動作與反應：透過事件讓網頁動起來

當你聽到人們討論 JavaScript 時，他們通常在談話中會提到 " 互動 "：" JavaScript 可讓網頁互動 "。他們真正的意思是說 JavaScript 讓網頁回應訪客的某個動作：移動滑鼠到導航列以顯示下拉選單；按下 radio 按鈕以顯示另外一組選項；點擊小圖以顯示大圖。

網頁能夠回應訪客所有不同的動作稱為事件（*event*），JavaScript 是事件驅動的語言：沒有事件網頁就無法回應訪客或其他事情。它如同桌上電腦，在早上啟動之後，如果沒有開啟應用程式、點擊檔案、選擇功能、移動滑鼠，則不會有什麼行動。

事件是什麼？

瀏覽器能夠辨識網頁載入、滑鼠移動、按鍵等基本動作。發生在網頁上的每件事都是事件。要讓網頁能夠互動時，你會撰寫程式來回應事件。如此就可讓 `<div>` 在訪客點擊滑鼠時出現或消失、顯示圖形，或檢查表單欄位的內容。

事件代表某事發生的瞬間。舉例來說，當你點擊滑鼠時，在釋放滑鼠按鈕的一瞬間，瀏覽器發出 `click` 事件的訊號。程式設計師稱瀏覽器指示事件發生的瞬間為引發（*fire*）事件。

點擊滑鼠時瀏覽器實際上發生了數個事件，首先是在按下滑鼠按鍵時引發 mousedown 事件；然後釋放按鍵時引發 mouseup 事件；最後會引發 click 事件（圖 5-1）。

注意：事件如何引發有點微妙。為了讓你測試不同的事件，這一章的練習檔案中有一個示範網頁。以瀏覽器開啟 *events.html*（在 *testbed* 目錄下），然後移動滑鼠，點擊滑鼠，敲擊鍵盤，看看網頁上有多少事件發生（圖 5-1）。

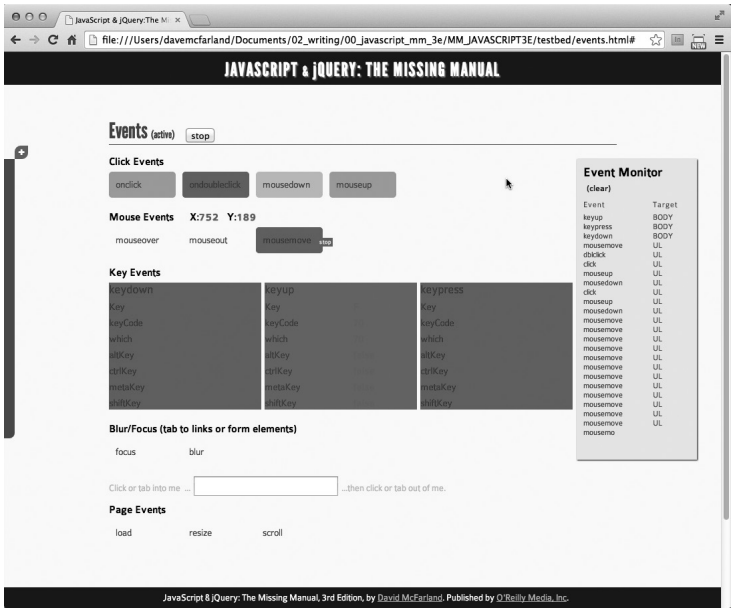

圖 5-1：
雖然你可能不知道，但瀏覽器在移動滑鼠，點擊滑鼠，敲擊鍵盤時會持續引發事件。舉例來說，雙擊滑鼠會引發兩個 mousedown，mouseup、onclick，以及 doubleclick 事件。練習檔中的 *events.html*（在 *testbed* 目錄下）顯示出這些事件。

滑鼠事件

自從 Steve Jobs 在 1984 年推出 Macintosh 之後，滑鼠（以及膝上型電腦的 trackpad）就是個人電腦上的重要配備。人們使用它來開啟應用程式、拖曳檔案、點擊選項，甚至是繪圖。瀏覽器自然也提供許多方法來追蹤訪客以滑鼠與網頁互動：

- **click**。在點擊與釋放滑鼠按鍵之後會引發 click 事件。你通常會指派 click 事件給連結：舉例來說，縮圖的連結可在被點擊時顯示大圖。但此事件不限於連結，你還可以讓網頁上的任何一個標籤回應此事件，就算是點擊空白處也一樣。

注意：click 事件也可透過鍵盤引發。如果以 tab 按鍵移動到連結上然後按下 Enter（Return）鍵也會引發 click 事件。

- **dblclick**。當你快速的連按滑鼠按鍵兩次時會引發雙擊（dblclick）事件。這跟開啟檔案夾或檔案的動作一樣。雙擊在瀏覽過程中不常見，所以若你使用這個事件，應該要讓訪客清楚的知道它的作用。注意雙擊滑鼠會引發 click 以及 dblclick 事件，所以不要同時指派 click 與 dblclick 事件給同一個標籤，不然 click 的程式會在 dblclick 的程式執行前運行兩次。
- **mousedown**。它是點擊動作的前半部，按下按鈕但還沒有放開的一瞬間。這個事件可用來追蹤滑鼠的拖曳。你可以讓訪客在網頁上拖動元素如同在桌面拖動檔案一樣，點擊某個項目（還未放開）並移動，然後在目標上放開按鍵（第 401 頁會討論如何以 jQuery UI 操作）。
- **mouseup**。此事件是點擊動作的後半部，釋放滑鼠按鍵的瞬間。它適用於回應拖放動作。
- **mouseover**。當你移動滑鼠到網頁的某個元素上時會引發 mouseover 事件。你可以指派事件處理程序給使用此事件的按鈕，以使滑鼠移動到按鈕上時顯示子選項（如果有用過 CSS 的 :hover 就知道此事件如何運作）。
- **mouseout**。移動滑鼠離開元素時會引發 mouseout 事件。你可以用此事件來辨識訪客將滑鼠移動到網頁外，或在滑鼠離開選單時隱藏子選項。
- **mousemove**。當滑鼠移動時就會引發 mousemove 事件，這代表此事件會持續的發生。你可使用此事件來追蹤指標目前的位置。此外，你也可以指派此事件給網頁上的特定元素（例如 ）以回應滑鼠在此元素中的移動。

