事件處理

JavaScript是一個以事件驅動(Event-driven)的程式語言。事件驅動程式設計的主要流程，是由圖形化使用者操作介面(UI)的互動事件為主要核心，藉由事件的觸發動作(滑鼠點按、鍵盤輸入等等)或是感應器的訊息，來啟動整體的程式流程。

依據"異步程式設計與事件迴圈"一章的內容中的說明，在JavaScript中有一個不斷偵測事件發生的事件迴圈(Event Loop)，所有在網頁上的DOM元素註冊的事件，都會進入一個佇列(queue)中，等待被觸發事件，然後作出相對的回應送還給使用者。負責實作事件迴圈的是瀏覽器環境，佇列有可能不只一個，一般認為至少會依照W3C所定義的各種不同會發生佇列情況，也就是有5種佇列，包含事件、回調、使用外部資源等等。

我們在這裡所稱的事件(Event)，通常我們把它稱為瀏覽器事件(browser events)，這些是網頁中的DOM(Document Object Model)元素的事件，標準制定者是W3C組織，然後由ECMAScript負責支援的角色。也有一些特定的事件是由瀏覽器品牌自行制定，這不在我們討論的範圍之中。

近年來實現了伺服器端的JavaScript語言執行環境的Node.js，它並沒有這一系列直接可以使用的事件，內建的事件模型是Node.js自行設計的，不過它採用了相當類似的樣式，使用一個 `EventEmitter` 類別的物件實體來作事件處理，在底層也使用事件迴圈(Event Loop)的設計來達成。不過，你如果要進入伺服器端的事件處理領域，建議先從熟悉瀏覽器端的事件先著手，你會發覺它們有很多相似的設計與樣式。

DOM事件

DOM(Document Object Model, 文件物件模型)是由W3C組織所制定的跨平台與跨程式語言的應用程式介面，它是把HTML、XHTML、XML文件的視為樹狀的結構，每個節點都是代表文件的一個物件。DOM的標準大戰可以從第一次瀏覽器大戰，由Netscape與微軟IE開始算起，到2015年已經是第4級(Level 4)的標準。

DOM事件則是可以註冊各種事件處理器/監聽器(event handlers/listeners)在DOM的節點元素上，在DOM標準的第2級(Level 2)時，W3C標準化了DOM中的事件模型，統一不同瀏覽器中的事件模型差異。這個標準也就是我們現在在瀏覽器上使用的事件模型基準。

JavaScript程式語言一直以來對瀏覽器中的物件模型(object model)與其事件模型，有非常高的支援性，雖然DOM標準從來就不是只專門制定給JavaScript單一種語言使用的，不過因為JavaScript的使用群太高了，現在也差不多就是制定給它用的。

那麼，什麼樣的DOM元素中會對應什麼樣的DOM事件，或是什麼樣的操作方式會觸發怎麼樣的事件？

這都由DOM中的標準來制定，例如像這個DOM事件列表非常的長，從滑鼠、鍵盤輸入到HTML表單…等等，近年來由於觸碰式的行動裝置很流行，W3C也開始制定觸碰事件(Touch Events)的標準。不過，有幾個瀏覽器品牌自己實作了不少非標準的事件，有些事件是只用於該瀏覽器的特殊用途。

註: DOM events (維基百科)

Event物件(介面)

JavaScript中定義了Event物件，其中包含了事件通用的屬性與方法，例如事件的對象元素等等，事件的來源是來自DOM，所以這個物件通常稱之為Event介面。W3C制定了標準的Event介面規格。其中有幾個常用的屬性與方法分列如下。

Event物件屬性(以下屬性都是只能讀不能寫):

- currentTarget: 目前的事件對象
- target: 分派事件的原始對象
- type: 事件的類型，共有數十種
- bubbles: 冒泡狀態，布林值，true代表會在DOM中往上冒泡
- cancelable: 事件是否為可取消的，布林值

