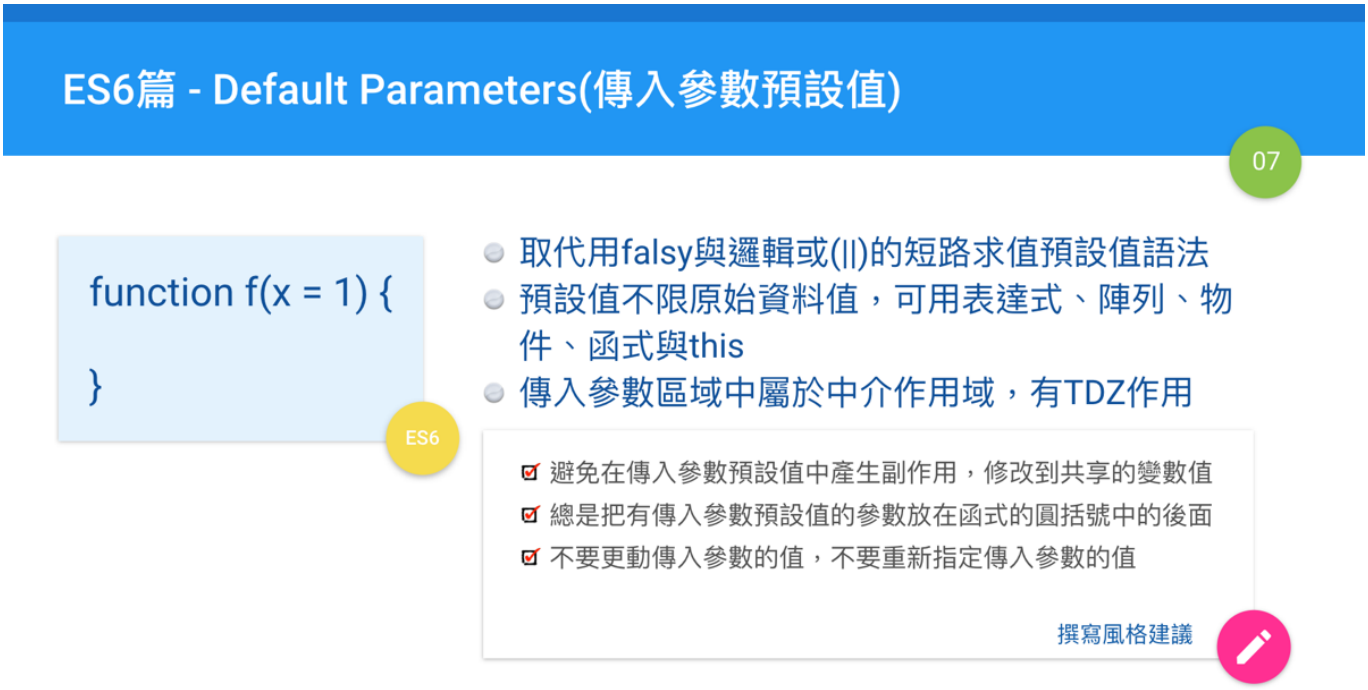
ES6篇 - Default Parameters(傳入參數預設值)



本章的目標是提供ES6中在函式中的傳入參數預設值的一些介紹。傳入參數預設值是在眾多ES6新特性中的其中一個，它改善了長期以來在JavaScript中都要使用技巧性的作法來設定函式傳入參數的預設值的語法。

> 註: 本文章同步放置於Github庫的這裡。

函式的傳入參數是是函式與外部環境溝通的管道，也就是輸入資料的部份。不過，你可能會在函式的"傳入參數"常會看到兩個不同的英文字詞，一個是parameter(或簡寫為param)，另一個是argument，這常常會造成混淆，它們的差異在於:

- parameters: 指的是在函式的那些傳入參數名稱的定義。我們在文章中會以"傳入參數定義名稱"來說明。
- arguments: 指的是當函式被呼叫時，傳入到函式中真正的那些值。我們在文章中會以"實際傳入參數值"來說明。

> 註: 函式中有個隱藏的物件也叫`arguments`這個名稱，所以它是捕捉到所有傳入到函式裡面的所有值的一個物件。

之前的傳入參數預設值怎麼作

你可能會看到書上或教學文章上，告訴你函式的傳入參數預設值是像下面這樣的程式碼範例:

```
//用邏輯或符號(||)
const link = function (point, url) {
    let point = point || 10
    let url = url || 'http://google.com'
    ...
}
```

```
//另一種設定的方式，typeof是回傳類型的字串值
const link = function (point, url) {
    let point = typeof point !== 'undefined' ? point : 10
    let url = typeof url !== 'undefined' ? url : 'http://google.com'
    ...
}
```