注意：由於 mousemove 事件會被頻繁的引發（移動滑鼠時），指派動作時要小心。如果程式執行時間較久，回應每個滑鼠移動事件會導致整個網頁回應變慢。

document/window 事件

瀏覽器視窗本身會在載入網頁與離開網頁時引發一些事件：

- **load**。此事件在瀏覽器完全載入所有檔案時引發：包括 HTML 檔、圖檔、Flash 影片、外部 CSS 與 JavaScript 檔等。網頁設計者傳統上會使用這個事件來啟動操控網頁的 JavaScript 程式，但若有許多圖檔或大檔案則載入所有檔案需要較長的時間，在某些情況下，這代表 JavaScript 在網頁顯示後會有一段時間不會執行。幸好 jQuery 提供的 `load` 事件可取代以提供更好的回應，細節見第 160 頁。
- **resize**。當你透過按下最大化按鈕，拖曳邊框來調整瀏覽器視窗大小時，瀏覽器就會引發 `resize` 事件。一些設計師會使用此事件在使用者調整視窗大小時改變網頁布局。舉例來說，在使用者調整視窗大小之後，你可以檢查視窗的寬度，如果視窗相當寬，你可以改變設計以放下更多行的內容來填滿空間。
- **scroll**。此事件在拖動捲動軸，或使用鍵盤（例如上下、home、end，與類似的按鍵）與滑鼠中鍵捲動網頁時引發。如果網頁沒有捲動軸就不會引發 `scroll` 事件。有些程式設計師使用此事件來幫助判斷元素是否出現在畫面上（網頁捲動後）。

注意：如同 `mousemove` 事件（第 149 頁），`scroll` 事件會在訪客捲動網頁時持續引發，指派複雜的動作時要謹慎。

- **unload**。當你點擊一個連結去另外一個網頁，關閉瀏覽器分頁，或關閉瀏覽器時，瀏覽器會引發 `unload` 事件。這就像是 JavaScript 的最後一口氣，可以在訪客離開網頁前有機會執行最後一個動作。不良的程式設計師會利用此事件讓離開網頁變得困難，每當訪客要關閉網頁時，就會跳出另一個視窗載入同一頁。但你也可以利用它作點好事：舉例來說，程式可以警告訪客表單尚未送出，或傳送資料給伺服器以在訪客離開前儲存資料。

表單事件

在 JavaScript 出現之前，人們與網站互動主要是靠點擊連結與填寫由 HTML 建構的表單。輸入資訊到表單上還是網路主要的任務之一，你會發現表單有很多學問：

- **submit**。只要訪客送出表單就會引發 submit 事件。表單可能是因為按下送出按鈕，或輸入焦點在文字欄位時按下 Enter（Return）鍵而送出。表單檢驗經常使用 submit 事件－在送出資料到伺服器前確保所有必填欄位正確輸入。第 273 頁會討論如何檢驗表單。
- **reset**。重置按鈕已經不常見，它能取消對表單作的異動，將表單回復到網頁載入時的狀態。你可以在訪客嘗試重置表單時以 reset 事件執行程式。舉例來說，如果訪客作了一些輸入，你可以以對話框詢問訪客是否確定要重置表單。此對話框可以給訪客確認的機會。
- **change**。許多表單欄位會在狀態改變時引發 change 事件：例如點擊 radio 按鈕或在下拉選項中選取。你可以使用這個事件來立即檢查選擇狀態。
- **focus**。以 tab 鍵或點擊文字欄位時會將焦點移動到該欄位上。換句話說，瀏覽器的注意力會集中在網頁的元素上。同樣的，選取 radio 按鈕或 checkbox 時也會轉移焦點。你可以使用 JavaScript 回應 focus 事件。舉例來說，文字欄位內可以事先填入欄位說明－"輸入姓名"。當訪客點擊該欄位時（收到焦點），程式可以刪除欄位說明，讓使用者可以填入資料。
- **blur**。此事件與 focus 相反，它在按下 tab 鍵或點擊欄位以外的地方而離開目前的焦點欄位時引發。blur 事件是另外一種檢驗表單的時機。舉例來說，當訪客在文字欄位輸入郵件帳號，按下 tab 到下一個欄位時，你可以立即檢查輸入值是否為有效的郵件帳號。

注意：focus 與 blur 事件同樣適用於網頁上的連結。連結收到焦點時會引發 focus 事件；離開連結時會引發 blur 事件。

鍵盤事件

瀏覽器也會追蹤訪客的鍵盤動作，因此程式可以指派命令給按鍵。舉例來說，按下空白鍵讓 JavaScript 動畫啟動或停止。

不幸的是不同瀏覽器對鍵盤事件的處理不同，甚至很難分辨到底是按下哪個字元（第 165 頁有辨識輸入字元的技巧）。

- **keypress**。按下按鍵的一瞬間會引發 keypress 事件，此時按鍵不需要釋放按鍵。事實上，只要不放開按鍵，keypress 事件會持續發生，因此可用來判別訪客是否按住按鍵不放。舉例來說，網頁賽車遊戲可以指定一個按鍵當做油門，玩家只要按住按鍵就可以持續加速。
- **keydown**。此事件如同 keypress 事件，按下按鍵時引發，實際上它會在 keypress 事件之前發生。在 Opera 瀏覽器中，keydown 事件只會發生一次，而其他瀏覽器的 keydown 事件則如同 keypress 事件，只要不放開按鍵，事件就會持續發生。
- **keyup**。此事件在放開按鍵時發生。

以 jQuery 處理事件

傳統上，事件的處理很微妙。有很長一段時間，Internet Explorer 處理事件的方式與其他瀏覽器不同，需要寫兩組程式（IE 一組，其他一組）才能讓程式正常運作。幸好 IE9 與之後的版本使用與其他瀏覽器相通的方式處理事件，因此降低程式設計工作量。然而還是有許多人持續使用 IE8，因此需要有對跨瀏覽器處理事件比較好的解決辦法。幸好有 jQuery。

如同上一章所述，jQuery 等 JavaScript 函式庫解決許多 JavaScript 程式設計的問題－包括瀏覽器相容性等。此外，函式庫通常會簡化基本任務。jQuery 可幫助處理事件與事件輔助程式（*event helper*，處理事件的函式）的指派。

第 111 頁討論過 jQuery 程式設計牽涉到兩件事：選取元素與處理元素。事實上，由於事件是 JavaScript 的核心，最好將 jQuery 程式設計視為三個步驟的程序：

1. **選取一或多個元素。**

   前一章解釋過 jQuery 如何讓你使用 CSS 選擇器來選出要操作的網頁元素。指派事件時，你也會選出要與訪客互動的元素。舉例來說，訪客會點擊什麼對象－連結、表格、圖形？如果要指派 mouseover 事件，哪個網頁元素需要引發此事件？

2. **指派事件。**

   在 jQuery 中，大部分的 DOM 事件有相對應的 JavaScript 函式。因此指派事件給元素時，只需加上句號，跟著事件名稱，然後是一組括號。舉例來說，如果要對網頁上的每個連結加上 mouseover 事件時：

