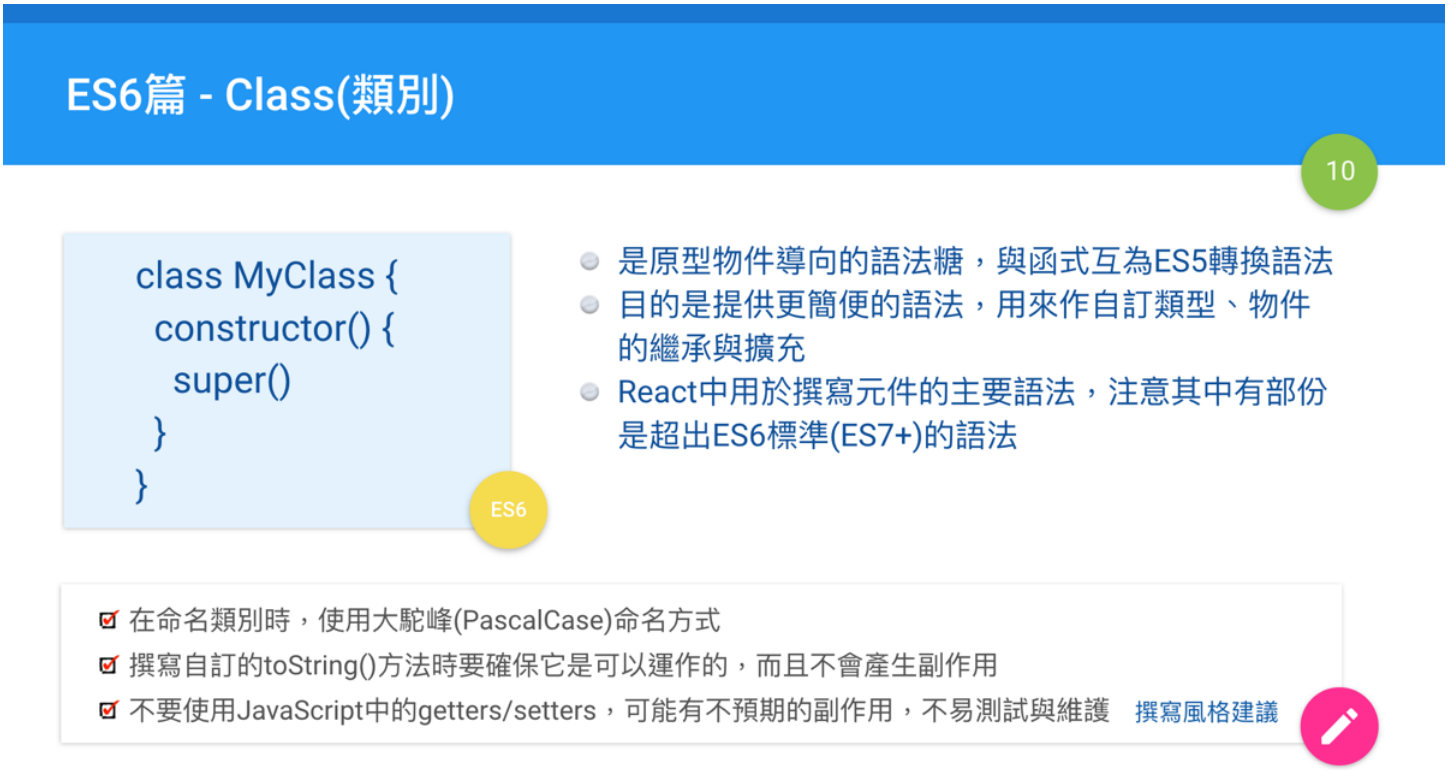
ES6篇 - Class(類別)



