ES6篇: Spread Operator & Rest Operator(展開與其餘運算符)

本章的目標是對展開運算符(Spread Operator)與其餘運算符(Rest Operator)提供一些使用上的說明。這些語法在React、React Native、Redux等等新式的函式庫應用上非常常見，是一個必學的語法。要注意的是，有些語法超出了ES6標準的範圍，本文的最後面有提供一些說明。

> 註: 本文章同步放置於Github庫的這裡。

展開運算符(Spread Operator)與其餘運算符(Rest Operator)是ES6中的其中兩種新特性，雖然這兩種特性的符號是一模一樣的，都是三個點(...)，但使用的情況與意義不同。我們常常在文字敘述或聊天時，這個(...)常用來代表了"無言"、"無窮的想像"或"還有其他更多的"的意思。

簡單摘要一下這個語法的內容：

- 符號都是三個點(...)
- 都是在陣列值運算
- 一個是展開陣列中的值，一個是集合其餘的值成為陣列

> 註: 三個點符號(...)，比較正式的英文字詞是Ellipsis，翻成中文是"省略符號"，不過它有各種形式(全形或半形)，也有超過三個點的情況，所以一般要說得明白只有三個點的情況，會用"three dots"與"dot-dot-dot"會更為明確，本文使用"三個點符號"的說法。

> 註: 目前看到的這種語法的名詞上並沒有統一。例如在ES6標準規格上的用語是SpreadElement與rest parameter，也沒有集合成為一個章節，內容會散落在各章節中。在MDN上用Spread operator與Spread syntax的名詞講法(標題是syntax，網址列上是operator)，以及Rest operator與Rest parameters(標題是parameters，但連結是operator)。

> 註: 用於物件上的類似語法並不是ES6中的特性，它是正在制定中的新語法標準，暫時稱它為ES7+標準。不過在React與Redux中很常見，這種語法可以透過babel編譯，稱為Object Rest/Spread Properties。

展開運算符(Spread Operator)

> 展開運算符是把一個陣列展開成個別的值的速寫語法，它只會在"陣列字面定義"與"函式呼叫"時使用

展開運算符(Spread Operator)是把一個陣列展開(expand)成個別值，這個運算符後面必定接著一個陣列。最常見的是用來組合(連接)陣列，對應的陣列方法是 `concat` ，以下是一個簡單的範例：

```
const params = [ "hello", true, 7 ]
const other = [ 1, 2, ...params ] // [ 1, 2, "hello", true, 7 ]
```

展開運算符可以作陣列的淺拷貝，當然陣列的淺拷貝有很多種方式，這是一種新語法，也是目前最簡單的一種語法:

```
const arr = [1,2,3]
const arr2 = [...arr]

arr2.push(4) //不會影響到arr
```

> 註: 淺拷貝(shallow-copy)對於陣列中的陣列值(多維陣列)，或是有複雜的物件值情況時，是只會拷貝參照值而已。

> 註: 上述的展開運算符在陣列字面中使用時，並沒有限制位置，或是個數。像 `const arr = [...a, 1, ...b]` 這樣的語法都是可以的。

你也可以用來把某個陣列展開，然後傳入函式作為傳入參數值，例如下面這個一個加總函式的範例：

```
function sum(a, b, c) {
  return a + b + c
}
const args = [1, 2, 3]
sum(…args) // 6
```

對照ES5中的相容語法，則是用 `apply` 函式，它的第二個參數也是使用陣列，以下是用ES5語法與上面相同結果的範例程式：

```
function sum(a, b, c) {
  return a + b + c;
}

var args = [1, 2, 3];
sum.apply(undefined, args) ;// 6
```

展開運算符還有一個特別的功能，就是把可迭代(iterable)或與陣列相似(Array-like)的物件轉變為陣列，在JavaScript語言中內建的可迭代(iterable)物件有String、Array、TypedArray、Map與Set物件，而與陣列相似(Array-like)的物件指的是函式中的隱藏物件"arguments"。下面的範例是轉變字串為單字串的陣列:

```
const aString = "foo"
const chars = [ ...aString ] // [ "f", "o", "o" ]
```

下面的範例是把函式中的隱藏偽物件"arguments"轉成真正的陣列:

```
function aFunc(x){
  console.log(arguments)
  console.log(Array.isArray(arguments))

  //轉為真正的陣列
  const arr = [...arguments]
  console.log(arr)
  console.log(Array.isArray(arr))
}

aFunc(1)
```

其餘運算符(Rest Operator)

> 其餘運算符是收集其餘的(剩餘的)這些值，轉變成一個陣列。它會用在函式定義的傳入參數識別名定義(其餘參數, Rest parameters)，以及解構賦值時

其餘運算符(Rest Operator)的主要用途會用在兩個地方，一個是比較常提及的在函式定義中的傳入參數定義中，稱之為其餘參數(Rest parameters)。另一種情況是用在解構賦值時。在這兩個地方的功用都是同樣的意義。

其餘參數(Rest parameters)

> 就像在電影葉問中的台詞："我要打十個"，其餘參數可以讓你一次打剩下的全部，不過葉問會變成一個陣列。

既然是一個參數的語法，當然就是用在函式的傳入參數定義。其餘參數代表是將"不確定的傳入參數值們"在函式中轉變成為一個陣列來進行運算。例如下面這個加總的範例：

```
function sum(…numbers) {
  const result = 0

  numbers.forEach(function (number) {
    result += number
  })

  return result
}

sum(1) // 1
sum(1, 2, 3, 4, 5) // 15
```

> 特別注意: 其餘參數在傳入參數定義中，必定是位於最後一位，並且在參數中只能有一個其餘參數。

其餘參數的值在沒有傳入實際值時，會變為一個空陣列，而不是 `undefined` ，以下的範例可以看到這個結果:

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

aFunc(1,2,3) //x = 1, y = [2, 3]
aFunc() //x = undefined, y = []
```

其餘參數的設計有一個很明確的用途，就是要取代函式中那個隱藏"偽陣列"物件 `arguments` ， `arguments` 雖然會包含了所有的函式傳入參數，但它是個類似陣列的物件卻沒有大部份陣列方法，它不太像是個隱藏的密技，比較像是隱藏的陷阱，很容易造成誤解或混亂，完全不建議你使用 `arguments` 這個東西。

其餘參數的值是一個真正的陣列，而且它需要在傳入參數宣告才能使用，至少在程式碼閱讀性上勝出太多了。

解構賦值(destructuring)時

解構賦值在另一獨立章節會講得更詳細，這裡只是要說明其餘運算符的另一個使用情況。解構賦值也是一個ES6中的新特性。

解構賦值是用在"陣列指定陣列"或是"物件指定物件"的情況下，這個時候會根據陣列原本的結構，以類似"鏡子"對映樣式(pattern)來進行賦值。聽起來很玄但用起來很簡單，這是一種為了讓陣列與物件指定值時更方便所設計的一種語法。例如以下的範例:

```
const [x, y, z] = [1, 2, 3]

console.log(x) //1
```

像這個例子就是最簡單的陣列解構賦值的範例，x當然會被指定為1，y與z你應該用腳底板也想得到是被指定了什麼值。

當使用其餘運算符之後，就可以用像其餘參數的類似概念來進行解構賦值，例如以下的範例:

```
const [x, ...y] = [1, 2, 3]

console.log(x) //1
console.log(y) //[2,3]
```

當右邊的值與左邊數量不相等時，"鏡子對映的樣式"就會有些沒對到，用了其餘運算符的那個識別名稱，就會變成空陣列。就會像下面這個例子一樣:

```
const [x, y, ...z] = [1]

console.log(x) //1
```

```
console.log(y) //undefined
console.log(z) //[]
```

在函式傳入參數中作解構賦值，這個例子的確也是一種解構賦值的語法，而且加了上一節的函式中的其餘參數的用法。例子出自MDN的這裡:

```
function f(...[a, b, c]) {
  return a + b + c;
}

f(1)          // NaN (b and c are undefined)
f(1, 2, 3)    // 6
f(1, 2, 3, 4) // 6 (the fourth parameter is not destructured)
```

你可以回頭再看一下"其餘參數"的使用情況，是不是與解構賦值時很相似。

> 特別注意: 在使用解構賦值時一樣只能用一個其餘運算符，位置也只能放在最後一個。

ES7+的其餘屬性(Rest Properties)與展開屬性(Spread Properties)

上面都只有談到與陣列搭配使用，但你可能會看到在物件上也會使用類似語法與符號(...)，尤其是在React與Redux中。這些都是還在制定中的ES7之後的草案標準，稱為其餘屬性(Rest Properties)與展開屬性(Spread Properties)。例如下面這樣的範例，來自這裡：

```
// Rest Properties
let { x, y, ...z } = { x: 1, y: 2, a: 3, b: 4 }
console.log(x) // 1
console.log(y) // 2
console.log(z) // { a: 3, b: 4 }

// Spread Properties
let n = { x, y, ...z }
console.log(n) // { x: 1, y: 2, a: 3, b: 4 }
```

有些新式的框架或函式庫中已經開始使用了，babel轉換工具可以支援轉換這些語法，但使用時要注意要額外加裝 `babel-plugin-transform-object-rest-spread` 外掛。

撰寫風格建議

- 不要使用函式中的arguments物件，總是使用其餘參數語法來取代它。(Airbnb 7.6, Google 5.5.5.2, eslint: prefer-rest-params).
- 不要在展開運算符與其餘運算符後面有空格，也就是與後面的識別名稱(傳入參數名稱、陣列名稱)之間要緊接著。(Google 5.2.5/5.5.5.2, eslint: rest-spread-spacing)
- 用展開運算符的語法來作拷貝陣列。(Airbnb 4.3)
- 用展開運算符的語法來取代函式中的 `apply` 的語法，作不定個數傳入參數的函式呼叫。(Airbnb 7.14, eslint: prefer-spread)
- 用展開運算符的語法來取代 `slice` 與 `concat` 方法的語法。(Google 5.2.5)
- 優先使用物件展開運算符(object spread operator)的語法取代Object.assign，來作物件的淺拷貝。(Airbnb 3.8) (ES7+標準)

結論

展開運算符比較容易理解，它是把已有(看得到)的陣列值"展開"為一個一個單獨的值。其餘運算符都是用在函式定義或指定值時，它是要收集其餘的(剩餘的)值，形成一個陣列再來進行運算，你可能還對展開運算符與其餘運算符的分別還有點混亂，因為符號都是相同的三個點符號(...)，只是在不同的使用情況下功用是不相同的，所以要區分它們要從使用情況來區分:

- 展開運算符: 用在陣列的字面文字定義裡面(例如 `[1, ...b]`)，或是**函式呼叫**時(例如 `func(...args)`)
- 其餘運算符: 用在函式的定義，裡面的傳入參數名稱定義時(例如 `function func(x, ...y)`)。或是在解構賦值時(例如 `const [x, ...y] = [1,2,3]`)

參考資源

- Spread syntax(MDN)
- Rest parameters
- Exploring ES6 - 10.7.2 Rest operator
- Exploring ES6 - 11.8 The spread operator
- ES6—default + rest + spread
- ES6 Spread and Butter in Depth