   ```
   $('a').mouseover();
   ```

要對 ID 為 menu 的元素加上點擊事件時：

```
$('#menu').click();
```

你可以使用第 148 頁到第 152 頁之間所列出的事件名稱（以及一些第 162 頁討論的 jQuery 事件）。

加入事件之後還有工作要做。為了在事件發生時執行任務，你必須提供函式給事件。

3. 傳遞函式給事件。

最後需要指定事件發生時要作什麼。傳遞給事件的函式由事件發生時要執行的命令所組成：舉例來說，讓隱藏的 顯示或凸顯滑鼠經過的元素。

你可以傳遞已經定義好的函式名給事件，例如：

```
$('#start').click(startSlideShow);
```

指派函式給事件時，要省略呼叫函式時會加上的括號。換句話說，下面的程式有錯：

```
$('#start').click(startSlideShow());
```

但最常見的事件處理函式是**不具名函式**。第 137 頁討論過不具名函式－它是沒有名稱的函式。不具名函式基本結構如下：

```
function() {
// 程式在這裡
}
```

對事件使用不具名函式的基本結構如圖 5-2 所示。

注意：更多關於如何使用 jQuery 與事件的資訊見 *http://api.jquery.com/category/events/*。

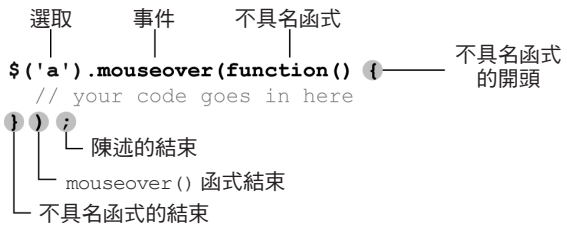

圖 5-2：
在 jQuery 中，事件的運作如同函式，因此可傳遞參數給事件。你可將不具名函式想成是參數－如同傳給函式的資料。這樣想會比較容易看出標點符號的組合。舉例來說，最後一行的 } 標示函式的結束（以及傳給 mouseover 函式的參數的結束）；) 是 mouseover() 函式的結束；分號是整個陳述的結束。

以下有個簡單的例子。假設網頁上有個 ID 為 menu 的連結。當訪客移動滑鼠到連結上，你想要顯示隱藏中的其他連結（假設這些連結的 ID 是 submenu）。因此要做的是將 mouseover 事件加到 menu 上，然後呼叫顯示 submenu 的函式。整個程序分解成四個步驟：

1. **選取 menu：**

```
$('#menu')
```

2. **加上事件：**

```
$('#menu').mouseover();
```

3. **加入不具名函式：**

```
$('#menu').mouseover(function() {

}); // end mouseover
```

你會遇到許多大括號，括號，與分號的組合－ }); －它標示在函式呼叫內部不具名函式的結束。它們隨處可見，因此加上註解－此例中的 // end mouseover －來標示這個符號組合的意義是個好主意。

4. **加上所需執行的任務（此例中是顯示 submenu）：**

```
$('#menu').mouseover(function() {
 $('#submenu').show();
}); // end mouseover
```

許多人很討厭不具名函式所牽涉的標點符號套疊組合，但最好要習慣它。接下來的練習可幫助複習以上的內容。

注意：show() 函式會在下一章第 182 頁討論。

練習：事件入門

這個練習對使用事件作初步的介紹，你會設計能對數個事件回應的網頁，以此了解 jQuery 的事件如何運作與使用。

注意：下載練習檔案的資訊見第 12 頁。

1. **以文字編輯器開啟 *chapter05* 目錄下的 *events_intro.html*。**

   接下來以加上 jQuery 檔案連結開始。

2. **點擊 </head> 上方空行並輸入：**

```
<script src="../_js/jquery.min.js"></script>
```

   這一行從網站內載入 jQuery 檔案。注意目錄名稱是 *_js*（前面有一個底線）。接著加入一組 <script>。

3. **在 jQuery 程式下方插入另外一組 <script>：**

```
<script src="../_js/jquery.min.js"></script>
<script>

</script>
```

   接著加入 document.ready()。

4. **在 <script> 之間輸入下面粗體部分：**

```
<script src="../_js/jquery.min.js"></script>
<script>
$(document).ready(function() {

}); // end ready
</script>
```

別忘記在 });後面加上註解，雖然要多打幾個字，但它對於辨識程式結構很有幫助。此時你已經完成在網頁上使用 jQuery 的基本步驟。

接下來要加上事件，你的第一個目標很簡單：在訪客雙擊網頁任意一點時顯示警示對話框。你需要選取要加上事件的元素（此例中是網頁本身）。

5. 在 .ready() 函式中輸入下面粗體部分：

```
<script src="../_js/jquery.min.js"></script>
<script>
$(document).ready(function() {
 $('html')
}); // end ready
</script>
```

$('html') 選取出 HTML 元素；基本上就是整個瀏覽器視窗。接著加入事件。

6. 在 jQuery 選取器後面輸入 .dblclick(); // end double click：

```
<script src="../_js/jquery.min.js"></script>
<script>
$(document).ready(function() {
  $('html').dblclick(); // end double click
}); // end ready
</script>
```

.dblclick() 是個 jQuery 函式，它在訪客雙擊網頁時讓瀏覽器可以去做一些事情。此處唯一遺漏的部分是 " 做一些事情 "，這需要傳入一個不具名函式作為 dblclick() 的參數（如果需要回顧函式如何運作以及傳遞參數代表的意義，見第 85 頁）。

7. 輸入下面粗體部分以加入不具名函式：

```
<script src="../_js/jquery.min.js"></script>
<script>
$(document).ready(function() {
  $('html').dblclick(function() {

 }); // end double click
}); // end ready
</script>
```

不用擔心，本書接下來不會繼續以這種步調進行，但重點是你必須理解程式每一個片段的作用。function() { }只是外包裝；它在加入程式到 { } 中間之前沒有任何作用，這是接下來要做的事。

8. 最後加上警示對話框陳述：

```
<script src="../_js/jquery.min.js"></script>
<script>
$(document).ready(function() {
  $('html').dblclick(function() {
    alert('ouch');
  }); // end double click
}); // end ready
</script>
```

如果以瀏覽器檢視此頁並雙擊網頁任一處就會出現 JavaScript 警示對話框顯示 "ouch"，如果沒有則檢查程式輸入是否有誤。

注意：花了這麼多時間寫程式，結果只是在螢幕上顯示 "ouch"，這一定讓你很失望。但要知道，alert() 部分不重要一它只是用來展示如何使用 jQuery 的事件。隨著你學習了更多的程式設計與 jQuery 知識，你就可以將 alert() 替換成在訪客雙擊網頁時會執行的一系列工作，例如移動元素、顯示投影秀，或啟動賽車遊戲。