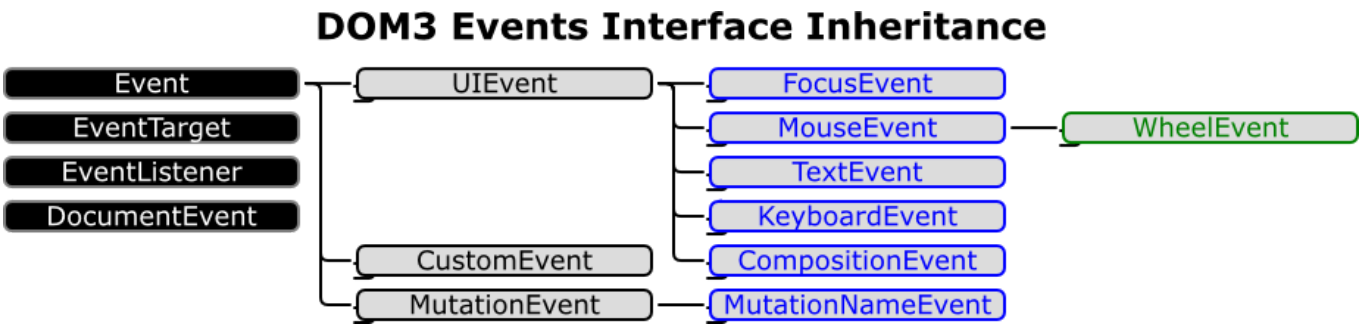
> 註: 目前的事件對象(currentTarget)與分派事件的原始對象(target)，在冒泡與捕捉階段，其中的值會不同。

Event物件方法:

- preventDefault(): 取消事件的行為(需為可取消的事件)，但無法阻止事件的傳播(propagation)
- stopPropagation(): 停止事件的傳播行為

> 註:事件的傳播行為在下面的章節有再加說明。

Event介面只是個基礎物件，從它擴充了使用於特定情況的事件，包含對特定事件的資訊，詳見以下的階層圖(出自這個網站):



依照事件階層圖中，UIEvent與CustomEvent(自訂事件)繼承自Event物件，從UIEvent中又擴充出各種不同的對應事件，例如針對滑鼠與鍵盤的，FocusEvent是設計給Focus(鎖定、聚焦)事件使用的，CompositionEvent是針對輸入文字事件使用的，這是因為輸入文字並不一定單純使用鍵盤(用輸入法輸入或用語音輸入等等)，它與鍵盤事件也可以相互輔助。而WheelEvent是有滾輪的設備使用的。

> 註: 許多外部函式庫例如jQuery，對於Event物件會以W3C的標準進行擴充。

EventTarget物件(介面)

EventTarget物件則是JavaScript所設計的一種當作介面的物件，它可以接收事件，以及讓監聽者註冊到上面。DOM元素、document、window物件，是最常見的EventTarget物件，另外也有其他的物件可作為EventTarget。EventTarget物件中有三個方法:

- `addEventListener`：在事件對象上加入事件監聽者
- `removeEventListener`：從事件對象移除事件監聽者
- `dispatchEvent`：送出事件給所有有訂閱的監聽者

另外需要提到的一點，W3C標準中對於EventListener也有定義它是一個介面，作為事件監聽者之用，不過JavaScript語言在所有的函式中都有實作這個介面，所以事件監聽者在呼叫 `handleEvent` (處理事件)方法時，相當於呼叫函式。EventListener(事件監聽者)或稱為事件處理函式，可以自動得到事件傳入參數值，以此可以存取得到事件的屬性與方法，例如以下的範例:

```
const me = document.getElementById('me')

me.addEventListener('click',
    function(e){
      console.log(e.currentTarget)
      console.log(e.target)
      console.log(e.type)
      console.log(e.bubbles)
      console.log(e.cancelable)
      e.stopPropagation()
    },
    false)
```

