事件的秘密

# 前言

在上一篇筆記，我們提到了事件的流程有兩種，「事件冒泡」與「事件捕獲」，一個從裡到外;一個從外到裡。

其實在其中隱藏著一個秘密，這個秘密就是今天的主題:

藏在Event Handler中的event

# 藏在Event Handler的event

當監聽事件發生的時候，瀏覽器會去執行我們透過addEventListener()註冊的Event Handler，也就是我們所指定的function。

這個時候Event Handler會去建立一個「事件物件」，裡面包含所有跟這個事件有關的屬性，並且以「參數」的形式傳給我們Event Handler，我們來看一下這個物件給了我們什麼些資訊吧!

CODE



* type 觸發事件的名稱
* target 觸發事件的元素
* bubbles 是否在冒泡階段觸發
* pagex/pagey 事件發生時，游標在網頁的位置

等等的屬性，這裡就不宜個一個說了，不過不是每次的event物件都會一樣，會依照觸發事件的不同而有些許的不同喔!

# e.preventDefault()

HTML部分元素都有預設行為，比如說<a>的連結，或是表單的sumbit等等，那如果我們想要取消這些預設事件，該怎麼做呢?。

現在我們有一個a tag，我們連結到google的網站

<a herf=”google” class=”btn”>Google</a>

假設我們希望當我們點擊的時候，會consle出Google字樣，我們可以設置監聽器，註冊click事件

Code

但我們會發現，當我們點擊連結的時候，依然會把我們帶到google的首頁，如果我們只想要console出GOOGLE字樣，而不要帶我們到連結的頁面的話，我們就可以使用e.preventDefault()來取消原本的預設事件(這裡是連結)

CODE

這樣就可以達到我們的目的囉

※不過要注意，e.preventDefault()不會阻止冒泡事件的傳遞，還有另外值得一提的是，(待確認，試過不行)function return false的話也可以達到e.preventDefault()的效果，不過記得要加在最後面，不然一return false 整個function的執行就結束拉!!

# e.stopPropagation()

如果我們想要阻止事件冒泡向上傳遞，就可以使用e.stopPropagation()。

最常用的情景:

通常為了增加chkbox的易用姓，會特地加個label標籤，然後給個for屬性給對應的chkbox

CODE

<label for=”xxx”></label>

<input type=”checkbox” name=”chkbox” id=”xxx”>

但有時會因為排版需求，不需要那麼多的ID，所以會改成:

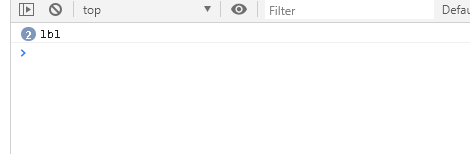
<label class=”lbl”>

Label <input type=”checkbox” name=”chkbox” id=”chkbox”>

</label>

上面這兩個使用起來的效果是一樣的，這時我們再label上註冊click事件的話:

CODE



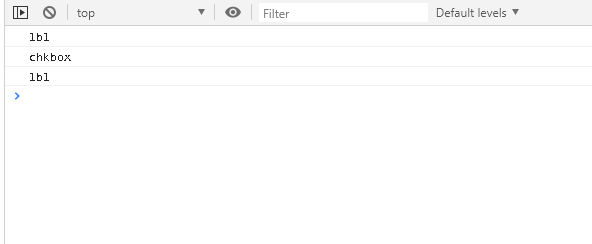
會發現事件被觸發了兩次，也就是lbl console了兩次，為什麼會這樣呢?

是因為label包覆chkbox情況下，當我們點擊label的時候出發了click事件，此時瀏覽器也會把這個click事件帶給chkbox

而因為冒泡事件的影響，chkbox又會把click事件傳遞至label上，導致lbl出現了兩次。

我們分別為chkbox與label來註冊事件看看

CDOE



第一次點擊先觸發了label一次，然後chkbox事件觸發後冒泡事件傳遞向上，lbl又觸發了一次。

既然已經知道是因為事件冒泡的原因，解決這個問題就只要

阻止chkbox冒泡事件向上就行，而這個方法就是e.stopPropagation()

CODE

這樣就完成拉!不過要注意的是要掛在chkbox上，不是在lael上

※另外值得一提的是，很多人會在 jQuery 的 event handler 最後加上 return false 來得到 preventDefault() 與 stopPropagation() 的效果，這是沒問題的。

但是在 JavaScript 的 addEventListener() 裡，最後面加上 return false 只會有 preventDefault() 的效果，不會有 stopPropagation() 的作用。

# 找回自己

如果我們要在觸發的時候，能夠對觸發的元素做某些事情，該怎麼做呢?在Event Handler裡再重新用選擇器挑選嗎?這真是個浩大的工程

其實我們可以用this

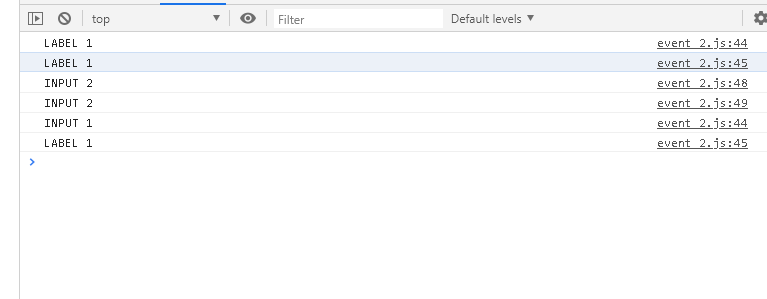
CODE

※this代表的不是e.target(可以透過target來取得觸發的元素)，而是event.currenTarget(觸發事件的目標)

用剛剛的label與input的例子

分別註冊了以下click事件

CODE



前面兩組很正常，先觸發了label後再觸發了chkbox，比較奇特的是最後一組，可以發現都是由label console出的，這時為何e.target與this所代表的是兩個不同的對象呢?

在事件被主動觸發的時候，的確e.target與this所指的是同一個對象(DOM NODE)，但我們已經知道最後的INPUT 1與LABEL 1事由chkbox的冒泡事件機制所發動的(這裡沒有去阻擋冒泡事件)，此時的e.target指的是chkbox

簡單言之，e.target指的是「觸發事件的元素」，而this指的是event.currentTarget「觸發事件的目標」，不考慮事件傳遞的情況下，當然e.target等同於this囉

# 事件委派

假設現在有一list裡面有三個item

<ul class=”list”>

<li class=”item”></li>

<li class=”item”></li>

<li class=”item”></li>

<ul>

我們分別為item註冊click事件，可以用for迴圈來做

CODE

但如果我們這時候新增一個元素到list裡面的話

CODE

會發現新家進來的元素,不會被註冊到clicK事件，就必須為這新的元素在設置直一次監聽事件，不過今天好幾十個元素，幾百個的話，不僅效能與維護上的問題，不斷的去重複監聽事件，又忘了移除監聽，甚至可能會造成memory leak(記憶體衰弱，效能)的嚴重問題

所以我們用事件委派的方法，也就是在父層(list)上監聽，再利用「事件流程」，來達成一個監聽器處理多個項目。

CODE

透過e.target的屬性(可以是ClassName、ID、tagName等等來判斷是不是我們需要的元素都可以)來確定是不是我們的目標節點後，才執行後續的動作，這樣的好處是事件管理會比較輕 松，後續新增的元素也會有click的功效，就無需去另外綁定clicK事件了

# End

事件的介紹就先到這裡，還有許多event物件的屬性可以讓我們去運用與搭配，下回我們稍微介紹下事件的種類，例如滑鼠、鍵盤、表單等等。