現在基礎結構已經完成，可嘗試其他的事件。

1. 輸入粗體部分的程式：

```
<script src="../_js/jquery.min.js"></script>
<script>
$(document).ready(function() {
  $('html').dblclick(function() {
    alert('ouch');
  }); // end double click
 $('a').mouseover(function() {

  }); // end mouseover
}); // end ready
</script>
```

這段程式碼選出網頁（也就是 $('a')），然後加上 mouseover 事件的不具名函式。換句話說，當滑鼠移動到網頁上的任何連結時，就會發生一些事情。

2. 加上兩個 JavaScript 陳述到剛才建構的不具名函式中：

```
<script src="../_js/jquery.min.js"></script>
<script>
$(document).ready(function() {
  $('html').dblclick(function() {
    alert('ouch');

  }); // end double click
  $('a').mouseover(function() {
    var message = "<p>You moused over a link</p>";
    $('.main').append(message);
  }); // end mouseover
}); // end ready
</script>
```

第一行（`var message = "<p>You moused over a link</p>";`）建構出 `message` 變數並儲存一個字串，此字串代表一個 HTML 段落。第二行選出網頁中 class 名稱為 `main` —也就是 `$('m.main')` —的元素，然後插入（加在該元素後面）`message` 變數的內容。網頁含有一個 class 為 `main` 的 `<div>`，因此這個程式會在訪客移動滑鼠經過網頁上的連結時將 "You moused over a link" 加到 div 後面（jQuery 的 `append()` 函式見第 127 頁）。

3. 存檔並以瀏覽器檢視，移動滑鼠到連結上。

每次移動滑鼠到連結上時，會有一個段落加到網頁上（圖 5-3）。現在對程式加上最後一個設計：訪客點擊表單按鈕時讓瀏覽器改變按鈕上的文字。

4. 輸入下面粗體部分程式碼：

```
<script src="../_js/jquery.min.js"></script>
<script>
$(document).ready(function() {
  $('html').dblclick(function() {
    alert('ouch');
  }); // end double click
  $('a').mouseover(function() {
    var message = '<p>You moused over a link</p>';
    $('.main').append(message);
  }); // end mouseover
 $('#button').click(function() {
  $(this).val("Stop that!");
 }); // end click
```

```
}); // end ready
</script>
```

你應該已經知道這些基本知識：`$('#button')` 選取 ID 為 `button` 的元素（此例中的表單按鈕），然後加上 `click` 事件，因此當按鈕被點擊時就會發生一些事。此例中按鈕會顯示 "Stop that!"。

在第 138 頁有如何在 jQuery 迴圈中使用 `$(this)` 的說明，在事件中有同樣的用法：`$(this)` 指向反應事件的元素－你所選出並加上事件的元素，此例中是表單按鈕（第 254 頁有 jQuery 的 `val()` 函式的討論，但基本上它就是讀取或設定表單元素的值，此例中傳遞 "Stop that!" 給 `val()` 以設定按鈕的值為 "Stop that!"）。

5. 存檔並以瀏覽器檢視，點擊表單按鈕。

按鈕的文字應該立即改變（圖 5-3）。作為一個額外的練習，你可以修改程式讓文字欄位的背景顏色在訪客點擊它時變成紅色。提示：你需要 (a) 選取文字欄位；(b) 使用 `focus()` 事件（第 259 頁）；(c) 在不具名函式中以 `$(this)` 指向文字欄位；(d) 使用 `.css()` 函式（第 132 頁）改變文字欄位的背景顏色。答案（以及完整的網頁）在 *chapter05* 目錄下的 *complete_events_intro.html*。

圖 5-3：
jQuery 可讓網頁與使用者互動變得簡單，例如在雙擊時顯示警示框，在滑鼠移動到連結上時加上文字，或點擊表單按鈕等。

其他 jQuery 事件概念

由於事件是網頁互動的核心，jQuery 包括了一些特定的 jQuery 功能，可讓程式設計更輕鬆，讓網頁更具互動性。

等待 HTML 載入

網頁載入時，瀏覽器一遇到程式碼會嘗試立即執行。因此在網頁前面的程式會在網頁完全載入前就執行－在第 92 頁已經談過，要等程式執行完成網頁才會顯示。不幸的是這種行為通常會引發問題。由於許多 JavaScript 程式設計牽涉到操控網頁內容－顯示跳出訊息、隱藏特定元素、加入一列到表格中等，如果程式嘗試操控還沒有被瀏覽器載入並顯示出的的元素，就會發生錯誤。

處理這個問題最常見的方式是使用瀏覽器的 `onload` 事件來等待網頁完全載入並顯示，然後再執行程式。不幸的是等待網頁完全載入再執行程式會產生相當怪異的結果。`onload` 事件只會在網頁的所有檔案載入後－所有的圖檔、影片、外部樣式表等等－才會發生。因此，若網頁有很多圖檔，訪客可能需要等待下載圖片好幾秒後才看到 JavaScript 程式執行。如果 JavaScript 對網頁做出許多變動－舉例來說，對表格套用樣式，隱藏選單，控制網頁的外觀－訪客會看到網頁突然做出改變。

幸好有 jQuery 可以解決這個問題。相較於依靠載入事件來啟動 JavaScript 程式，jQuery 有一個特殊的 `ready()` 函式可等待 HTML 載入後再執行程式。以此方式，JavaScript 可立即操控網頁而無需等待圖檔或影片（這是複雜且有用的功能－另外一個使用 JavaScript 函式庫的理由）。

你已經在本書幾個練習中使用過 `ready()` 函式，其基本結構像這樣：

```
$(document).ready(function() {
  // 程式在這裡
});
```

基本上，你的所有程式從這個函式內開始。事實上此函式很常用，你或許會在每個使用到 jQuery 的網頁中引用它。你只需要引用它一次，它通常是程式的第一與最後一行。你必須將它放在一對 `<script>` 之間（畢竟還是 JavaScript）以及引用 jQuery 的 `<script>` 與 `</script>` 之後。

因此在這個背景下，此函式看起來會像是這樣：

```
<!DOCTYPE html>
<html>
<head>
```

```
<meta charset="UTF-8">
<title>Page Title</title>
<script src="js/jquery.js"></script>
<script>
$(document).ready(function() {
    // 所有的 JavaScript 程式從此開始
}); //ready() 的結束
</script>
</head>
<body>
The web page content...
</body>
</html>
```

小技巧：由於 ready() 函式幾乎使用在所有引用 jQuery 的網頁上，它有一個縮寫方式。你可以省略 $(document).ready 而寫成這樣：

```
$(function() {
  // do something on document ready
});
```

