日期與時間

Date在JavaScript中是一個內建的特殊物件，Date(日期)的資料型態應該是指Datetime(日期時間)而言，所以不只包含所謂的日期 - 年月日而已，它也包含時間中的時、分、秒與微秒。

`Date` 物件複雜的地方不只是它的進位並非十進位而已，年月日的進位還涉及閏年、星期幾的問題，以及全球各種不同語言、全球時區使用的本地化格式的問題等等各種問題。所以 `Date` 物件有時是個很難處理的東西，在複雜的應用情況時，以及需要較好的瀏覽器相容性時，我們會用其他的工具函式庫來協助，例如一套很常使用的函式庫 - Moment.js，不過它並不在本章的範圍之中，有需要可以再參考使用。

`Date` 物件中包含了許多對日期時間處理方法，大致上可以分成以下幾類:

- getter: 獲取某種時間格式，例如getFullYear是回傳西元年
- setter: 設定某種時間格式，例如setFullYear設定西元年
- 格式化的getter: 通常是要獲取不同地區的日期(或時間)格式，以及轉換成各種資料格式

> 註: 微秒(Millisecond, ms)是千分之一秒

`Date` 物件的建構式

`Date` 物件一定只能使用 `new` 運算符來建立，這是JavaScript中的硬規則，有一些特殊的內建物件一定要使用建構式進行物件的實體化。

`Date` 物件有四種可在建構式傳入參數的類型，如以下的語法說明，其中只有第三種是可以傳入字串資料類型，傳入參數 `dateString` 指的是遵守通用的日期時間字串、國際標準RFC2822或ISO 8601這幾種的字串格式，後面會再說明其中的內容。

```
const date = new Date()
const date = new Date(milliseconds)
const date = new Date(dateString) //只有這種方式是傳入字串類型，其他都不是。
const date = new Date(year, month, day, hours, minutes, seconds, milliseconds)
```

以下就每種建立 `Date` 物件的應用情況分別說明，從範例中來理解它的用法。

取得目前日期時間

`new Date()` 不加任何參數，就會使用瀏覽器環境來取得的目前日期時間作為傳入參數，說白一點也就是獲得當下的日期時間，輸出時會使用國際日期標準的格式輸出，相等於呼叫 `toString()` 方法，注意它仍然是個物件類型，只是無法直接輸出裡面資料結構而已:

```
const datetime = new Date()
console.log(datetime) //Tue Jul 12 2016 11:35:48 GMT+0800 (CST)
console.log(datetime.toString()) //Tue Jul 12 2016 11:35:48 GMT+0800 (CST)
console.log(datetime.toUTCString()) //Tue, 12 Jul 2016 03:35:48 GMT
console.log(datetime.toTimeString()) //11:35:48 GMT+0800 (CST)
```

另一種要用於計算時間用的格式，則是把 `Date` 物件轉成只用微秒數值的格式，它是一個由 `1 January 1970 00:00:00 UTC` 開始計算，到目前日期時間的微秒數字值。這個時間類似稱之為UNIX時間，原本是UNIX系統用來表示時間的方式，不過UNIX時間只計算到秒，而這個數字值是計算到微秒，現在很常用於在程式中計算時間之用。有兩種方式可以獲得這個數字值:

```
//很常用，正號(+)是強制轉換為數字的語法
const timeInMs1 = +new Date()

//因為是靜態方法，不常見
const timeInMs2 = Date.now()
```

> 註: 使用 `Date` 物件來評估或測量JavaScript應用程式(或語句)的執行時間並不夠可靠與精確，有這個需求的話，你應該使用performance.now或benchmark.js函式庫。

取得日期時間其中某個值

在得到 `Date` 物件實體後，可以對 `Date` 物件取得包含在其中的某個值，例如年、月、日、星期、時分秒等等。

我們先來看年月日與星期怎麼取得的範例:

```
const dateObject = new Date() //Fri Jul 15 2016 16:23:49 GMT+0800 (CST)

const date = dateObject.getDate() //15
const day = dateObject.getDay()  //5
const month = dateObject.getMonth()  //6
const year = dateObject.getFullYear()  //2016
```