事件處理模型

現行的事件處理模型通常會使用"監聽"(listen)的方式，作為事件處理的標準樣式，因為DOM標準制定歷史版本不同，實際上有好幾種不同的方法可以作事件處理。以下分述這幾級的差異，其中第一種與第二種是舊的方式，不建議使用。

內聯模型

這種方式是最簡單的，也稱為內聯模型(Inline model)。它直接在HTML裡DOM元素中標記中使用，每個元素都會實作對應的可使用事件，名稱都會是像"onxxxxx"這樣的全小寫字詞，例如按鈕會實作onclick的事件屬性(attribute)，就在這裡面寫上事件處理的程式碼:

```
<button onclick="console.log('hello!');"> Say Hello! </button>
```

JavaScript引擎中會產生一個對應的匿名函式，包含在onclick中的語句。這個方式也是最不建議的方式，它完全不像個用JavaScript語言寫的應用程式，也因為需要直接寫在HTML中，完全沒有彈性可言。

傳統模型

傳統模型(Traditional model)方式，提供了分離HTML與JavaScript程式碼的語法，它比之前內聯模型的方式好得多了。不過它依然有個大問題，就是它只能在一個元素上使用一個事件，所以也不建議使用。

```
document.getElementById('myButton').onclick = function(){
    console.log('hello!')
}
```

DOM Level 2

這個方式又被稱為W3C方式(W3C Way)模型，這個方式才能說它是用事件監聽(Event Listen)的方式，使用callback(回調)函式，作為事件的監聽者(或稱之為事件處理函式)。

```
const el = document.getElementById('myButton')

el.addEventListener( 'click', function(){
     console.log('hello!')
}, false)
```

事件監聽的方式可以對一個元件附加多個事件處理函式，而且以標準來說基本有定義三種方法可使用:

- `addEventListener`：在事件對象上加入事件監聽者
- `removeEventListener`：從事件對象移除事件監聽者
- `dispatchEvent`：送出事件給所有有訂閱的監聽者