函式的傳入參數預設值，在未指定實際的傳入值時，一定是undefined，這與變數宣告後但沒有指定值很相似。而且在函式定義時，我們並沒有辦法直接限定傳入參數的資料類型，所以在函式內的語句部份，一開始都會進行實際傳入參數值的資料類型檢查。所以會使用typeof的回傳值來判斷是否為undefined，或是用邏輯或(||)運算符的預設值設定方式。

使用邏輯或符號(||)的容易造成初學者混亂，這也是一個常在討論區中看到的問題。這是因為JavaScript採用了與其他程式語言不同的邏輯運算的回傳值設計，我們把這兩個運算符，指的是邏輯或(||)以及邏輯與(&&))稱為"短路求值(Short-circuit)"的運算符。在經過邏輯與(&&)以及邏輯或(||)的運算後，它的回傳值是 - 最後的值(Last value)，並不是像在其他常見程式語言例如Java、C++、PHP中的是布林值。

因此，短路求值運算變成JavaScript中一種常被使用的特性，尤其是邏輯或(||)。關於邏輯或(||)運算在JavaScript語言中的可以這樣說明:

> 邏輯或(Logical OR)(||)運算符在運算時，如果當第1個運算子為"falsy"時，則回傳第2個運算子。否則，將會回傳第1個運算子。

也就是說像下面這樣的程式碼，基本上它的回傳值都不是布林值，而是其中一個運算子的值:

```
console.log('foo' || 'bar') // 'foo'
console.log(false || 'bar') // 'bar'
```

falsy的概念是另一個重要的JavaScript概念，Douglas Crockford大師主張使用"**truthy**"與"**falsy**"來描述資料類型的值(在JavaScript: The Good Parts這本書中有這一段)。也就是說，像上面說講的那些會轉換為布林值的false值的資料類型的值，通稱為"falsy"(字典裡是沒這個字詞，意思是"false的")，你可以把它當成是"false家族成員"，大概的定義如下:

> "falsy"包含了0, -0, null, NaN, undefined, ''(空白字串)，當然也一定包含了false值

相對於falsy，就是truthy，也就是不包含上面說的這些之外的值。falsy的概念在JavaScript中的撰寫風格中，可以說到處可見，尤其是與布林相關的邏輯運算語法，像下面這幾個例子:

```
if (isValid) {
  // ...stuff...
}


if (!name) {
  // ...stuff...
}
```

```
if (!collection.length) {
  // ...stuff...
}
```

用falsy與短路求值的概念來作預設值，老實說是一種技巧性的語法，但它並不精確，原因如下：

- 空物件({})與空陣列([])都不是falsy成員之一，實際上對於物件類型的值完全無法判斷
- 如果預設值可以是0或''(空白字串)，甚至可以是null，這種方式判斷不出來，要額外再作判斷

以上大致上就是在ES6這個新特性之前，JavaScript中對於函式傳入參數預設值的一些作法。這些作法已經用了10幾20年有了，一直到ES6後才有新的改進。

ES6的傳入參數預設值

在ES6之後的函式傳入參數中，就可以指定預設值，例如上一節中的例子的改寫過後：

```
const link = function (point = 10, url = 'http://google.com') {
    //...
}
```

傳入參數的預設值順序與傳入參數的位置是一致的，這部份我想不用再多說明。也就是說像上面的函式，如果只有用`link(1)`，後面那個沒給的傳入參數就會看有沒有預設值可用，沒有就為undefined。

這樣的語法會比之前的預設值語法來得更佳簡單明確，除此之外，傳入參數預設值它與之前使用短路求值的作法有一個最大的不同之處：

> 只有在傳入參數為undefined時或是不存在，才會使用預設值

當然，我把undefined與不存在寫成兩個情況，undefined對JS來說，是一種原始的資料類型值。有的時候例如像在物件中存取某個還未定義的屬性，也是相當於undefined值。所以上面說的"不存在"大概是指這種情況。

> 注意：只有undefined才會觸發傳入參數預設值的指定運算，null並不會。這一點要特別注意，尤其你會用到物件類型的值時。

ES6的傳入參數預設值的設計，與許多程式語言中類似的設計都有點不太一樣。以下分別把重點的不同之處列出來。

傳入參數預設值裡可以用表達式

在短路求值的邏輯或(||)中可以使用表達式，並沒有說只能用固定的值。在傳入參數預設值中的也有這種設計，下面的語法是可以這樣用的：

```
function foo(a = 0, b = a+100) {
  console.log(a+b)
}
```

```
foo() // 100
foo(1)  // 102
foo(1, 2) // 3
```

用呼叫其他的函式當傳入參數的預設值也是可以的，例如下面的例子:

```
function go(x) {
  return x + 10
}

function foo(a = 1, b = go(a)) {
  console.log(a+b)
}

foo()  // 30
foo(1)  // 12
foo(1, 2) // 3
```