`Date` 物件的取值方法，說實在如果直接從名稱上理解會有些不清楚，先看比較沒問題的部份:

- getDate: 取得日期的值，範圍為1-31
- getFullYear: 取得年(西元年)，為4位數數字

那麼不清楚或一眼就看得出來的部份是下面兩個:

- getDay: 取得星期幾的值，範例為0-6
- getgetMonth: 取得月份的值，範例為0-11

這兩個方法的回傳值，實際上是陣列索引值，也就是說如果你要轉成真正的月份與星期的值，需要有一個對照的陣列，然後用這個回傳值當作陣列索引值去轉換出來。仔細想想，會這樣設計也是合理的，因為這兩個值在不同世界地區或語言中，都有不同的代表字詞(例如台灣的"星期"，又可以稱為"週"或"禮拜"，台語也可簡稱"拜")，所以乾脆保留給設計師自行處理。以下是月份的範例:

```
const monthNamesEn = [
  'January', 'February', 'March', 'April', 'May', 'June',
  'July', 'August', 'September', 'October', 'November', 'December'
]

const monthNamesZh = [
  '一月', '二月', '三月', '四月', '五月', '六月',
  '七月', '八月', '九月', '十月', '十一月', '十二月'
]

console.log(monthNamesEn[month]) //July
console.log(monthNamesZh[month]) //七月
```

以下則是星期的範例，注意陣列的第一個(索引值為0)是指"星期日"，有些人真的不知道"星期日"才是一週的第一天:

```
const dayNamesEn = [ 'Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday']
const dayNamesZh = [ '星期日', '星期一', '星期二', '星期三', '星期四', '星期五', '星期六']

console.log(dayNamesEn[day]) //Friday
console.log(dayNamesZh[day]) //星期五
```

時、分、秒與微秒的取得就很標準一致，以下為範例，後面註解說明它的回傳數字的範圍:

```
const hour = dateObject.getHours() //0-24
const minute = dateObject.getMinutes() //0-59
const second = dateObject.getSeconds() //0-59
const ms = dateObject.getMilliseconds() //0-999
```

> 註: 仔細比對一下上面範例，取得年月日星期的方法為單數英文字詞，取得時分秒微秒的是複數英文字詞。

另外， `Date` 物件中還有數個名稱中帶有UTC的取得方法，這些都是在獲取UTC(標準時間)的值。至於取得時區的方法是 `getTimezoneOffset` ，它是用分鐘計算的值，而且是以目前程式執行的時區為基準，所以例如以下的範例會回傳 `-480` ，實際上是 `UTC+8` 的時區:

```
const timezoneByMins = dateObject.getTimezoneOffset() //-480

//除以60後正負相轉才是時區
const timezone = -(dateObject.getTimezoneOffset()/60)
```

> 註: 時區處理也有可能很複雜，如果有需要可以用專門處理時區的工具函式庫來協助，例如Moment Timezone。

設定日期時間 與 日期時間字串

在 `Date` 物件實體化時，給定傳入參數值可以直接設定這個 `Date` 物件內的內容，除了不給定內容的建構式會自動抓取瀏覽器目前的日期時間。其它兩種建構式就不多作說明了，本節的重點在於日期時間字串，

JavaScript中的日期格式字串是個頭痛的問題，目前雖然已經是標準的一員。但這標準可不是一開始就有的，而是在ES5標準後才訂定的，所以有些舊版的瀏覽器中並不一定可以這樣用，也有可能在這個瀏覽器可以這樣定義，到別的瀏覽器就不能用了。這裡有一份參考的時間日期字串格式相容表。

通用的格式

> 建議使用

這並不算什麼標準，只是用這樣定義的話，幾乎在每種瀏覽器品牌或版本都可以相容使用。除非你有非不得已要用其他格式的理由，通常都建議只用這種定義格式就好了，而且最好是使用固定位數的數字，實作上會方便很多:

```
2009/07/12
2009/7/12
2009/07/12 12:34
2009/07/12 12:34:56
```