> 註: 微軟IE瀏覽器在舊版本中使用自己定義的事件處理方法，所以要與舊版本相容時需要特別注意。IE9之後就能使用上述的事件監聽方式。

事件觸發的順序

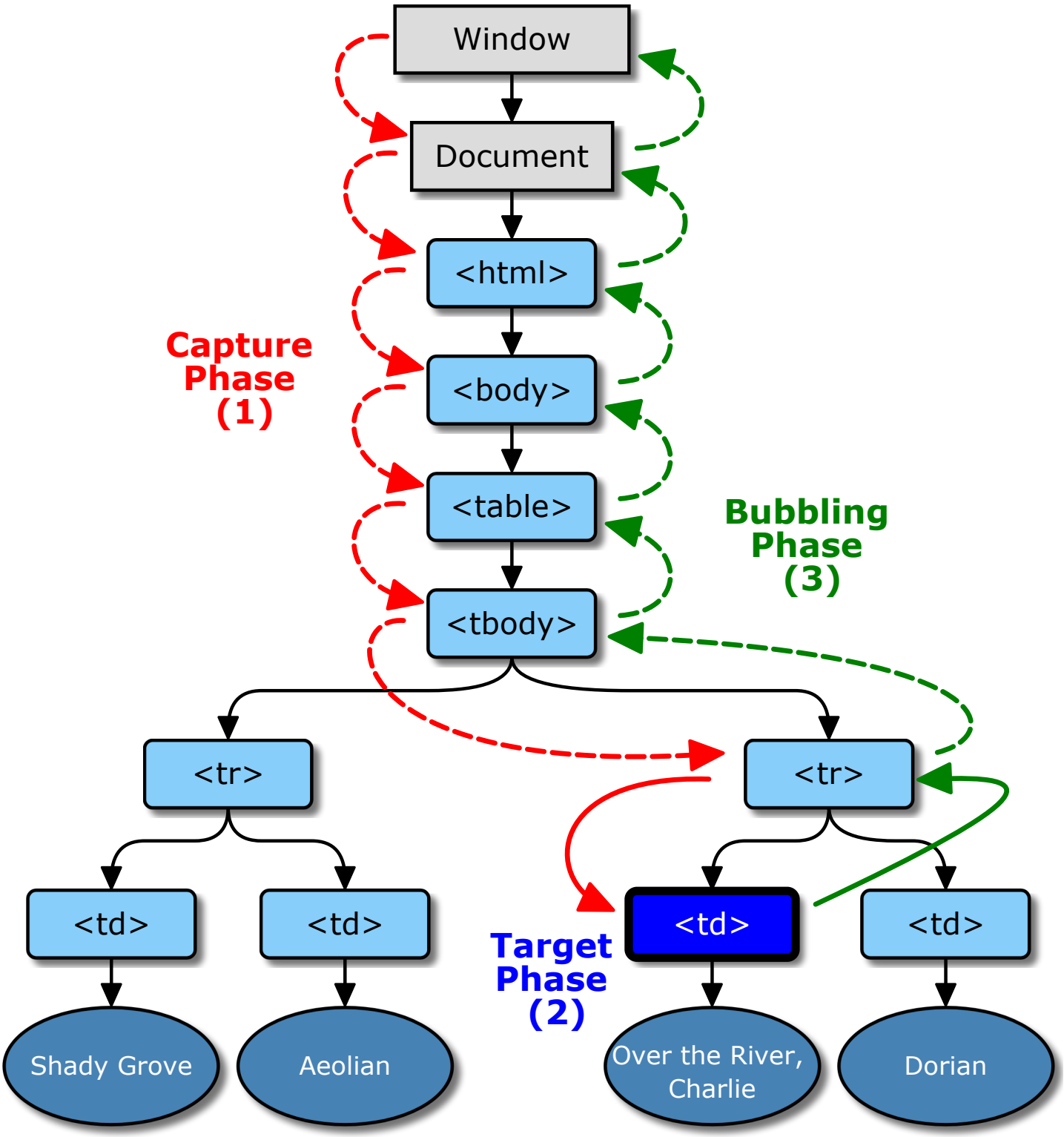
在同一個元素上註冊多個不同的事件監聽者(事件處理函式)，它的在事件觸發時的執行順序是怎麼樣子的？是按照程式碼所寫的順序嗎？

由於W3C的DOM第2代標準中並沒有明確的說明順序這件事，所以當如果有多個事件處理函式在同一元素中註冊時，它們的執行順序將會由瀏覽器來決定，有些舊的瀏覽器並不是按照程式碼中的順序執行。

在W3C的DOM第3代標準也已經明定順序由註冊到事件目標的監聽者順序決定，而且現在瀏覽器各種不同品牌與新版本，都會依照程式碼中的順序為執行的順序。

事件的冒泡、捕捉

事件的冒泡(往上冒泡)與捕捉(往下捕捉)是兩種事件在DOM中的傳播(propagation)的方式。這是由於DOM元素的樹狀結構，它是有父母-子女關係(parent-child)，這兩種傳播會在事件監聽時，形成一種特別的事件傳播模型，影響事件監聽者在不同父母-子女關係(parent-child)時的執行順序。例如下面的圖解(出自這個網站):



那為什麼會出現兩種相反的事件傳播方式？這是因為在第一次瀏覽器大戰時，Netscape採用了事件捕捉(capturing)，而微軟採用了事件冒泡(bubbling)，現行的W3C標準則一併使用了兩者，目前的瀏覽器品牌中都有支援兩者，而微軟從IE9之後也支援兩者。事件捕捉(capturing)與事件冒泡(bubbling)並沒有說哪一種就比較好，這純粹是看程式設計師的需求而定，不過要特別注意的是，有些幾個事件並沒有支援事件的傳播，例如onfocus或onblur。

那麼要如何控制使用哪一種？一般都是使用事件監聽的方法，以傳入參數值作控制，也就是addEventListener方法的最後一個參數 `phase` (階段)來決定。 `addEventListener` 的語法如下:

> addEventListener(type, handler, phase)

`phase` (階段)是一個布林值，如果是 `false` 就用事件冒泡(bubbling)，如果是 `true` 就使用事件捕捉(capturing)。預設沒寫的話，就是 `false` ，也就是預設使用事件冒泡(bubbling)機制。

> 註: 記法有很多種，自行想像發揮力。例如 fb(false = bubbling)。true與capture都有"t"。抓兔子(捕捉=true)，廢砲(false=冒泡)，浴室才會有泡泡(預設使用泡泡)。

事件冒泡(bubbling)的情況時，當最內部的元素被觸發事件時，會先執行自己本身的事件處理函式，然後才會執行上層父母元素的事件處理函式。以下為範例:

```
const parent = document.getElementById('parent')
const child = document.getElementById('child')

parent.addEventListener('click',
     function(){ console.log('parent clicked') }, false)

child.addEventListener('click',
     function(){ console.log('child clicked') }, false)
```

HTML中的程式碼如下:

```
<div id="parent">
      click me(parent)
    <div id="child">
      click me(child)
    </div>
</div>
```

事件捕捉(capturing)則是倒過來的情況，當最內部的元素被觸發事件時，會先從最外圍的事件處理函式執行，依序到最後才是執行自己本身的處理函式。以下為範例:

```
const parent = document.getElementById('parent')
const child = document.getElementById('child')

parent.addEventListener('click',
     function(){ console.log('parent clicked') },true)

child.addEventListener('click',
     function(){ console.log('child clicked') }, true)
```

HTML中的程式碼如下:

```
<div id="parent">
      click me(parent)
    <div id="child">
      click me(child)
    </div>
</div>
```

不論事件的傳播方式為何種，Event物件提供了 `stopPropagation()` 方法，可以阻止事件的傳播，這可以在某些應用情況中使用。不過這個 `stopPropagation` 方法在事件冒泡(bubbling)與事件捕捉(capturing)兩種不同情況下使用時要特別小心，以免連元素自己的事件處理函式都被擋住。例如像下面的例子:

```
const taiwan =  document.getElementById('taiwan')
const taipei = document.getElementById('taipei')
const me = document.getElementById('me')

taiwan.addEventListener('click',
     function(){ console.log('taiwan clicked') },false)

taipei.addEventListener('click',
     function(){ console.log('taipei clicked') },false)

me.addEventListener('click',
     function(){ console.log('me clicked') }, false)
```

index.html上的HTML程式碼如下，為了明顯標出每個區域，加了顏色與大小的樣式:

```
<div id="taiwan" style="border:1px solid green; height:300px; width:300px">
  Click Taiwan
  <div id="taipei" style="border:1px solid blue; height:200px; width:200px">
```

```
    Click Taipei
    <div id="me" style="border:1px solid red; height:100px; width:100px">Click Me</div>
  </div>
</div>
```

taiwan 是最外圍的DOM元素，然後裡面含有 taipei 層，最裡面是 me 這一層。以下是點按me層元素的結果:

```
//事件冒泡(bubbling)，全部false參數值的結果是
me -> taipei -> taiwan

//事件捕捉(capturing)，全部為true參數值的結果是
taiwan -> taipei -> me
```

假使在第2層中，也就是taipei層加上 stopPropagation 方法，阻止事件傳播，如以下的程式碼:

```
taipei.addEventListener('click',
     function(e){
      console.log('taipei clicked')
      e.stopPropagation()
    },false)
```

你可以再比對一次點按me層元素的結果如下:

```
//事件冒泡(bubbling)，全部false參數值的結果是
me -> taipei (stop!)

//事件捕捉(capturing)，全部為true參數值的結果是
taiwan -> taipei (stop!)
```

