

 eyesofkids / ironman2017

Code

Issues 2

Pull requests

Actions

Projects

Wiki

Security

Insights

 master ▾

...

ironman2017 / day05_es6_let_const /



eyesofkids ...

on 5 Dec 2016



..



asset

4 years ago



README.md

4 years ago

README.md

鐵人賽第 5 天: ES6篇 - let與const

ES6篇 - let 與 const

05

let

- 變數宣告
- 在for迴圈中每次循環都會重新綁定

const

- 常數宣告
- 不可再重覆指定(can't re-assignment)
- 物件、陣列、函式等參照類型使用

- 區塊作用域(block scope)
- 有提升特性(Hoisting)，但因Temporal Dead Zone(TDZ, 時間死區)的設計，會產生錯誤

 共通特性

很快地進入到鐵人賽的第5天，也進入到ES6篇的章節之中。本章的目標是針對let與const這兩個用於宣告變數與常數的語句，作重點式的介紹。有很多是整理我個人從討論區的回答文章。

註: 本文章同步放置於[Github庫的這裡](#)。

關於ES6

因為是ES6篇的第一篇，所以稍微介紹一下ES6標準。

ES6是ECMAScript第6版本的簡稱，又稱為ES2015，因為它是去年(2015年)才正式定案的一個標準。ECMAScript是Javascript程式語言的規格標準，以下是各版本的發行日期，但標準發行並不代表在各個瀏覽器品牌中，即可使用其中的功能規格，還需等待各瀏覽器品牌廠商進行實作。

- ECMAScript 6 (ES6) 發行於2015年中，為目前最新的官方版本
- ECMAScript 5 (ES5) 發行於2009年底
- ECMAScript 4 (ES4) 棄用
- ECMAScript 3 (ES3) 發行於1999年底

Ecma全名為歐洲計算機製造商協會(European Computer Manufacturers Association)，設立了一個TC39委員會，專門為制定ECMAScript標準進行討論與審核哪些功能要列入標準。TC39的介紹可以看這篇[TC39 - ECMAScript](#)，而TC39的制定流程可以參考[The TC39 Process](#)這篇。目前有很多新的功能標準都正在制定當中。

ES6以[ECMAScript compatibility table\(相容性表格\)](#)網站來看目前的瀏覽器品牌與版本的相容程度，新式的像Chrome、Safari、Edge與Firefox支援性都已達到90%以上，但在IE11(桌面)、Android、iOS9手機內建上，支援性還是非常的差。這也是目前為什麼要先用babel編譯器先轉為ES5標準的程式碼的主要原因。ES5的程式碼可以取得目前市面上最大的相容性，從IE9以上，一直到Android 4.1與iOS6都可以支援得很好。

ES6加入了非常多的新特性，可以說整個提升JavaScript程式語言的內建特性的質與量，有許多新的特性是在現代程式語言中非常重要而不可缺少的。而有一些是新的內建物件或類型，它們是屬於較為進階的內容，在某些使用情況下會特別有用。由於在ES6標準最終發行版本的差不多時間前，HTML5標準也發行了最後的版本，它的制定組織是W3C，有許多JavaScript中的特性是與HTML5的標準息息相關的，尤其是在網頁上與DOM元件或事件相關的許多方法。所以除了ES6中的眾多新特性，目前也有很多是來自於HTML5的新特性，這些特性在目前的新式瀏覽器中都逐漸被實作出來，這也是近幾年JavaScript語言蓬勃發展的結果。

let與const

let 與 const 無疑是要取代原本的使用 var 語句來定義變數與常數。在ES6之前，並沒有"常數"這個東西，只有"變數"而已，也就是用 var 所宣告的識別名稱。在這份舊的[Google JavaScript樣式風格指引](#)中，會告訴你要用全大寫英文字元來作為常數定義，像是 var MAX_HEIGHT = 10 這種定義方式，甚至是使用非常特別的註解中的 @const 標記，有可能Google Chrome瀏覽器會在內部處理時認得它是個常數，但很少見到有人這樣用，畢竟也只有對Chrome有用而已。對JavaScript來說，用 var 來宣告就是個變數。

實際上在多年的只有 `var` 可用，加上其他亂用亂寫的語法下，在JavaScript中共有4種宣告變數的方式，它們分別是：

```
a = 10
this.a = 10
window.a = 10
var a = 10
```

註：關於上面的這麼多種變數宣告的語法，只是提出來說明而已，如果有興趣請參考我在segmentfault討論區中[這篇回答](#)或是這篇[Stack Overflow上的問答](#)