本章的目標是提供ES6中的Class(類別)一些廣泛的討論與使用建議。Class(類別)是一個全新的語法，但它的目前整體的內容，如果與Java、C++這些語言相比，還顯得粗糙與簡單，目前也沒什麼發展出什麼設計模式。不過，目前在React裡已經開始採用這種語法，作為元件撰寫時的必用語法。

> 註: 本文章同步放置於Github庫的這裡。

在ES6中的Class(類別)語法，並不是真的是以類別為基礎(class-based)的物件導向，在骨子裡仍然是以原型為基礎(prototype-based)的物件導向，它只是個語法糖(syntactical sugar)。加入Class(類別)語法的目的，並不是要建立另一套物件導向的繼承模型，而是為了提供更簡潔的語法來作物件建立與繼承，當然，一部份的原因是，讓已經熟悉以類別為基礎的物件導向程式語言的開發者使用，以此提供另一種在物件導向語法上的選擇。

在ES6之前

對於已經熟悉JS的開發者而言，在JS中原本就有設計一個四不像的物件導向語法，也就是以建構函式來作為類別，然後用new運算符來實體化物件的這種語法。以下是個簡單的例子:

```
function Player(fullName, age) {
  this.fullName = fullName
  this.age = age
}

Player.prototype.sayHi = function() {
  console.log('Hi! ' + this.fullName)
}

const inori = new Player('Inori', 16)
inori.sayHi()
```

你也可以把物件的方法寫在Player函式之中，只是每個物件在實體化時都會複製一份。為了節省資源的花費，所以物件的方法因為通常會在 `prototype` (原型鏈)上定義，以此達到所有物件共享這些方法的目的，它也有一些特殊的作用，例如物件實體化後，然後再定義原型鏈上的方法，這樣也可以讓已經實體化的物件使用這些方法。例如在上面的程式碼後再加上:

```
// 這段程式碼是在inori已經實體化之後加上的
Player.prototype.myAge = function(){
  console.log('My age is ' + this.age)
}

// 可以再使用
inore.myAge()
```

這方式對初學者來說是相當的怪異，大概是幾個項目:

- 容易與一般函式混用時造成混亂: 作為建構函式(constructor function)的函式通常沒有回傳值，裡面有用this。
- 如果忘了加 `new` 運算符: 像 `const gi = Player('Gi', 15)` 這樣的語句，不會回報錯誤， `gi` 也是個undefined，也不是物件。
- 雖然它像個類別的語法: 這個物件中的成員"封裝"，相當於沒有封裝，所有的屬性與方法都是公開的(public)，這個函式也沒繼承這種功能。

在漫長的10幾20年歲月中，老實說這種建構函式的語法並沒有很流行，函式有很多其他的功用，物件也可以用物件字面定義的方式，原型物件導向有其他的方式可以來對物件作方法或屬性的擴充。並不太需要以類別為基礎的物件導向的語法，或是用它來發展許多設計模式之類的。

Douglas Crockford大師很早就提出來的模擬類別繼承的研究 - Classical Inheritance in JavaScript，以及模擬私有與保護成員的一些樣式 - Private Members in JavaScript。是具有研究的價值，但在實作上不會這樣作。就算今天有了ES6新的類別語法，JavaScript中的以原型為基礎的物件導向，仍然是它設計的基石。對開發者來說仍然是必學，而且一定要學會。

而以原型為基礎的物件導向，目前容易看得到的有在使用的語言，就只有JavaScript一種，要學它會較不容易入門，但用起來很簡單。不容易入門是因為它的概念與以類別為基礎的物件導向概念不同，很容易就會想用之前學過的概念來套用或思考。實際上它在使用上有一些好處，反而更有彈性，這需要等待你去發掘。

小結一下，這邊所提供的意見，如果你已經有其他以類別為基礎的程式語言的基礎，這一整篇類別中差不多就是目前JS語言中所有的內容了。你可能會對這個新語法抱有太高的期待，但它仍然很初期內容也少，很多你覺得它應該作到的事都作不到，因此覺得JS設計得很差，這個想法是不對的。另一種想法是想說，原來這麼簡單，我學這些花個幾小時學一學就很厲害了，這也不對。