也就是說只要經過有阻止傳播的層，就會不再有事件處理函式被觸發，不論這個事件處理函式是不是已經被註冊為該層的事件處理函式，這種特性需要特別小心使用。

另外，混用事件冒泡(bubbling)與事件捕捉(capturing)兩種在程式碼中，絕對是個不智之舉，在簡單的DOM結構時，可能可以推測出來事件處理函式的順序，但在真實情況，DOM的結構的複雜程度是超過你所能想像的，千萬不要這麼作。如果你不確定該使用哪一種方式，使用預設的事件冒泡(bubbling)方式，也就是使用 false 值就行了。

> 註: 事件捕捉(capturing)也有人稱它為事件滴流(trickling)，滴流是向下流動的意思。總之就是向上冒泡的反義詞。

this 與 event.target 、 event.currentTarget

在W3C標準的定義中， this 會相等於 event.currentTarget ，也就是說 this 永遠會指向目前的事件目標對象，就像地震的傳播一樣，地震中心點先震完了，開始傳播到別的地區， event.currentTarget 會跟著改變。事件監聽者(事件處理函式)是一個callback(回調)函式，但這個行為與一般的callback(回調)函式不同，一般的callback(回調)函式的 this 值會總是指向全域的window物件。

但不管是事件冒泡(bubbling)或事件捕捉(capturing)， event.target 永遠指向觸發事件的那個元素，也就是地震的發生源。以下有一張事件冒泡(bubbling)的解說圖片，你可以看一下 event.target 與 this (也就是 event.currentTarget)的比較。(來自這個網站):

自訂事件

自訂事件是由開發者自己建立所需要的事件，再來是附加到某個DOM元素上監聽，最後也是要由開發者自行觸發。會這樣作的原因是，DOM元素上的事件早就被定義固定了，按鈕有按扭可用的事件，表單中的元素有自己的事件。那如果在開發者自己作的應用程式中，想要依照程式中的某個執行功能，定義自訂的事件要怎麼作？例如會員登入時想要用個會員登入事件，聊天室程式常見的好友上線時的好友上線事件，這就是自訂事件的功用了。

首先我們會先使用CustomEvent這個介面(物件)來定義自訂的事件物件，它只需要設定一些屬性，這些屬性與Event中無異，唯一的差異是在於它可以使用 detail 屬性，來額外新加入自訂事件的屬性值，下面是範例:

```
const myEvent = new CustomEvent(
  'userLogin',
  {
    detail: {
                        message: 'Hello World!',
                        time: new Date(),
                },
                bubbles: true,
                cancelable: true
  }
)
```

'userLogin'是這個自訂事件的名稱， `detail` 物件中的屬性是額外的屬性。 `bubbles` 與 `cancelable` 是可以定義也可以不需要定義的屬性，沒定義會使用預設值 `false` ，CustomEvent介面(物件)的屬性是繼承自Event(介面)物件而來，所以會有這兩個屬性。 `bubbles` 值代表可否使用冒泡機制， `cancelable` 則是代表可否使用 `stopPropagation()` 方法。

接下來在你想要監聽這個事件的元素上，加入這個事件與對應的事件處理函式:

```
const myButton = document.getElementById('myButton')

myButton.addEventListener('userLogin', function(e) {
        console.log('Event is: ', e)
        console.log('Custom data is: ', e.detail)
})
```

要觸發這個自訂事件，你只能手動地在程式碼中觸發，使用下面的程式碼:

```
myButton.dispatchEvent(myEvent)
```

自訂事件在IE系統的瀏覽器沒辦法直接使用，有相容性的問題，在其他的瀏覽器上則不會有。IE9之後的瀏覽器可以使用這裡的Polyfill(填充)程式碼或是外部函式庫(如jQuery)來擴充這個功能。

參考資料

- UI Events(W3C)
- Document Object Model (DOM) Level 2 Events Specification(W3C)
- HTML5 - 6.1.5 Events(W3C)
- Event developer guide(MDN)
- Advanced event registration models
- Event order
- Bubbling and capturing
- A crash course in how DOM events work
- The Order of Multiple Event Listeners
- Unable to understand useCapture attribute in addEventListener
- How to Create Custom Events in JavaScript