Event

# 前言

Javascript為一個事件驅動(Event-driven)的程式語言，當瀏覽器載入之後，雖然會馬上讀取相關程式碼，但要等到「事件被觸發」才會執行相應的程式碼。

什麼意思?什麼是事件?就拿日常生活來說，我們的一個舉動

**電話響了，我接了電話。**

電話響了是一個事件(When)，當電話響時(事件被觸發)，我(Who)才去接電話(What，做對應的事情)。

換作網頁的話，比如說使用者按下的按鈕後，才會啟動對話框的顯示。「點擊按鈕」這件事情我們就稱為「事件(event)」，而負責處理事件的程式通常被稱為「事件處理者(Event Handler)」，也就是負責「啟動對話框」這個動作。

# 事件流程

假如現在有兩層DIV，外層是outer，內層是inner

CODE

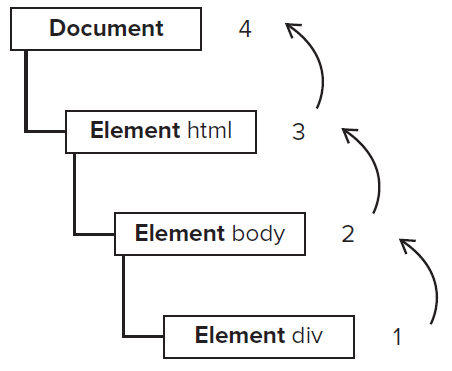
這時內層的位置就在外層的裡面對吧(不考慮absolute)，就表示inner也屬outer的一部份，那麼當我麼點擊inner的時候，也就表示我們也點擊到了outer?甚至在看遠一點，實際上我們點到了整個網頁。

事件流程(Event Flow)指的就是「網頁元素接收事件的順序」。而事件流程分成兩種機制

* 事件冒泡(Event Bubbling)
* 事件捕捉(Event Capturing)

兩種機制怎麼運作跟差異在哪呢?

## 事件冒泡(Event Bubbling)



冒泡事件運作方式經真的跟冒泡一樣，從底層到上層，當我們點擊事件的時候，就會從該節點開始一直往上層傳遞，直到

Document。

就拿我們剛剛的inner與outer來說:

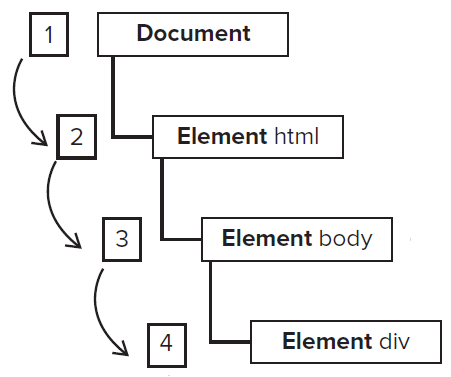
HTML

當我們點擊inner的時候，「事件冒泡」的機制下，事件的觸發順序為

* Innter
* Outer
* Body
* Html
* Document

這就是我們說的「事件冒泡」

## 事件捕捉

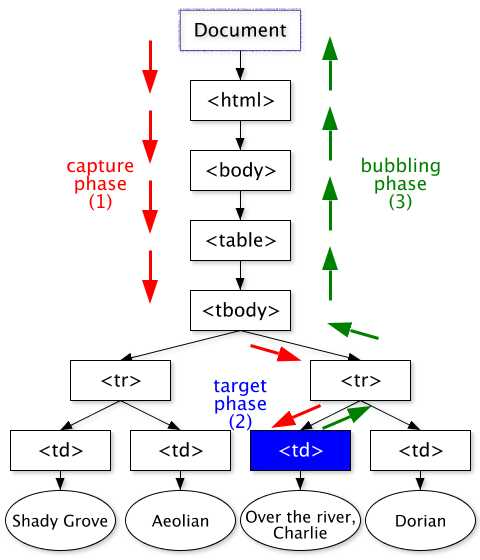


「事件捕獲」則是跟事件冒泡相反，「由上往下」的觸發順序，跟剛剛一樣的HTML下，事件捕獲的觸發順序為

* Document
* Html
* Body
* Outer
* Inner

既然事件機制有這兩種，那我們要怎麼知道當我們點擊按鈕的時候，事件是依賴哪一種機制在執行的呢?

答案是兩種都會執行，拿下圖當作例子



當我們點擊藍色的<td>click事件的時候，會先走紅色的「capture phase」

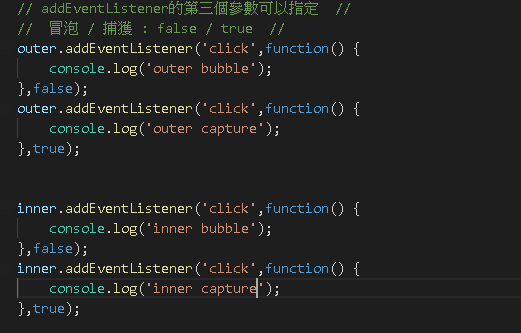
* Document
* Html
* Body
* Table
* Tbody
* Tr
* Td(實際上我們點擊的元素)

由上而下出發他們的click事件，然後再繼續執行綠色的「bubble phase」，反方向的由<td>一直往上觸發，直到document，整個事件流程到此結束。

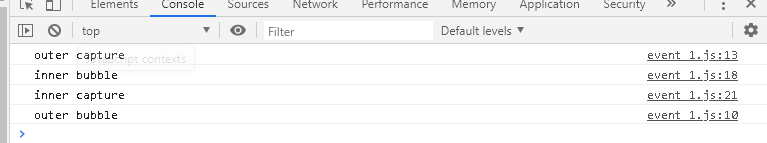
我們可以用addEventListener()方法綁定click事件，來看一下剛剛我們所說的