真正的JavaScript語言的物件導向設計，是以"原型(Prototype)"為基礎的物件導向，這10幾20年來都沒變過，目前許多樣式、函式庫、框架都是從這個基礎發展出來的，這是我一直強調的。而有一些新式的函式庫或框架例如React，它裡面會設計可以使用這個類別語法，純粹只是為了要簡化一些其中的語法樣式，或是提供不同的開發者不同的選擇，吸引已經有以類別為基礎的物件導向知識的開發者，不見得就會是用到物件導向的什麼設計模式。

> 註: 有興趣可以參考我的電子書中的這章: 原型基礎物件導向。

類別(Class)介紹

類別(Class)是先裡面定義好物件的整體結構藍圖(blue print)，然後再用這個類別定義，以此來產生相同結構的多個的物件實例，類別在定義時並不會直接產生出物件，要經過實體化的過程(`new` 運算符)，才會產生真正的物件實體。另外，目前因為類別定義方式還是個很新的語法，在實作時除了比較新的函式庫或框架，才會開始用它來撰寫。以下的為一個簡單範例:

```
class Player {
    constructor(fullName, age, gender, hairColor) {
        this.fullName = fullName
        this.age = age
        this.gender = gender
        this.hairColor = hairColor
    }

    toString() {
        return 'Name: '+this.fullName+', Age:'+this.age
    }
}

const inori = new Player('Inori', 16, 'girl', 'pink')
console.log(inori.toString())
console.log(inori.fullName)

const tsugumi = new Player('Tsugumi', 14, 'girl', 'purple')
console.log(tsugumi.toString())
```

> 註: 注意類別名稱命名時要使用大駝峰(ClassName)的寫法

> 註: 類別目前在ES6標準中與函式(Functions)屬同一章節。

下面分別說明一些這個例子中用到的語法與關鍵字的重要概念，以及類別延伸的一些語法。

this

`this` 簡單的說來，是物件實體專屬的指向變數， `this` 指向的就是"這個物件實體"，以上面的例子來說，也就是當物件真正實體化時， `this` 變數會指向這個物件實體。 `this` 是怎麼知道要指到哪一個物件實體？是因為 `new` 運算符造成的結果。

`this` 變數是JavaScript的一個特性，當函式呼叫或物件實體化(用new運算符)時，都會以這個 `this` 變數的指向對象，作為執行期間的依據。我們在函式中，使用作用範圍(Scope)來說明以函式為基礎的檢視角度，在函式區塊中可見的變數與函式的領域的概念。而JavaScript中，另外也有一種上下文環境(Context)的概念，就是對於 `this` 的在執行期間所依據的影響，即是以物件為基礎的的檢視角度。

`this` 也就是執行上下文可以簡單用三個情況來區分:

1. 函式呼叫: 在一般情況下的函式呼叫， `this` 通常都指向window(或全域)物件。這也是預設情況。
2. 建構式(constructor)呼叫: 透過 `new` 運算符建立物件實體，等於呼叫類型的建構式， `this` 會指向新建立的物件實例
3. 物件對其中的方法呼叫: `this` 指向呼叫這個方法的物件實體

所以當建構式呼叫時，也就是使用 `new` 運算符建立物件時， `this` 會指向新建立的物件，也就是下面這段程式碼:

```
const inori = new Player('Inori', 16, 'girl', 'pink')
```

因此在建構式中的指定值的語句，裡面的 `this` 值就會指向是這個新建立的物件，也就是 `inori`：

```
constructor(fullName, age, gender, hairColor) {
        this.fullName = fullName
        this.age = age
        this.gender = gender
        this.hairColor = hairColor
    }
```