$(document).ready() 的替代方法

將 $(document).ready() 放在 HTML 文件的 <head> 段可延遲 JavaScript 執行到 HTML 載入為止，但也有其他方法可以達到同樣的效果：將 JavaScript 程式放在 HTML 之後。舉例來說，許多程式設計師將 JavaScript 直接放在 </body> 後面：

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Page Title</title>
</head>
<body>
The web page content...
<script src="js/jquery.js"></script>
<script>
    // 所有的 JavaScript 程式從此開始
</script>
</body>
</html>
```

這種狀況下就不需要 `$(document).ready()`，因為程式載入時文件也已經載入。這種方法有些優點，首先是不需要額外將程式放在 `.ready()` 函式中，其次是載入並執行 JavaScript 會凍結瀏覽器直到程式被載入與執行完成。如果你引用許多外部 JavaScript 檔案且需要一段時間下載，則網頁不會立即顯示。對網站的訪客來說，網頁看起來下載很慢。

你或許會在網路上看到文章說將程式放在最後面是加上 JavaScript 的正確方式，然而這種方式也有缺點。在許多情況下，你加上的 JavaScript 程式對網頁外觀有很重大的影響。舉例來說，你可以使用 JavaScript 完全的重新規劃複雜表格的樣式以讓它更容易檢視與排列。或者你會改變字型讓網頁看起來更酷（*http://letteringjs.com*）。

在這種情況下，如果必須等到 HTML 載入並顯示然後再下載 jQuery 與執行你的 JavaScript 程式，網站的訪客可能會看到網頁未上妝前的樣子（被 JavaScript 化妝前）且被看到打扮過程。這種 " 變臉戲法 " 有時令人不適。此外，如果要建構不能沒有 JavaScript 的網頁應用程式，就沒有理由在載入 JavaScript 之前顯示網頁－畢竟按鈕、工具，與 JavaScript 驅動的介面在 JavaScript 運行之前只是裝飾品。

因此 JavaScript 要放在哪裡的答案是 " 看情況 "。某些情況下，將 JavaScript 放在 HTML 之前能讓網站看起來反應更好，某些情況反而是之後。由於瀏覽器的快取機制，一旦網頁載入必須的 JavaScript 檔案之後，後續其他頁就可以立即從快取取得這些檔案而不用浪費時間再次下載。換句話說，別急：如果你覺得網頁顯示的不夠快，則你可以嘗試將程式移到網頁後面，如果這樣可以改善就繼續下去。但在很多情況下，是否在前面使用 `.ready()` 並沒有多少差別。

注意：在自己的電腦上寫程式並直接以瀏覽器檢視時，不會遇到這一節所討論的問題。它只會在網頁伺服器端跨過網際網路連線時，才有可能看出來載入與顯示網頁所花的時間是否產生任何問題。

滑鼠移入與移出元素

`mouseover` 與 `mouseout` 事件經常併用。舉例來說，當滑鼠在按鈕上方時會顯示一個選單；移動滑鼠離開按鈕時選單就消失。由於結合這兩個事件很常見，jQuery 對此有簡化方式。jQuery 的 `hover()` 函式運作如同其他事

件，但卻以兩個函式而不是一個函式作為參數。第一個函式會在滑鼠移動到元素上方時執行，第二個函式會在滑鼠移出元素時執行。基本架構如下：

```
$('#selector').hover(function1, function2);
```

hover() 通常會使用兩個不具名函式，這種程式看起來有點怪；下面的範例可以讓它清楚一點。假設滑鼠移入 ID 為 menu 的連結時要將（目前隱藏中的）一個 ID 為 submenu 的 DIV 顯示，滑鼠移出時則將 submenu 隱藏，你可以使用 hover() 執行這個功能：

```
$('#menu').hover(function() {
   $('#submenu').show();
}, function() {
   $('#submenu').hide();
}); // end hover
```

要讓帶有多個不具名函式的陳述更容易閱讀，可將每個函式獨立安排，因此上面的程式可以寫成下面這個更容易閱讀的版本：

```
$('#menu').hover(
  function() {
   $('#submenu').show();
  }, // end mouseover
  function() {
   $('#submenu').hide();
  } // end mouseout
); // end hover
```

圖 5-4 顯示此程式如何在 mouseover 與 mouseout 事件上運作。

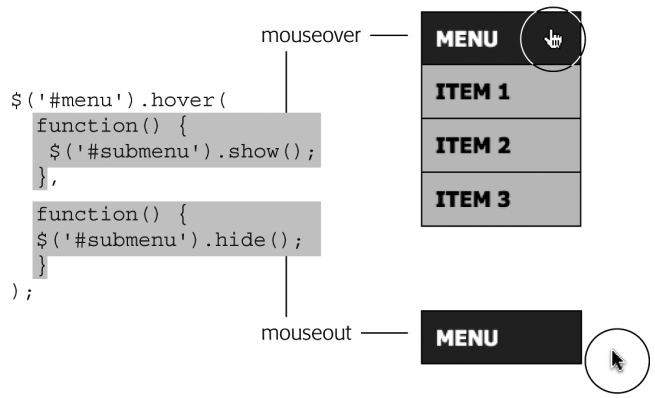

圖 5-4：
jQuery 的 hover() 讓你指派兩個函式，第一個函式會在滑鼠移動到元素上方時執行，第二個函式會在滑鼠移出元素時執行。

如果不具名函式很難理解，你可以使用原始的具名函式（第 84 頁）來完成任務。首先建構一個函式給 mouseover 事件執行；再建構另一個函式給 mouseout 事件；最後將兩個函式的名稱傳給 hover()。換句話說，你可以將上面的程式改寫成這樣：

```
function showSubmenu() {
  $('#submenu').show();
}
function hideSubmenu() {
  $('#submenu').hide();
}
$('#menu').hover(showSubmenu, hideSubmenu);
```

如果你覺得這種方式較清楚，就使用這種方式。兩種方式並沒有實際的差異，然而有些程式設計師喜歡用不具名函式讓程式安排在一個陳述中，而不是展開到不同的陳述中。

注意：在 1.9 版之前的 jQuery 有個非常實用的 toggle() 函式，它的功能如同 hover()，但目標是點擊事件，適合用在點一下顯示，再點一下隱藏的場合。由於 jQuery 不再提供 toggle()，第 175 頁的練習會來重現這個功能。

事件物件

每當瀏覽器引發事件時，它會記錄事件的資訊並儲存於事件物件（*event object*）中。事件物件帶有事件發生時所收集到的資訊，像是滑鼠的垂直與水平座標、事件發生的來源元素，與事件發生時 Shift 鍵是否按著。

在 jQuery 中，指派事件處理的函式可存取事件物件。事實上此物件會被當做參數傳給函式，所以只要在函式中引用變數名稱就可以存取。舉例來說，如果你想要找出滑鼠點擊時指標的 X 與 Y 座標：

