箭頭函式

箭頭函式(Arrow Functions)是ES6標準中，最受歡迎的一種新語法。它會受歡迎的原因是好處多多，而且沒有什麼副作用或壞處，只要注意在某些情況下不要使用過頭就行了。有什麼好處呢？大致上有以下幾點:

- 語法簡單。少打很多字元。
- 可以讓程式碼的可閱讀性提高。
- 某些情況下可以綁定 `this` 值。

語法

箭頭函式的語法如下，出自箭頭函數(MDN)：

```
([param] [, param]) => {
   statements
}

param => expression
```

簡單的說明如下:

- 符號是肥箭頭(=>) (註: "->"是瘦箭頭)
- 基本特性是"函式表達式(FE)的簡短寫法"

一個簡單的範例是：

```
const func = (x) => x + 1
```

相當於

```
const func = function (x) { return x + 1 }
```

所以你可以少打很多英文字元與一些標點符號之類的，所有的函式會變成匿名的函式。基本上的符號使用如下說明：

- 花括號({})是有意義的，如果函式沒回傳東西就要花括號。例如 `()=>{}`
- 只有單一個傳入參數時，括號(())可以不用，例如 `x=>x*x`

最容易搞混的是下面這個例子，有花括號({})與沒有是兩碼子事:

```
const funcA = x => x + 1
const funcB = x => { x + 1 }

funcA(1) //2
funcB(1) //undefined
```

當沒用花括號({})時，代表會自動加 `return` ，也只能在一行的語句的時候使用。使用花括號({})則是可以加入多行的語句，不過 `return` 不會自動加，有需要你要自己加上，沒加這個函式最後等於 `return undefined` (註: 這是JavaScript語言中函式的設計)。

綁定 `this` 值

箭頭函式可以取代原有使用 `var self = this` 或 `.bind(this)` 的情況，它可以在詞彙上綁定 `this` 變數。這在特性篇"this"章的內容中有提到。但有時候情況會比較特殊，要視情況而定，而不是每種情況都一定可以用箭頭函式來取代。

原本的範例:

```
const obj = {a:1}

function func(){
```

```
  const that = this

  setTimeout(
    function(){
      console.log(that)
    }, 2000)
}

func.call(obj) //Object {a: 1}
```

可以改用箭頭函式:

```
const obj = {a:1}

function func(){
  setTimeout( () => { console.log(this) }, 2000)
}

func.call(obj)
```

用 `bind` 方法的來回傳一個部份函式的語法，也可以用箭頭函式來取代，範例出自Arrow functions vs. bind():

```
function add(x, y) {
       return x + y
   }

const plus1 = add.bind(undefined, 1)
```

箭頭函式的寫法如下:

```
const plus1 = y => add(1, y)
```

不可使用箭頭函式的情況

以下這幾個範例都是與 `this` 值有關，所以如果你的箭頭函式裡有用到 `this` 值要特別小心。以下的範例都只能用一般的函式定義方式，"**不可**"使用箭頭函式。

使用字面文字定義物件時，其中的方法

箭頭函式會以定義當下的 `this` 值為 `this` 值，也就是window物件(或是在嚴格模式的undefined)，所以是存取不到物件中的 `this.array` 值的。

```
const calculate = {
  array: [1, 2, 3],
  sum: () => {
    return this.array.reduce((result, item) => result + item)
  }
}

//TypeError: Cannot read property 'array' of undefined
calculate.sum()
```

物件的prototype屬性中定義方法時

這種情況也是像上面的類似，箭頭函式的 `this` 值，也就是window物件(或是在嚴格模式的undefined)。

```
function MyCat(name) {
  this.catName = name
}

MyCat.prototype.sayCatName = () => {
```

```
  return this.catName
}

cat = new MyCat('Mew')
// ReferenceError: cat is not defined
cat.sayCatName()
```

DOM事件處理的監聽者(事件處理函式)

箭頭函式的 `this` 值，也就是window物件(或是在嚴格模式的undefined)。這裡的 `this` 值如果用一般函式定義的寫法，應該就是DOM元素本身，才是正確的值。

```
const button = document.getElementById('myButton')

button.addEventListener('click', () => {
  this.innerHTML = 'Clicked button'
})
//TypeError: Cannot set property 'innerHTML' of undefined
```

建構函式

這會直接在用 `new` 運算符時拋出例外，根本不能用。

```
const Message = (text) => {
  this.text = text;
}
// Throws "TypeError: Message is not a constructor"
const helloMessage = new Message('Hello World!');
```

其他限制或陷阱

- 函式物件中的 `call` 、 `apply` 、 `bind` 三個方法，無法"覆蓋"箭頭函式中的 `this` 值。
- 箭頭函式無法用於建構式(constructor)，使用 `new` 會產生錯誤。(上面有範例)
- 箭頭函式沒有一般函式有的隱藏arguments物件。
- 箭頭函式不能當作generators使用，使用 `yield` 會產生錯誤。
- 箭頭函式用於解決一般的 `this` 問題是可以，但並不適用於全部的情況，尤其是在像jQuery、underscore之類有callback(回調)之類的API時，有可能不是如預期般的結果。

參考資料

- Arrow Functions
- TypeScript Deep Dive
- ES6 Arrow Functions: The New Fat & Concise Syntax in JavaScript
- When 'not' to use arrow functions