把年放在最後面也是可以，不過這方式是英文語言使用者常用的方式，請不要搞混月份與日期的位置，第一個位置是"月份":

```
07/02/2012
7/2/2012
7/2/2012 12:34
```

如果你想要直接在字串中加上微秒的定義，會出現嚴重的不相容性，大概只有Chrome瀏覽器認得。所以解決方式就拆開所有的值，改用使用第四種的 `Date` 物件建構式分別傳入每個值，或是再額外用設定值的方法 `setMilliseconds` 直接設定才行。

> 註: 月、日的表示如果要用兩位數字就都用兩位(例如01/01與11/04)，如果要用1到2位的數字就統一這樣用(例如1/1或11/4)。如果你要混用的話，不見得一定可以用，瀏覽器的相容性可能會有問題。

> 註: 在時間上的字串值，時與分至少都要有，而且都用兩位數字，例如01:00或12:09。

ISO 8601格式

> 注意: 舊版瀏覽器不相容

EMCAScript標準中定義的格式，實際上它就是ISO 8601標準，格式如下:

```
YYYY-MM-DDTHH:mm:ss.sssZ
```

YMD代表年月日，Hms代表是時分秒，這兩部份比較沒什麼太大的問題。

那麼 `T` 代表什麼？ `T` 只是分隔年月日與時分秒而已。

那麼 `Z` 代表什麼？ `Z` 是時區的意思，單純用Z代表是UTC標準時間，如果是其他時區的時間可以用 `+` 或 `-` ，再加上 `HH:mm` 作為時區的表達式，所以像下面的幾個日期時間字串是合於這個規定的:

```
2016-01-01 //年月日可以只有年、年月
2016-07-15T09:00  //時間的部份至少要有時與分
2016-07-15T09:26:23.216Z
2016-07-15T09:20:37.788+08:00
```

RFC 2822格式

另一種格式則是RFC2822格式，它並不是一種專門用於定義日期時間格式的標準，這個標準所定義的標題名稱是"Internet Message Format"(網際網路訊息格式)，主要是用於定義Email(或RSS)的相關格式的標準，而其中有一章是關於日期時間格式的章節。所以，這種日期時間格式，主要的使用群是英文使用者，原因在它使用了英文中的月份與星期縮寫在字串，以下為格式範例:

```
ddd, DD MMM YYYY HH:mm:ss ZZ
MMM DD, YYYY
DD MMM, YYYY
```

以下是幾個範例:

```
Mon, 15 Aug 2005 15:52:01 +0000
Thu, 18 Feb 2016 15:33:10 +0200
Jan 1, 2015
1 Jan, 2015
```

由於月份與星期都已經字串化(不是數字)，所以你可能會看到有很多種位置不同的寫法，基本上瀏覽器處理這種字串並不會太困難。RFC2822標準並沒有明確指出格式的限制，程式語言都可以很容易實作與解析這種日期字串。例如在MSDN中的說明，把它歸類為其他格式之中，與其他的日期格式使用相同的規則來作解析處理。

> 注意: 這段的內容只是說明RFC2822與ISO 8601格式定義日期時間的用法，一般情況下你只需要使用第一種的定義方式即可。

> 註: 你可能不太明白為何在設定日期時間時，還要加上星期幾的意義何在。實際上這個格式通常是用來轉換到其他的格式之用。

> 注意: JavaScript有一個內建的Date.parse()方法，它是對照使用日期時間字串的 `Date` 物件實體化的靜態方法，但它的不相容性相當高，建議不要使用它。

設定方法

設定方法對照取得某個值的方法，除了把get變為set外，也少了星期的部份。要注意的是，除了上面說的日期時間字串(dateString)，其他的傳入參數一律只能用數字類型。

所有的set方法，都可以使用正負整數或0作為傳入值，例如以 `setMonth` 為例，雖然它的設定範例是0-11，例超過這個範圍後，會開始進行相加或相減，有可能會影響到年份增加或減少。傳入參數值的規則如下:

- 0: 對 `setDate` 方法是回傳上個月的最後一日。 `setMonth` 方法則是回傳一月(索引值為0)。
- 範圍內的整數: 直接設定為該數字
- 其他正整數: 日期相加
- 其他負整數: 日期相減

