obj` 物件中 `a` 以外的屬性。

優點：

- **處理不定數量的參數：**對於函數參數數量不確定的情況，可以將所有參數收集到一個數組中，避免手動處理。
- **簡化數組和物件解構：**可以方便地擷取數組或物件中的特定元素，並將其餘部分收集起來。

- **提高代碼可讀性**：用 Rest Operator 可以簡化和優化解構操作，讓代碼更加直觀易懂。

Import & Export

import 和 export 是 ES6（ECMAScript 2015）引入的模組系統的關鍵部分。它們允許 JavaScript 在不同的檔案之間進行代碼共享和重用。使用 import 和 export，開發者可以更容易地管理大型應用的代碼結構，將代碼分割成模組，並在不同的檔案之間進行引用。

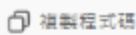
1. Export

定義：export 用來將某個函數、變數或類別等從一個模組中匯出，使其能夠被其他模組所使用。通過 export，可以將模組內的內容公開，使它們可以被外部引入。

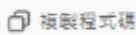
方法：export 有兩種主要形式：

- **具名匯出（Named Exports）**：具名匯出是將具體的變數、函數或類別等標識符匯出，並且在匯入時必須使用相同的名稱。
- **預設匯出（Default Export）**：預設匯出允許每個模組有一個「預設的」匯出項目。預設匯出並不需要具名，匯入時可以使用任意名稱。

範例 1：在定義時匯出

```
javascript   
  
// math.js  
export const add = (a, b) => a + b;  
export const subtract = (a, b) => a - b;
```

範例 2：在結尾匯出

```
javascript   
  
// math.js  
const add = (a, b) => a + b;  
const subtract = (a, b) => a - b;  
  
export { add, subtract };
```

2. Import

1. **定義**：import 用來從其他模組中引入匯出的內容，使它們可以在當前模組中使用。
2. **具名匯入 (Named Import)**：具名匯入是指引入具名匯出的內容，並且名稱必須和匯出時的名稱相匹配。
3. **預設匯入 (Default Import)**：預設匯入用來引入模組的預設匯出項目。在匯入時，可以為預設匯出項目指定任意名稱。

全局元素

document

- `document.images`：獲取文檔中的所有 `` 標籤。
- `document.links`：獲取文檔中的所有 `<a>` 和 `<area>` 標籤。

```
var links = document.links;
console.log(links.length); // 顯示頁面上所有 <a> 和 <area> 標籤的數量
```

- `document.forms`：獲取文檔中的所有 `<form>` 標籤。
- `document.files`：這不是標準屬性，應該是指 `input` 元素中的 `files` 屬性來操作文件選擇。
- `document.getElementById()`：根據 ID 獲取單個 DOM 元素。
- `document.getElementsByClassName()`：根據類名獲取一組 DOM 元素，返回的是一個 `HTMLCollection`。
- `document.getElementsByTagName()`：根據標籤名獲取一組 DOM 元素，返回的是一個 `HTMLCollection`。

```
var divs = document.getElementsByTagName('div');
console.log(divs.length); // 顯示所有 <div> 標籤的數量
```

- `document.querySelector()`：返回匹配指定 CSS 選擇器的第一個元素，若找不到則返回 `null`。

- `document.querySelectorAll()` :返回匹配指定 CSS 選擇器的所有元素，返回的是一個 `NodeList`。
- `document.createElement()` :創建一個新的 HTML 元素節點。

```
var newDiv = document.createElement('div');
newDiv.textContent = '這是一個新的 div 元素';
document.body.appendChild(newDiv);
```

window獲取元素屬性值

`window.getComputedStyle(物件).css屬性`

`window.getComputedStyle()` 是 JavaScript 中的一個方法，用於獲取指定 DOM 元素的計算後樣式。這個方法會返回一個包含所有計算後樣式的 `CSSStyleDeclaration` 對象，你可以通過該對象訪問各種 CSS 屬性。

```
// 獲取該元素
var element = document.getElementById('myElement');

// 獲取該元素的計算後樣式
var computedStyle = window.getComputedStyle(element);

// 訪問不同的 CSS 屬性
console.log(computedStyle.width); // "200px"
console.log(computedStyle.height); // "100px"
console.log(computedStyle.backgroundColor); // "rgb(255, 0, 0)"
```

注意事項：

1. **返回的樣式是計算後的樣式**，這意味著無論元素的 CSS 是內聯樣式、外部樣式還是由 JavaScript 動態修改的，`getComputedStyle()` 都會返回最終應用到該元素上的樣式。
2. 該方法返回的樣式屬性是 **計算後的屬性值**，例如 `width` 可能是像素（`px`）單位，而 `background-color` 可能是 RGB 格式，而不是原始 CSS 屬性中的字符串。

需要注意的 CSS 屬性：

- 某些 CSS 屬性在 `getComputedStyle()` 中會返回不同的格式或需要特別處理（例如 `display: none` 的元素）。
- 透過 `getComputedStyle()` 可以獲取絕大多數屬性，但有些屬性會被忽略，像是 `:hover` 等偽類的樣式，這些需要通過特定的 JavaScript 事件來處理。

ELSE

1. 定義Object時請用`const`修飾字來保護資料
2. id 是 unique ， class 可以標記複數次
3. input number 的step屬性 可以控制在form裡面的小數點後幾位
4. 先呼叫函數，再補宣告:let const 宣告的變數不能先執行再補宣告
5. 在 JS 中，Infinity / Infinity 的結果是 NaN，這是因為這個運算被視為數學上的 **不確定形式** (indeterminate form)，而 JavaScript 會返回 NaN
6. `array.splice(start, deleteCount, item1, item2, ..., itemN)`
 - **start** (必填)：指定開始修改的位置（索引）。如果為正數，表示從數組的開頭數（0 開始）；如果為負數，則從數組尾部倒數的位置開始。
 - **deleteCount** (可選)：指定要刪除的元素數量。如果省略此參數，將刪除從 start 位置開始的所有元素。如果為 0，則不刪除任何元素。
 - **item1, item2, ..., itemN** (可選)：這些是要插入到數組中的新元素。如果沒有提供這些參數，`splice()` 僅會刪除元素而不進行插入。
 - **返回值**：返回一個由被刪除的元素組成的數組。如果沒有刪除任何元素，則返回空數組。
7. 「Duck type」是程式設計中的一個概念
 - 源自於英文「If it looks like a duck, swims like a duck, and quacks like a duck, then it probably is a duck」（如果它看起來像鴨子、游得像鴨子、叫聲像鴨子，那麼它可能就是一隻鴨子）。這句話的意思是，對於一個物件的操作，不需要檢查其具體的類型，而只要檢查它是否擁有執行特定行為所需的方法或屬性。
 - 簡單來說，**duck typing** 是指根據物件是否擁有某些方法或屬性來判斷它的類型，而不是根據它的實際類別來判定。這一概念特別在動態型別語言（如

Python、JS) 中常見。

8. 結構化程式設計

- a. 循序式(sequence) :按照從上到下的順序執程式中的每一條指令。
- b. 迴圈式(Iteration):重複執行某些操作，直到滿足某個條件為止。
- c. 選擇式(selection):根據某個條件來選擇性地執程式中的某些語句。

Author:王彥2024/12/25