也就是說在建立物件後，經建構式的執行語句，這個 `inori` 物件中的屬性值就會被指定完成，所以可以用像下面的語法來存取屬性:

```
inori.fullName
inori.age
```

第3種情況是呼叫物件中的方法，也就是像下面的程式碼中， `this` 會指向這個呼叫toString方法的物件，也就是 `inori`：

```
inori.toString()
```

對於 `this` 的說明大致上就是這樣而已，這裡都是很直覺的說明。 `this` 還有一部份的細節與應用情況， `this` 的概念在JavaScript中十分重要，初學者真的需要多花點時間才能真正搞懂。

建構式(constructor)

建構式是特別的物件方法，它必會在物件建立時被呼叫一次，通常用於建構新物件中的屬性，以及呼叫上層父母類別(如果有繼承的話)之用。用類別(class)的定義時，物件的屬性都只能在建構式中定義，這與用物件字面的定義方式不同，這一點是要特別注意的。如果物件在初始化時不需要任何語句，那麼就不要寫出這個建構式，實際上類別自己有預設的建構式，它會自動幫你作建構的工作。

關於建構式或物件方法的多形(polymorphism)或覆蓋(Overriding)，在JavaScript中**沒有**這種特性。建構式是會被限制只能有一個，而在原本在物件中的方法也沒這個特性，在物件中定義同識別名稱的方法只會有一個定義被使用，這與傳入參數有或沒有，或是有幾個無關。

所以如果你需要定義不同的建構式在物件中，因應不同的物件實體的情況，只能用函式的不定傳入參數方式，或是加上傳入參數的預設值來想辦法改寫，請參考函式其它內容中的說明。以下為一個範例:

```
class Option {
    constructor(key, value, autoLoad = false) {
```

```
        if (typeof key != 'undefined') {
            this[key] = value
        }
        this.autoLoad = autoLoad
    }
}

const op1 = new Option('color', 'red')
const op2 = new Option('color', 'blue', true)
```

> 註: 此處講的"建構式(constructor)"，與上一節中的"建構函式(constructor function)"是不同的東西，要特別注意。

私有成員

JavaScript截至ES6標準為止，在類別中並沒有像其他程式語言中的私有的(private)、保護的(protected)、公開的(public)這種成員存取控制的修飾關鍵字詞，基本上所有的類別中的成員都是公開的。雖然也有其他"模擬"出私有成員的方式，不過它們都是複雜的語法，這裡就不說明了。

目前比較簡單常見的區分方式，就是在私有成員(或方法)的名稱前面，加上下底線符號(_)前綴字，用於區分這是私有的(private)成員，這只是由程式開發者撰寫上的區分差別，與語言本身特性無關，對JavaScript來說，成員名稱前有沒有有下底線符號(_)的，都是視為一樣的變數。以下為簡單範例:

```
class Student {
    constructor(id, firstName, lastName) {
        this._id = id
        this._firstName = firstName
        this._lastName = lastName
    }

    toString() {
        return 'id is '+this._id+' his/her name is '+this.firstName+' '+this.lastName
    }
}
```

有一種樣式是利用建構式是私有的特性(有自己的作用域)，把私有屬性寫在建構式裡面，getter與setter也寫在裡面，這種樣式的確可以達到私有成員的目的。但是，寫在建構式裡面的getter或setter，在每次物件建立實體時，就會複製一份新的，這會花費更多的資源與效能。而且這也不太像是以類別為基礎的物件導向寫法。如下面的例子:

```
class Student {
    constructor(id, firstName, lastName) {
      let age = 20 // 這是私有的

      this.name = firstName + ' ' + lastName // 這是公開的

      this.greet = function () {
         // 這裡可以存取得到name與age
         console.log(`name: ${this.name}, age: ${age}`);
      }
    }

    toString() {
        return 'id is '+this._id+' his/her name is '+this.firstName+' '+this.lastName
    }
}

const aStudent = new Student(123, 'Eddy', 'Chang')
console.log(aStudent.age)
aStudent.greet()
```