先看年月日的部份，這部份除了月的部份也是要用陣列的索引值外，還有一個 `setDate` 要特別注意:

```
const dateObject = new Date()

dateObject.setFullYear(2015) //年，4位數字
dateObject.setMonth(0) //月，陣列索引值 0-11
dateObject.setDate(24) //日，正負整數
```

時分秒與微秒的設定方法部份，都有類似 `setDate` 的規則，只要超過可設定的範例，就會開始進行相加或相減的情況，以下為簡單的範例:

```
dateObject.setHours(2)
dateObject.setMinutes(20)
dateObject.setSeconds(60)
dateObject.setMilliseconds(150)
```

最後是 `setTime` 方法，它是對照傳入值為微秒數值的 `Date` 物件實體化的建構式，傳入值是上面所說的由 `1 January 1970 00:00:00 UTC` 開始計算，到目前日期時間的微秒數字值。

此外，設定方法中也有多個帶有UTC字串的方法，這些都是專門用於設定UTC時間用的，用法與上述類似，就不再多說。

> 注意: 由於setYear與getYear兩個方法存在了Y2K(千禧蟲危機)的問題而被棄用了，目前已改用 setFullYear與getFullYear。

本地化與格式化

`Date` 物件中已有幾個可以輸出本地化格式字串的方法，主要有三個分別為 `toLocaleString` 、 `toLocaleDateString` 、 `toLocaleTimeString` ，它們會依照JavaScript程式所執行的環境(瀏覽器)進行本地化，一個簡單的範例如下:

```
const date = new Date()

console.log(date.toLocaleString())     //2016/9/14 下午4:33:41
console.log(date.toLocaleDateString())  //2016/9/14
console.log(date.toLocaleTimeString()) //下午4:33:41
```

格式化的需求通常會用於各種不同的日期或時間的顯示，配合上面所說的獲取日期或時間中某個值的說明，就可以進行各種日期或時間的格式化。 Date 物件並沒有提供可以進行格式化(format)的標準方法，所以這部份需要程式設計師自己撰寫或使用外部函式庫來輔助，一個簡單的格式化範例如下:

```
const now = new Date()
const months = ['Jan','Feb','Mar','Apr','May','Jun','Jul','Aug','Sep','Oct','Nov','Dec']
const formattedDate = now.getDate() + '-' + months[now.getMonth()] + '-' + now.getFullYear()

console.log(formattedDate)
```

> 註: toLocaleFormat() 方法是一個非標準的方法，大部份的瀏覽器品牌都不能使用。

日期比較

Date 物件可以直接比較大小，這種比較是比較日期時間的早晚順序，日期時間愈晚的 Date 物件會愈大，一個簡單的範例如下:

```
const myDate = new Date()
const today = new Date()

myDate.setFullYear(2015,8,9)

if ( myDate > today ){
  console.log('今天比2015.8.9日早')
} else {
  console.log('今天比2015.8.9日晚')
}
```

Date 物件也可以轉為自微秒數值的格式，它是一個由 1 January 1970 00:00:00 UTC 開始計算，到目前日期時間的微秒數字值。要取得這個數值是使用 getTime 方法，一個簡單的範例如下:

```
const myDate = new Date()
const today = new Date()

myDate.setFullYear(2015,8,9)

console.log(myDate.getTime())
console.log(today.getTime())
```

實例

時鐘

一個簡單的電子時鐘範例如下:

```
function startTime(){
  const today = new Date()
  const hour  = today.getHours()
  const min   = today.getMinutes()
  const sec   = today.getSeconds()

  // 轉成字串，如果低於10，前面加上'0'
  const hourString = ( hour < 10)? ('0'+ hour) : ('' + hour)
  const minString = ( min < 10)? ('0'+ min) : ('' + min)
  const secString = ( sec < 10)? ('0'+ sec) : ('' + sec)

  document.getElementById('content').innerHTML = hourString + ':' + minString + ':' + secString
```

```
}

//重覆每秒執行
setInterval(startTime ,1000)
```