同樣是outer與inner的HTML，但是我們綁定click事件

CODE



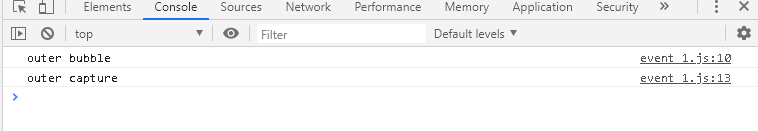
點擊子元素(inner)後結果:



我們會看到觸發的順序為:

* Outer capture
* Inner bubble
* Inner capture
* Outer bubble

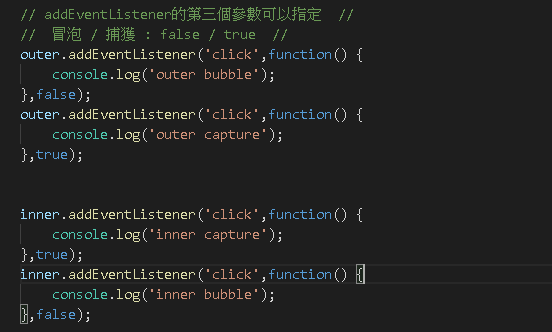
而單單只點擊父元素(outer)的話:



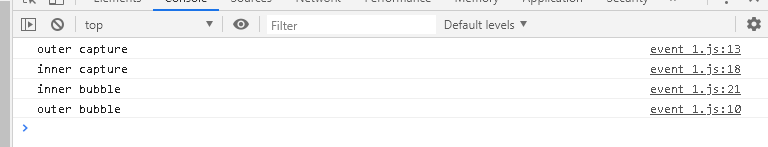
由此可知，當我們點擊子元素的時候，父元素的capture會先觸發，之後才是bubble(想想看剛剛那張圖，先跑紅色的capture再跑綠色bubble的)。

那子層的是先觸發capture 還是 bubble呢?

這就要看程式碼順序而定了，要是剛剛的子層程式碼互相對調位置



結果就會變成:



# 事件的註冊綁定三種方法

## On-event(HTML屬性)

對於HTML元素來說，我們可以用on+事件名的屬性來註冊事件:

<button id=”btn” onclick=”console.log(‘Hi’);”>Click</button>  
上面的button透過了onclick屬性，就可以在button上註冊

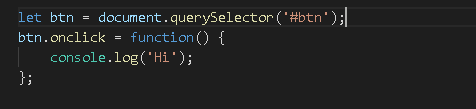
Click事件，也就是說當我按下Button之後，就會執行console.log(‘Hi’)。

但是基於程式碼的使用性與維護考量，現在已經不建使用這樣的方式來綁定事件了。

## On-enevt (非HTML屬性)

比如說剛剛的button，我們就可以透過DOM API取得button後去做事件綁定:

CODE



要解除事件的話指定on-event hendler為null就好

## 事件監聽器(EventTarget.addEventListener() )

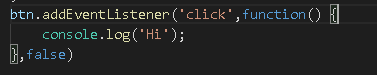
剛剛介紹的 on-event 對應的 function 指的是「事件的處理器」，而現在我們要回頭來說明 addEventListener() 這個「事件的監聽器」。

addEventListener()總共有三個參數，分別是「事件名稱」、「事件的處理器(事件觸發時要執行的function)」以及一個「事件機制(Boolean)」事件機制不指定的話，預設是冒泡機制。

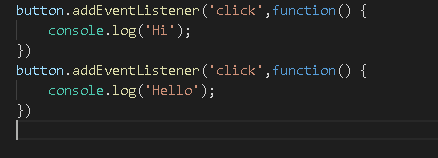
可以想像成「在什麼時候(When)，誰(Who)要去做什麼事情(What)」

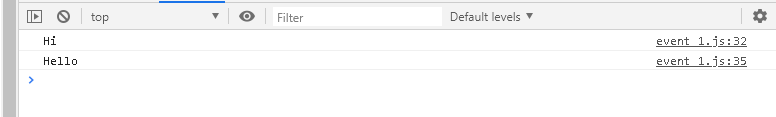
Who.addEventListener(When,What,機制選擇)。

CODE



使用這種方式來註冊事件的好處是可以重複指定多個「處理器」(handler) 給同一個元素的同一個事件：



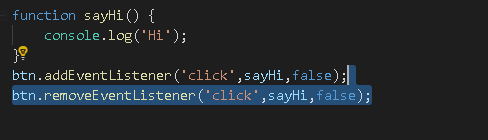


解除事件的話，則是用removeEventListener()來取消。

但是需要注意的是，由於 addEventListener() 可以同時針對某個事件綁定多個 handler，所以透過 removeEventListener() 解除事件的時候，第二個參數的 handler 必須要與先前在 addEventListener() 綁定的 handler 是同一個「實體」。



要像這樣子才會有用



# End

這篇筆記就先到這裡囉!下一篇會談談，關於處理「事件﹞需要的更多注意事項，跟事件的「秘密」。除了click事件之外，還有很多的事件，比如說滑鼠、鍵盤的事件或是元素的狀態等等，之後會介紹。