```
$(document).click(function(evt) {
  var xPos = evt.pageX;
  var yPos = evt.pageY;
  alert('X:' + xPos + ' Y:' + yPos);
}); // end click
```

重點在於 evt 變數。當函式被呼叫時（滑鼠點擊瀏覽器視窗內任何位置），事件物件就被儲存在 evt 變數中。在函式的內部可以透過 dot 語法存取事件物件的屬性－舉例來說，evt.pageX 回傳指標的水平座標（換句話說，從視窗左緣計算的像素距離）。

注意：此例中，evt 只是程式設計師給的變數名稱，它不是 JavaScript 關鍵字，只是用於儲存事件物件的變數。你可以使用任何名稱，像是 event 或 e。

事件物件有許多不同的屬性，且（很不幸的）各種瀏覽器有各自不同的屬性。表 5-1 列出一些常見的屬性。

表 5-1　每個事件會產生可從事件處理函式中存取的事件物件以及不同的屬性

事件屬性	說明
pageX	滑鼠指標與瀏覽器視窗左邊緣的（像素）距離。
pageY	滑鼠指標與瀏覽器視窗上邊緣的（像素）距離。
screenX	滑鼠指標與螢幕左邊緣的（像素）距離。
screenY	滑鼠指標與螢幕上邊緣的（像素）距離。
shiftKey	若事件發生時 shift 按著則為 true。
which	用於 keypress 事件判別按鍵的字元碼（見下面的說明）。
target	事件 " 對象 " 的物件－舉例來說，點擊事件中被點擊的元素。
data	用於 on() 函式傳遞資料給事件處理函式的 jQuery 物件（第 167 頁）。

小技巧：如果存取 keypress 事件物件的 which 屬性，會取得按鍵碼。若要知道特定的按鍵（a、K、9 等），則必須以 JavaScript 的函式處理 which 屬性來轉換鍵盤碼數字成實際的字元、數字，或符號：

```
String.fromCharCode(evt.which)
```

停止事件行為

有些 HTML 元素有預先設計的事件回應。舉例來說，連結通常在被點擊時載入新網頁；表單的 Submit 按鈕會在被點擊時送出表單資料給網頁伺服器處理。有時你不想要瀏覽器執行預設的行為，舉例來說，當表單送出時（submit() 事件），若有必填欄位未填，你會想要阻止表單送出資料。

你可以透過 preventDefault() 函式阻止瀏覽器的事件預設回應，此函式實際上是事件物件（見前一節）的一部分，因此你可在事件處理函式內存取。舉例來說，若網頁有一個 ID 為 menu 的連結，此連結指向另一個選單網頁（讓關閉 JavaScript 功能的訪客能夠使用選單）。由於你已經對連結加上一些 JavaScript 動作，因此訪客點擊連結時選單會出現在同一頁。通常瀏覽器會跟隨連結到選單網頁，因此要阻止它的預設行為：

```
$('#menu').click(function(evt){
  //JavaScript 動作
 evt.preventDefault(); // 不要跟隨連結
});
```

另外一種技巧是在事件函式最後回傳 false 值，舉例來說，下面程式的功能與上面的程式功能一樣：

```
$('#menu').click(function(evt){
  //JavaScript 動作
  return false; // 不要跟隨連結
});
```

移除事件

有時你需要移除之前指派給標籤的事件，jQuery 的 off() 函式可以移除事件。使用時先建構一個要移除事件對象的 jQuery 物件，然後加上 off() 函式，傳入事件名稱的字串。舉例來說，如果你想要阻止所有 class 為 tabButton 的標籤回應 click 事件，程式可以這麼寫：

```
$('.tabButton').off('click');
```

下面以一段程式來觀察 off() 函式如何運作。

```
1   $('a').mouseover(function() {
2     alert('You moved the mouse over me!');
3   });
4   $('#disable').click(function() {
5     $('a').off('mouseover');
6   });
```

1 至 3 行對網頁上所有的連結（<a> 標籤）加上 mouseover 事件，移動滑鼠到連結上時會開啟警示對話框顯示 "You moved your mouse over me!"。但由於一直出現對話框很煩，4 至 6 行可讓訪客關閉警示。當訪客點擊 ID 為 disable 的標籤（像是一個按鈕）時，所有連結上的 mouseover 事件就會解除，於是不再出現警示框。

如果你要移除元素的所有事件，就不要傳任何參數給 `off()` 函式。舉例來說，若要移除 submit 按鈕的所有事件－ `mouseover`、`click`、`dblclick` 等，程式可以這樣寫：

```
$('input[type="submit"]').off();
```

這是殺傷力很強的方式，大部分情況下你不會移除元素所有的事件處理程序。

注意：更多 jQuery 的 `off()` 資訊見 *http://api.jquery.com/off/*。

程設高級班

終止事件傳遞

事件模型可讓事件越過第一個收到事件的元素傳遞下去。舉例來說，假設你要指派事件處理程序給某個連結的點擊事件；當你點擊連結引發點擊事件時，函式就會執行。然而事件本身並未停在這裡，每一個祖先（ancestor）元素（包圍被點擊元素的標籤）也可以回應同一個點擊。因此若有指派點擊事件輔助程序給包圍連結的 `<div>`，該 `<div>` 的事件函式也會執行。

這種概念稱為**事件泡泡**（*event bubbling*），意思是可以有多個元素回應同一個動作。另外一個例子：假設你對一個圖形加上點擊事件以讓圖形在被點擊時會以另一個圖形取代，此圖形被一個同樣有指派點擊事件的 `<div>` 包圍。此例中的 `<div>` 被點擊時會顯示警示對話框。現在點擊圖形時，兩個函式都會執行。換句話說，就算你只點擊圖形，`<div>` 同樣也會收到點擊事件。

你或許不會經常遇到這種狀況，但遇到時會有點麻煩。假設在上一段的例子中，你不希望 `<div>` 在圖形被點擊時做任何動作，此時你必須阻止點擊事件上升到 `<div>` 且不會妨礙圖形的點擊事件處理程序。換句話說，當圖形被點擊時，點擊事件函式會換圖，然後終止點擊事件。

`stopPropagation()` 函式可阻止事件上升到祖先標籤，它是事件物件（第 164 頁）的函式，因此可從事件處理函式中存取：