> 特別注意: 避免在傳入參數預設值中造成副作用，例如像`function foo(a = 1, b = ++a) { console.log(a+b) }`，會造成`a`被修改到，你可能會得到不預期的結果。

傳入參數預設值的求值是每個函式呼叫獨立的(Evaluated at call time)

有些程式語言(例如Python)是定義期間決定傳入參數的預設值，但ES6的設計是函式呼叫時才決定的，也就是在執行期間才進行設定，對每個函式呼叫都是獨立的。下面的範例來自這裡:

```
function foo(x = []) {
  x.push(1)
  console.log(x)
}

foo() // [1]
foo() // [1]
foo() // [1]
```

這個設計可以免除掉在傳入參數預設值使用了複雜的物件類型值時，如果在函式中對傳入參數進行"具有副作用"求值運算的一些情況。當然，副作用是開發者要想辦法儘量避免的。

傳入參數預設值會隱蔽外在的作用域

傳入參數預設值會除蔽外在作用域的同名變數/常數，這也為了避免一些混亂的情況。下面的範例來自這裡:

```
let x = 1

function foo(x, y = x) {
  console.log(y)
}
```

```
foo(2) // 2, not 1!
```

傳入參數預設值的TDZ(Temporal Dead Zone, 時間死區)作用

在let與const章節中，我們有看到TDZ(Temporal Dead Zone, 時間死區)的作用，同樣的在傳入參數預設值中，也有類似的作用存在。

下面的例子可以看到在傳入參數預設值的識別名稱，在未經初始化(有指定到值)時，它會進入TDZ，因而產生錯誤，而這個錯誤是只有在函式呼叫時，要使用傳入參數預設值時才會出現:

```
function foo(x = y, y = 1) {
  console.log(y)
}

foo(1) // 這不會有錯誤
foo(undefined, 1) // 錯誤 ReferenceError: y is not defined
foo() // 錯誤 ReferenceError: y is not defined
```

從這個例子可以知道TDZ(Temporal Dead Zone, 時間死區)的作用不只有在let與const的設計上有，實際上在ES6中都有類似的作用。對於傳入參數預設值它的作用域到底是全域作用域還是函式中的作用域的議題，較正確的說法是它是處於中介的作用域，夾在這兩者之間，但仍然會互相影響。

本節的內容可以參考這幾篇文章TEMPORAL DEAD ZONE (TDZ) DEMYSTIFIED、ES6 Notes: Default values of parameters與這個Default parameters intermediate scope討論文。

傳入參數預設值使用物件、其他函式與this時

傳入參數預設值沒有限定一定只能用原始資料類型的值，你也可以使用物件、陣列、其他函式(通常是callback)當預設值，甚至是使用this。

我們用物件類型的值作為傳入參數傳到一個函式時，這個函式中如果有使用this的情況，因為一般的函式在呼叫時，都是在以window或全域物件來作為預設this值。傳入參數預設值按照規則也是如此，下面的例子只是怕你會開始搞混，所以把三種呼叫的情況列出來，順便複習一下:

```
function foo(x = this, y = this.value) {
  console.log(x)
  console.log(y)
}

foo()
foo({value: '= =+'})
foo.call({value: '^^y~'})
```

第一個呼叫是直接呼叫`foo()`，this會指到window或全域物件中，x相當於window或全域物件。y沒這屬性存在，所以是undefined。

第二個呼叫用了一個物件帶有value屬性，作為傳入參數。按照傳入參數，x會是這個物件，y沒給所以用預設值，y是this.value，那麼y應該與第一個呼叫一樣嗎？是的，因為此時this會指到window或全域物件中，y沒這屬性存在，所以是undefined。

第三個呼叫用了call方法，裡面是一個物件帶有value屬性，作為呼叫這個函式的擁有者物件，所以this會指到這個物件，x為這個物件，y是裡面的value值，也就是'^^y~'字串。

使用其他函式作為傳入參數預設值也是很常見的情況，通常會是一個callback(回調、回呼)函式，一般的情況沒什麼特別。不過這個地方有可能會遇到上一節說到的中介作用域(或稱為傳入參數作用域)的情況，例如以下的例子，這個例子來自這裡:

```
let x = 1

function foo(a = 1, b = function(){ x = 2 }){
  let x = 3
  b()
  console.log(x)
}

foo()

console.log(x)
```

這個例子中的最後結果，在函式foo中輸出的x值到底是1、2還是3？在最外圍的x會被改變嗎？

函式中的x不可能是1，已經很明顯，因為函式區塊中有另一個x的宣告與指定值let x = 3。剩下的是傳入參數預設值中的那個函式，是不是會變數到函式區塊中的x值的問題。在全域中的那個x變數，會不會被改變？這也是一個問題。