上面的寫法只有最後一個 `var a = 10` 才是真正的變數宣告，其他的都是只在全域物件中加入屬性。從這裡就可以知道，JavaScript的自由度令人覺得可怕，用 `a = 10` 就可以直接生出一個變數來用，而且是在全域中產出變數。所以需要各種撰寫風格與輔助工具來協助開發者，這是我一直強調的部份。

雖然在ES5中已經加入了嚴格模式(strict mode)的設計，對 `this` 的存取使用會變得沒那麼自由，當然，有些老的舊的程式碼會變得無法執行，這是它們本來需要作調整與改變的，實際上經過bebel編譯工作編出來的ES5程式碼，預設都會加上嚴格模式(strict mode)。

`let` 與 `const` 是區塊作用域(block scope)，而 `var` 是函式作用域(function scope)，這是第一個我們會看到的差異性。區塊作用域與函式作用域會差在什麼地方？

`var` 是函式作用域的設計，也就是說它只能以函式為變數作用域的分界，在一些使用了區塊語句(用花括號的語句)的像if, else, for, while等等區塊語句中，在這裡面用 `var` 宣告的變數仍然是會曝露到全域之中可被存取，例如：

```
function test(){
  var a = 10
}

if(true){
  var b = 20
}

console.log(a) // a is not defined 存取不到
console.log(b) // 存取得到
```

這對初學者容易造成誤解外，如果再搭配到隱藏的提升特性(最下面有說明)，整個程式碼經常會有出人意表的結果。在許多撰寫風格指引通常會提醒這點，而且叫你一定要把 `var` 語句寫在程式碼檔案的最上面。(甚至連for語句中的 `var` 宣告也要寫到最上面)

如果使用了 `let` 或 `const` 來宣告，就是以區塊語句為分界的作用域，它會比較明確而且不易發生錯誤。一些之前對於 `var` 語句的麻煩撰寫風格，就可以不需要了。從下面的例子可以得到：

```
function test() {  
  let a = 10  
}  
  
if (true) {  
  const b = 20  
}  
  
console.log(a) // a is not defined 存取不到  
console.log(b) // b is not defined 存取不到
```

總之，不要再用 `var` 了，用 `let` 或 `const` 來取代它就是了。像我們有使用的ESLint檢查工具，一定出現會叫你不要使用 `var` 的訊息。

const

`const` 針對是常數的定義，常數在一宣告時就必定要指定給值，不然會產生錯誤。而對於常數在ES6的定義是：

不可再指定(can't re-assignment)

指定的意思就是用等號(=)作指定運算，像下面這例子就是再指定值(或重覆指定值)，所以會產生錯誤：

```
const a = 10  
a = 20 // TypeError: Assignment to constant variable. 錯誤
```

註：JS中的指定運算符除了等號(=)外，還有多種等號(=)與其他運算符組合而成的指定運算符，請參考MDN上面的這篇[Assignment_Operators](#)

宣告了一個常數，代表這個識別名稱的參照(reference)是唯讀的(read-only)，並不代表這個參照指定到的值是不可改變的(immutable)。這是在講什麼？這是在講如果你宣告的常數是一個物件或陣列類型，像這種參照類型的值，裡面的值是可以作改變的。像下面的例子都是合法的使用：

```
const a = []  
a[0] = 1  
  
const b = {}  
b.foo = 123
```

所以對於物件、陣列、函式來說，使用 `const` 常數來宣告就可以，除非你有需要再指定這個陣列或物件的參照。

註: 既然可以用`const`來宣告物件與陣列，用全英文字元全大寫來命名常數，也已經不需要，像一般的正常命名變數就可以了，一般用小駝峰(`camelCase`)命名法，例如 `myBook`、`userName` 這樣。

let使用於for語句

這一段有兩個重點，第一個只是有可能被誤解所以提出來說明。第二個是會重新綁定這個特性。

for圓括號中的let變數仍然是在區塊作用域

這很容易從例子中理解，寫出來只是怕有些剛學的朋友，不確定在for圓括號中的第一個表達式，用`let`宣告變數時，是不是也會是被限制到for語句的區塊中作用域。答案是**"會的"**，見下面的例子：

```
for (let i = 0; i < 10; i++) {  
  console.log('in for statement: i', i)  
}  
  
console.log(i) // ReferenceError: i is not defined(...) 存取不到
```

for迴圈中的let變數會作重新綁定

註: 這一段內容是改寫自我在segmentfault討論區的[這篇問答](#)