```
$('#theLink').click(function(evt) {
  // 執行一些工作
  evt.stopPropagation(); // 終止事件
});
```

進階事件管理

光靠前幾頁討論的 jQuery 事件函式與概念就可以達成程式設計的任務，但若真的想要好好利用 jQuery 的事件處理機制，你需要好好學習 on() 函式。

注意：如果前一節讓你頭疼，可以先跳到第 175 頁的練習感受一下事件處理再回來。

---

on() 函式是比 click() 或 mouseover() 等 jQuery 處理事件的特定函式更為彈性的事件處理方式。它不只讓你指定事件與處理事件的函式，還可以讓你傳入特定資料給事件處理函式使用。這可讓不同的元素與事件（例如點擊連結或滑鼠移動到圖形上）傳遞不同的資訊給同一個事件處理程序－換句話說，一個函式可以根據觸發來源而做出不同的動作。

注意：隨著 jQuery 的演進，加上與移除事件給元素的函式名稱也有變化。你可能會在舊書或網站上發現 bind()、live()，與 delegate() 等函式，這些都已經被 on() 取代。此外，off() 函式取代了原來的 unbind() 函式。

---

on() 的基本結構如下：

```
$('#selector').on('click', selector, myData, functionName);
```

第一個參數是帶有事件名稱的字串（例如 click、mouseover，或其他出現在 148 至 152 頁的事件）。

第二個參數是選擇性的，因此使用 on() 時不一定要指派值。如果有指派參數值，則必須是 tr、 .callout，或 #alarm 之類的選擇器。

注意：第二個參數可用來指派事件給所選出的不同元素，這種技術稱為事件委派（*delegation*），第 171 頁會加以討論。

第三個參數是要傳遞給函式的資料－物件實字或帶有物件實字的變數。物件實字（第 134 頁）基本上是一系列的屬性與值：

```
{
   firstName : 'Bob',
   lastName : 'Smith'
}
```

你可以將物件實字儲存在變數中：

```
var linkVar = {message:'Hello from a link'};
```

第四個參數是另一個函式－事件引發時要執行的動作。此函式可以是不具名函式，或具名的函式－換句話說，這個部分與第 152 頁所討論的正規 jQuery 事件相同。

注意：傳遞資料給 on() 函式是選擇性的，如果只是透過 on() 來加上事件函式，可以省略資料變數：

```
$('selector').on('click', functionName);
```

這跟下面的程式功能相同：

```
$('selector').click(functionName);
```

假設你要以顯示警示框回應事件，但需要根據引發事件的元素來顯示不同的訊息。一種方法是建構有不同物件實字的變數，然後傳給 on() 函式，例如：

```
var linkVar = { message:'Hello from a link'};
var pVar = { message:'Hello from a paragraph'};
function showMessage(evt) {
    alert(evt.data.message);
}
$('a').on('mouseover',linkVar,showMessage);
$('p').on('click',pVar,showMessage);
```

圖 5-5 分解這段程式的運行。它建構 linkVar 與 pVar 兩個變數，每個變數帶有一個物件實字，具有相同的屬性名稱但內文不同。showMessage() 會接受事件物件（第 164 頁）並儲存於 evt 變數中。此函式執行 alert() 命令，顯示 message 屬性（它本身是事件物件的 data 屬性的屬性）的值。要記得 message 是由物件實字定義的屬性名稱。

圖 5-5：
jQuery 的 on() 函式可讓你傳遞資料給回應事件的函式，如此就能以單一具名函式處理多個不同元素（甚至是不同類型的事件），且能夠使用每個事件輔助程序的特定資料。

其他使用 on() 的方式

jQuery 的 on() 函式讓你的程式設計更有彈性，除了上面所討論的技術之外，它還可以讓你綁定兩個或以上的事件到同一個函式中。舉例來說，若要在訪客點擊許多縮圖之一以顯示大圖（許多網站上都有使用的 "lightbox" 效果），並在點擊其他地方或按下任一按鍵時（提供兩個選項給慣用滑鼠與慣用鍵盤的訪客）隱藏大圖：

```
$(document).on('click keypress', function() {
  $('#lightbox').hide();
}); // end on
```

重點在於 'click keypress'，程式透過多個以空白分隔的事件名稱告訴 jQuery 對列出的事件執行此不具名函式。此例中，click 與 keypress 事件都會引發此程序。

此外，如果想要加上數個引發不同動作的事件，你無需使用 on() 多次。換句話說，如果你想要在使用者點擊元素的時候執行某個動作，滑鼠移動到同一個元素上時執行另一個動作，程式可以這樣寫：

```
$('#theElement').on('click', function() {
  // 做些有趣的事情
}); // end on
$('#theElement').on('mouseover', function() {
  // 做些其他有趣的事情
}); // end on
```

你也可以透過傳遞由事件名稱、冒號、不具名函式組成的物件實字給 on() 以達成同樣功能。上面的程式可以改寫成下面這樣，呼叫 on() 一次並傳入物件實字（粗體部分）：

```
$('#theElement').on({
  'click' : function() {
    // 做些有趣的事情
   }, // end click function
   'mouseover' : function() {
     // 做些有趣的事情
   } // end mouseover function
}); // end on
```

以 on() 委派事件

如第 168 頁所述，on() 可以接受第二個參數，它是另一個選擇器：

```
$('#selector').on('click', selector, myData, functionName);
```

第二個參數可以是任何有效的 jQuery 選擇器，像是 ID、class、元素，或任何第 116 頁討論過的選擇器。傳遞選擇器給 on() 大幅的改變 on() 的運作，沒有傳遞選擇器時，事件是套用在原始選擇器上，如上例的 $('#selector')。假設你在網頁中加入下列程式：

```
$('li').on('click', function() {
  $(this).css('text-decoration': 'line-through');
}); // end on
```

這段程式碼對訪客點擊的每個 <li> 標籤套用 'line-through'。要記得此例中的 $(this) 指向處理事件的元素－也就是被點擊的 <li> 元素。換句話說，click 事件被 " 綁定 " 在 <li> 標籤上。在大部分情況下，這就是程式想要的－定義在訪客與特定元素互動時要執行的函式。你可以寫出在訪客點擊連結、滑鼠移至選單、送出表單之類的動作時執行的函式。

注意：還不知道 $(this) 的用途嗎？見第 137 頁的討論以及第 159 頁 $(this) 如何在事件處理程序中使用。

但這種將事件處理程序加入到元素的方式有一個問題：它只能用在已經存在於網頁的元素上。如果動態的在加入如 click()、mouseover()，或 on() 事件處理程序之後再新增 HTML，新的元素不會有任何的事件處理程序。也許很含糊，下面以一個範例來清楚的展示。

想像有一個待辦事項應用程式可讓訪客管理工作，當應用程式首次載入時還沒有待辦事項（圖 5-6 的第一張）。訪客可以填入文字欄位並點擊 Add Task 按鈕以新增工作到清單中（圖 5-6 的第二張）。訪客完成工作後可以點擊該項目以標示完成（圖 5-6 的第三張）。

要將任務標示為完成，你需要加上點擊事件給每個 <li> 標籤，以讓它被點擊時可以標示為完成，圖 5-6 是以淡色畫線方式表示完成。問題是當網頁載入時還沒有 <li> 標籤，因此對每個 <li> 加入點擊事件處理程序的動作沒有效果。換句話說，第 171 頁的程式派不上用場。

注意：更多關於 jQuery 的委派見 *http://api.jquery.com/on/*。

相對的，你必須要委派事件，意思是套用事件到父元素上（已經存在的元素）然後傾聽特定子元素的事件。由於事件套用在已經存在的元素上，加入新的子元素不會影響此程序。換句話說，你將傾聽事件的任務委派給已經存在的父元素。更細節的解釋見第 175 頁。下面是如何以 on() 函式設定委任的範例：