> 註: 如果是私有成員，就不能直接在外部存取，要用getter與setter來實作取得與修改值的方法。私有方法也不能在外部呼叫，只能在類別內部使用。

Getter與Setter

在類別定義中可以使用 `get` 與 `set` 關鍵字，作為類別方法的修飾字，可以代表getter(取得方法)與setter(設定方法)。一般的公開的原始資料類型的屬性值(字串、數字等等)，不需要這兩種方法，原本就可以直接取得或設定。只有私有屬性或特殊值，才需要用這兩種方法來作取得或設定。getter(取得方法)與setter(設定方法)的呼叫語法，長得像一般的存取物件成員的語法，都是用句號(.)呼叫，而且setter(設定方法)是用指定值的語法，不是傳入參數的那種語法。以下為範例:

```
class Option {
    constructor(key, value, autoLoad = false) {
        if (typeof key != 'undefined') {
            this['_' + key] = value;
        }
        this.autoLoad = autoLoad;
    }

    get color() {
      if (this._color !== undefined) {
        return this._color
      } else {
        return 'no color prop'
      }
    }

    set color(value) {
      this._color = value
    }
}

const op1 = new Option('color', 'red')
op1.color = 'yellow'

const op2 = new Option('action', 'run')
op2.color = 'yellow'
```

> 註: 所以getter不會有傳入參數，setter只會有一個傳入參數。

靜態成員

靜態(Static)成員指的是屬於類別的屬性或方法，也就是不論是哪一個被實體化的物件，都共享這個方法或屬性。而且，實際上靜態(Static)成員根本不需要實體化的物件來呼叫或存取，直接用類別就可以呼叫或存取。ES6中的類別目前只有靜態方法，沒有靜態屬性，使用的是 `static` 作為方法的修飾字詞。以下為一個範例:

```
class Student {
    constructor(id, firstName, lastName) {
        this.id = id
        this.firstName = firstName
        this.lastName = lastName

        //這裡呼叫靜態方法，每次建構出一個學生實體就執行一次
        Student._countStudent()
    }

    //靜態方法的定義
    static _countStudent(){
      if(this._numOfStudents === undefined) {
          this._numOfStudents = 1
      } else {
          this._numOfStudents++
      }
    }

    //用getter與靜態方法取出目前的學生數量
    static get numOfStudents(){
      return this._numOfStudents
    }

}

const aStudent = new Student(11, 'Eddy', 'Chang')
console.log(Student.numOfStudents)
```

```
const bStudent = new Student(22, 'Ed', 'Lu')
console.log(Student.numOfStudents)
```

靜態屬性目前來說有兩種解決方案，一種是使用ES7+的Class Properties(Class Fields & Static Properties)，可以使用 `static` 關鍵字來定義靜態屬性，另一種是定義到類別原本的定義外面，這個語法在React中已經有看到在使用了，主要是用在定義 `defaultProps` 與 `propTypes` 使用的，下面為一個簡單的例子:

```
// ES7語法方式
class Video extends React.Component {
  static defaultProps = {
    autoPlay: false,
    maxLoops: 10,
  }
  render() { ... }
}


// ES6語法方式
class Video extends React.Component {
  constructor(props) { ... }
  render() { ... }
}

Video.defaultProps = { ... }
```

> 註: ES7+的靜態(或類別)屬性的轉換，要使用bebal的babel-plugin-transform-class-properties外掛。

繼承

用extends關鍵字可以作類別的繼承，而在建構式中會多呼叫一個 `super()` 方法，用於執行上層父母類別的建構式之用。 `super` 也可以用於指向上層父母類別，呼叫其中的方法或存取屬性。

繼承時還有有幾個注意的事項:

- 繼承的子類別中的建構式， `super()` 需要放在建構式第一行，這是標準的呼叫方式。如果有需要傳入參數可以傳入。
- 繼承的子類別中的屬性與方法，都會覆蓋掉原有的在父母類別中的同名稱屬性或方法，要區為不同的屬性或方法要用 `super` 關鍵字來存取父母類別中的屬性或方法，例如 `super.toString()`

```
class Point {
    constructor(x, y) {
        this.x = x
        this.y = y
    }
    toString() {
        return '(' + this.x + ', ' + this.y + ')';
    }
}

class ColorPoint extends Point {
    constructor(x, y, color) {
        super(x, y)
        this.color = color
    }
    toString() {
        return super.toString() + ' in ' + this.color
    }
}
```

類別中的箭頭函式

在類別中使用箭頭函式來取代裡面原有方法的定義，這在React的元件撰寫時，也是很常見的一種語法。這個語法是有其目的的，主要是為了要作this的綁定。例如以下的例子，範例來自這裡:

```
class PostInfo extends React.Component {
  handleOptionsButtonClick = (e) => {
```

```
    this.setState({showOptionsModal: true});
  }
}
```

因為在React元件使用ES6 Class的語法來定義元件時，官方取消了原有的Autobind(自動綁定)功能，所以在現在新式的元件寫法中，開發者必須自行綁定類別中的方法。有兩種方法可以進行綁定，第二種像上面的例子這樣，第一種是在建構式中使用函式的bind方法，有些時候會使用第一種語法。像下面這個例子:

```
class PostInfo extends React.Component {
  constructor(props) {
    super(props);
    // 手動綁定的語法，綁定this到元件的實體...
    this.handleOptionsButtonClick = this.handleOptionsButtonClick.bind(this);
  }
  handleOptionsButtonClick(e) {
    // ...確保'this'可以參照到元件的實體
    this.setState({showOptionsModal: true});
  }
}
```

這個語法也不是ES6標準的語法，它是正在訂定中的新標準語法，是屬於Class Properties(Class Fields & Static Properties)。不過babel編譯工具透過外掛，可以正確編譯就是。

> 註: ES7+的靜態(或類別)屬性的轉換，要使用bebal的babel-plugin-transform-class-properties外掛。

> 註: 你可能會好奇，為何babel編譯工具都可以正確支持編譯React中的JSX與一些超出ES6標準的語法？因為這babel工具專案實際上是Facebook贊助(出錢養的)的專案。

風格指引

- 在命名類別或建構式時，使用大駝峰(PascalCase)命名方式。(Airbnb 22.3)
- 撰寫自訂的toString()方法是很好的，但要確定它是可以運作，而且不會有副作用的。(Airbnb 9.4)
- 如果 屬性/方法 是布林值，使用像isVal()或hasVal()的命名。(Airbnb 23.3)
- 不要使用JS的getters/setters，因為它們會造成不預期的副作用，而且難以測試與維護，使用像getVal()與setVal('hello')的方法。(Airbnb 23.2)

結論

本章很簡單的說明在ES6中的新特性Class(類別)的用法，相當的簡單而且內容很少。Class(類別)目前有很多新的語法正在新的標準中訂定中，但是可以看到像React之類的新式函式庫，都已經等不及正在使用中，這是一個很特別的情況。不過新訂定的標準有可能還會更動，而且章節名稱會改來改去，這一點是要注意的。實際上，React採用類別語法作為元件撰寫的語法，也不過是去年底到今年的事，時隔才短短一年而已。

類別的語法目前雖然內容少而簡單，但它仍然有不少的好處。如果你把它當作是一種自訂物件類型的簡寫語法來看，它隱藏了很多JS中物件在作初始化與繼承的複雜語法，提供了較為簡單、閱讀性高、較容易維護的語法。並不是說直接使用原型語法來寫達不到，只是原型物件導向的特性需要理解到一定的程度，才有辦法寫出像類別這麼簡短的語法能達到的事情。