這個特性算是 `let` 的特別之處，這是由於區塊作用域造成的結果，在每次的for迴圈執行時，用 `let` 宣告的變數都會重新綁定(re-bind)一次。這是在for語句中的 `var` 與 `let` 的差異：

`for (let x...)`的循環在每次迭代時都為 `x` 建立新的綁定

以下用程式碼直接看會比較容易的理解。這個改進主要是為了解決在for語句中的閉包結構的問題。

原來的使用 `var` 的程式碼，與去糖(de-sugar)後來看它在執行時是這樣的模擬執行代碼：

```
// 原來程式碼  
for (var i = 0; i < 10; i++) { setTimeout(()=>console.log("i:",i), 1000) }  
  
// 不需要加區塊符，因為區塊也不會影響  
var i  
i = 0;  
if (i < 10)  
  setTimeout(()=>console.log("i:",i), 1000)  
  i++  
  if (i < 10)  
    setTimeout(()=>console.log("i:",i), 1000)
```

```

    i++
//...

```

而使用了 `let` 後，會有區塊作用域的影響，原來的程式碼與執行時的去糖模擬執行代碼如下：

```

// 原來程式碼
for (let i = 0; i < 10; i++) { setTimeout(()=>console.log("i:",i), 1000) }

// 用區塊符區分每次循環的語句
// 每次for語句開始，i指定為一個全域刻度__status，這只是方便說明而已
// __status會記錄for語句i最後的值
{ let i
  i = 0
  __status = {i}
}
{ let {i} = __status
  if (i < 10)
    setTimeout(()=>console.log("i:",i), 1000)
    __status = {i}
}
{ let {i} = __status
  i++
  if (i < 10)
    setTimeout(()=>console.log("i:",i), 1000)
    __status = {i}
}
//...

```

為何可以這樣模擬？因為在ES6標準中，有一段是關於`CreatePerIterationEnvironment`，也就是for語句每次循環所要建立環境的步驟，裡面有提及有關詞法環境(LexicalEnvironment)的相關步驟，這與使用 `let` 時會有關。所以如果你使用了 `let` 而不是 `var`，`let` 的變數除了作用域是在for區塊中，而且會為每次循環執行建立新的詞法環境(LexicalEnvironment)，拷貝所有的變量名稱與值到下一個循環執行。以最簡單的方式改寫原先的程式碼，相當於下面這樣寫：

```

let k
for (k = 0; k < 10; k++) {
  let i = k //注意這裡，每次循環都會建立一個新的i變數
  setTimeout(() => console.log("i:", i), 1000)
}

```

提升(Hoisting)

註：這一段的說明改寫自我在[segmentfault討論區的這篇問答](#)

變數/常數、函式(主要是指函式定義方式, FD)、類別都有提升的特性。這是因為在執行的過程, 也就是執行上下文(EC)在建立的過程時, 就有一個階段是稱為綁定(binding)的步驟, 在這步驟時會尋找目前要建立的這個上下文(EC)的變數、函式、類別等資訊, 這一段在ECMAScript標準中, 有一節稱為[Declaration Binding Instantiation\(宣告綁定初始化\)](#), 其中就有提到它的規則, 以下簡單翻一段:

每個執行上下文(EC)都會有個關聯的VariableEnvironment(變量環境)。被評估需要包含在執行上下文(EC)中的在ECMAScript程式碼裡所定義的變數與函數, 會被加入到VariableEnvironment(變量環境)的Environment Record(環境記錄)裡成為綁定。對於函式的程式碼來說, 傳入參數也會加入到Environment Record(環境記錄)成為綁定。

Environment Record(環境記錄)就是執行上下文(EC)用來綁定宣告的記錄部份, 其中又分兩類declarative environment records與object environment records...