我只能說這就像騎機車如果在某個小路口，沒有待轉區一樣是待轉再左轉有沒有違反交通規則一樣，按照這個例子的出處文章的說明，作者認為答案是3與1。但是…下面的幾個瀏覽器與編譯器就不是這樣認為:

- babel編譯器: 2與1
- Traceur/Closure Compiler: 3與2
- Google Chrome(v54): 3與2
- Firefox(v50): 2與1
- Edge(v38): 3與2

實際測試，怎麼樣都不會有3與1的答案，要不就3與2，要不就2與1。

3與2的答案是讓b傳入參數的x=2執行出來，但因為受到中介作用域的影響，因此干擾不到函式中的原本區塊中的作用域，但會影響到全域中的x變數。

2與1的答案則是倒過來，只會影響到函式中的區塊，對全域沒有影響。

所以除非中介作用域完全與函式區塊與全域都不相干，才有可能產生3與1的結果，這是這篇文章的作者所認為的。

不管如何，這個作用域的影響仍然是有爭議的，答案並沒有統一。如果你研究到這個程度，代表你已經開始看到ES6雖然標準定好了，但裡面的一些新特性仍然有實作上的差異，這些都只是學術研究領域的範圍，這些情

況要儘量避免，以免產生不相容的情況。

要如何避免這種情況？最首要的就是，"不要在傳入參數預設值中作有副作用的運算"，上面的`function(){ x = 2 }`是有副作用的，它有可能會改變函式區塊中，或是全域中的同名變數。保持良好的習慣不管怎麼樣都是很重要的，騎機車要不要待轉是一回事，直接左轉其實危險是你自己。

必要的傳入參數

如果你想要讓某個傳入參數是必要的，如果在使用時沒給傳入參數就報錯，這也是可以作得到的。有兩種方式，因為傳入參數在傳入值為undefined時才會作值指定運算，所以你可以在函式區塊中檢查，或是用另一個函式來專門用於報錯使用，這個樣式是比較接近使用傳入參數預設值的風格。以下的範例是這兩種方式:

```
//方式一
function func1(x){
  if(x === undefined) throw new Error('Missing parameter')

  //...
}

//方式二
function throwIfMissing() {
    throw new Error('Missing parameter');
}

function func2(x = throwIfMissing()){
  //...
}
```

其他情況的使用說明

有一些其他的使用時注意事項，以及後面的章節會再說明的，先簡單的列在這個地方。

預設值的位置並沒有限制

傳入參數的預設值並沒有限定個數，或是一定是只能用在哪一個傳入參數(有語言會限定在最後一個)，使用上也沒有限定位置。不過，有一些撰寫風格會認為如果一整串的傳入參數，有預設值的要往後寫會比較清楚。例如以下的的例子，來自Airbnb 7.9:

```
// 壞寫法
function handleThings(opts = {}, name) {
  // ...
}

// 好寫法
function handleThings(name, opts = {}) {
  // ...
}
```

箭頭函式中也可以使用

上一次介紹的箭頭函式也可以使用傳入參數預設值，不過箭頭函式的撰寫風格是當只有一個傳入參數時，傳入參數部份可以不加圓括號(())，如果你用了傳入參數預設值，就一定要加圓括號，例如以下的例子:

```
const func = (x = 100) => x * 2
```

傳入參數預設值中可以使用解構賦值(destructing assignment)

解構賦值在後面的章節會介紹，你有可能會在新式的函式庫看到有人這樣使用，這都會是在傳入參數是個物件或陣列類型值的情況，例如以下的這兩個例子:

```
function func({a, b} = {a: 1, b: 2}) {
  console.log(a, b)
}

func()
func({a: 5})  // 5 undefined
func({a: 10, b: 22})
```

```
function func({a=1, b=2} = {}) {
  console.log(a, b)
}

func()
func({a: 5})  // 5 2
func({a: 10, b: 22})
```

> 註: 解構賦值也會用在陣列上，之後章節會再介紹。如果你有興趣可以先參考我寫的電子書中的這篇解構賦值

撰寫風格建議

- 避免在傳入參數預設值中產生副作用，修改到其他共享的變數值。(Airbnb 7.8, Google 5.5.5.1)
- 總是把有傳入參數預設值的參數放在函式的圓括號中的後面。(Airbnb 7.9)
- 不要更動傳入參數的值，不要重新指定傳入參數的值。(Airbnb 7.12/7.13, eslint: no-param-reassign)

結論

我原本認為的一小篇新特性介紹，最後竟然寫了這麼多的內容。這篇文章大概有8千多字，扣掉程式碼的部份大概也打了千字有。可見這個小小的ES6新特性，裡面有太多的細節，如果要研究得透徹，也是需要花不少時間的。這個改進我個人認為是一個很棒的新特性，相信在未來的JavaScript函式庫或框架中，都會使用到它。