所以在JS引擎在執行時, 會先作宣告部份的綁定, 這個步驟時會對已經存在記錄里的聲明進行掃描, 如果已有記錄的, 新的宣告會覆蓋掉已有的。

下面是綁定步驟的一個的程式碼範例, 參考自[這篇文章](#):

```
var foo

;(function(){
  foo = 1    // 此步驟時, 沒有要綁定的
  var foo    // 清除之前綁定的, 然後新建一個聲明, 指定為undefined
  foo = 2    // 此步驟時, 沒有要綁定的
})();
```

上面所說的是執行上下文的三大部份中的VariableEnvironment, 而作用域是屬於LexicalEnvironment, 可以參考[標準中的說明](#)。

再補充一點是, 以V8引擎的原始碼來看(剛那篇文章有提到), JS代碼在執行前的確是有經過編譯過程的, 然後最後才在電腦中執行。只是通常以它對開發的執行過程而言, 會稱它是"直譯"執行, 而不是"編譯為可執行檔"的那種執行。

最後, 會被提升的目前有下面這些, 不過仍有一些細節上的不同:

```
var, let, const, function, function*, class
```

有些人會認為 let 與 const 不會被提升, 實際上是會的。因為它們被定義有一段時間是無法存取的, 這是在被宣告與進入作用域之間時, 這段時間稱為 Temporal Dead Zone (TDZ, 時間死區), 所以不同於 var 或 function, 存取 let 或 const 提升的變數/常數會產生錯誤 ReferenceError, 而不是 undefined。

註: dead zone在英文裡是專有名詞, 是指"電波達不到的地區"。

要理解 `let` , `const` 是否會被提升，可以用下面的簡單例子來看。第一個例子，是正常可以輸出 `x` 變數的值：

```
let x = 'outer scope'

;(function() {
  console.log(x)
})();
```

第二個例子，會產生錯誤 `ReferenceError`。這是因為函數中的那個用 `let` 宣告的 `x` 變數被提升到函數中區塊的最上面，因此造成錯誤：

```
let x = 'outer scope'

;(function() {
  console.log(x)
  let x = 'inner scope' //多加這行程式碼，這行被提升到函式區塊中的最上面一行
})();
```

有興趣可以再參考[這裡的問答](#)，與上面這個範例的[出處文章](#)。

註：上面例子中的 IIFE 語句，前面加上分號(;)是一種保護的語法，可以看我寫的這篇[部落格文章](#)裡有提到。

撰寫風格建議

根據 Airbnb 的 [JavaScript 風格指引](#)，以及最近改版的 [Google JavaScript 風格指引](#)，以下總結幾點對於 `let` 或 `const` 在使用上，幾個建議的採用的撰寫風格：

1. 不要再用 `var` 來宣告變數，改用 `let` 與 `const`，而且優先使用 `const`，除非需要再指定值才用 `let`。(Google 5.1.1, Airbnb 2.1/2.2)
2. 不要使用逗號(,)在同一行來定義(宣告)多個變數或常數，例如 `let a = 1, b = 2` 是不必要的，應該是一行一個定義(宣告)。(Google 5.1.2, Airbnb 13.2)
3. 並不是在區塊中或函式中區域的最上面來宣告變數/常數，而是在合理的位置，在變數/常數首次被使用時的上面一行來宣告變數。(Google 5.1.3, Airbnb 13.4)

第1點的 `var` 的部份不要再用的理由，上面的內文已經有說明。`const` 可以用在物件、陣列與函式上，常數一宣告時就要指定值，犯錯的機會會減少很多。另外，JS引擎也可以作最佳化。所以大概9成的情況都是用 `const`，只有像 `for` 迴圈語句或一些需要再指定值的情況才會用到 `let`。

第2點的理由主要是為了除錯，除錯器可以一步步明確地停在每個變數/常數宣告的那一行。另一個理由是不用逗號(,)後，可以減少寫錯符號的機會，程式碼的可閱讀性也可以提升。說實在不需要偷懶省那幾行，當然有些開發者是因為習慣這樣作了，這就看你自己吧。

第3個的理由是因為現在的編譯器與JS引擎都已經作得很好，而且 `let` 與 `const` 都是區塊作用域，不用再擔心常數/變數會曝露到全域中的問題，提升特性如果你已經學過知道了，不要亂用也很難遇到。總之變數/常數在要用到前再宣告就行了，這是所謂的"合理的位置"。這會與 Douglas Crockford 大師所提的，一定要宣告在函式或程式檔案的最上面，這個撰寫風格的說法會不太一樣，不過當年也只有 `var` 可用，大師的想法自然有他的道理，但現在有了 `let` 與 `const` 就不用這樣作了。

結論

本文章對 `let` 與 `const` 作了一個完整的說明。其中有部份的章節內容是我之前在討論區中回答問題時，整理出來的部份。這兩個新的ES6特性，最近經常被問到，回答時查了不少資料，應該對你有幫助，雖然有些內容可能比較艱深些，我想多看幾次就應該能理解。說實在真的要解釋是什麼為什麼，不稍微說明一下底層的實作設計，也很難回答的清楚。