```
$('ul').on('click', 'li', function() {
  $(this).css('text-decoration': 'line-through');
}); // end on
```

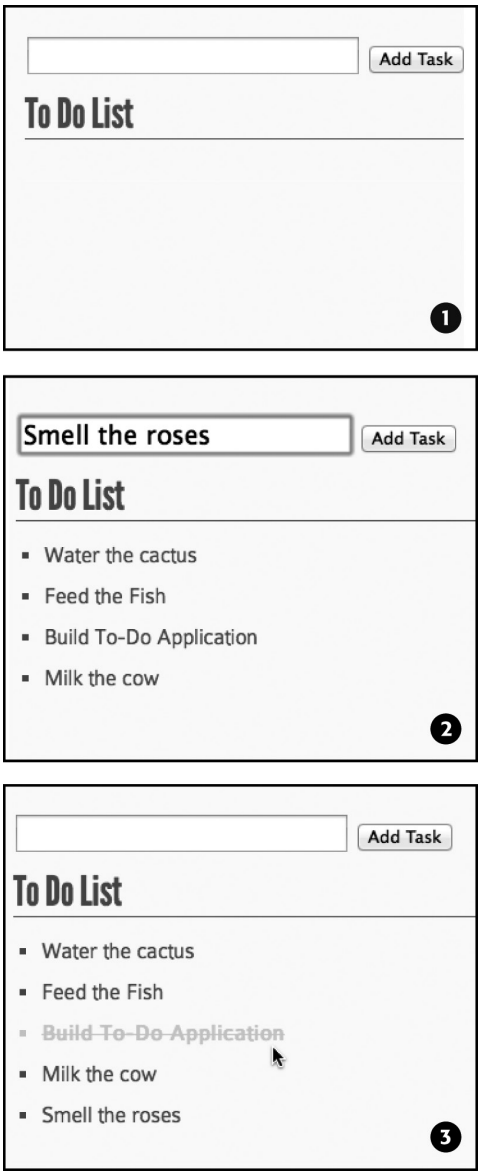

圖 5-6：

有時你會動態的以 JavaScript 新增 HTML 到網頁上。此例中，訪客可以新增項目（待辦事項）到清單上。當網頁初次載入時上面還沒有項目，只是個空的 `<ul></ul>`（上）。當訪客輸入新的項目並點擊 Add Task 按鈕後，新的 `<li>` 就被加入到網頁（中）。要標示任務完成時，只需點擊清單中的項目（下）。這個範例可見練習檔 *chapter05* 目錄下的 *to-do-list.html*。

當你建構網頁時，將空白的 <ul> 加入作為新增 <li> 的容器。如此當網頁載入時，一個空白的 <ul> 就已經就緒。之後執行上面的程式將 on() 加入，訪客此時尚未加入任何待辦事項，因此其中沒有 <li> 標籤。當你加入 'li' 選擇器作為 on() 的第二個參數，這表示你想要傾聽的不是 <ul> 而是任何所屬的 。 何時加入不重要，因為傾聽是由 <ul> 執行。

事件委派如何影響 $(this) 物件

如同之前提過的，$(this) 物件指向目前正在迴圈（第 137 頁）中或事件處理程序（第 159 頁）處理的元素。在事件處理程序中，$(this) 通常指向原始選擇器，例如：

```
$('ul').on('click', function() {
  $(this).css('text-decoration': 'line-through');
}
```

在上面的程式中，$(this) 指向訪客點擊的 <ul> 標籤。然而在事件委派時，原始選擇器不再是原來互動的對象－它只是包含訪客點擊（或滑鼠移動、取得焦點等）對象的元素。再看一次事件委派程式：

```
$('ul').on('click', 'li', function() {
  $(this).css('text-decoration': 'line-through');
}); // end on
```

此例中，訪客會點擊 ，它是必須回應事件的元素。換句話說， 只是容器，函式必須在 <li> 被點擊時執行。因此 $(this) 會指向 ，且上面的函式會將每個被點擊的 <li> 做標示。

許多工作完全不需要事件委任，但如果需要將事件套用於網頁載入時還沒就緒的 HTML 上，則需要使用這種技術。舉例來說，當你使用 Ajax（第十三章）時，會需要使用事件委派來套用事件到從網頁伺服器動態加入到網頁的內容上。

注意：某些情況下，你會想要使用事件委任來提升 JavaScript 效能。如果加入太多標籤給同一個事件處理程序，例如數百個大型表格儲存格，通常將事件委派給 <table> 會比較好：

```
$('table').on('click', 'td', function () {
  // 程式寫在這裡
});
```

加入事件給表格可避免直接套用事件處理程序給數百個儲存格或數千個元素，那個方式會消耗瀏覽器記憶體與處理器運算能力。

練習：單頁 FAQ

"FAQ（常見問答）" 網頁在網路上很常見，它們可以藉由 24 小時無休的執勤幫助提升顧客服務。不幸的是大部分的 FAQ 網頁不是太長，不然就是一頁連結到各個答案頁的問題頁。這兩種解決方案都會拖延訪客尋找答案的過程：第一種方案強迫訪客捲動網頁尋找答案，第二種方案讓訪客等待載入新網頁。

在這個練習中，你會透過建構 JavaScript 驅動的 FAQ 網頁來解決這個問題，所有問答題目在網頁載入時就顯示，因此很容易找到。回答一開始是隱藏的，等到問答題目被點擊後才淡入顯示（圖 5-7）。

任務概觀

這項工作的 JavaScript 需要完成幾件事：

- 在題目被點擊時顯示回答。
- 點擊顯示中的回答就會將它隱藏。

補充說明

事件委派的真相

我能初步理解事件委派，但它實際上如何運作？

如同第 167 頁所述，事件 " 泡泡 " 在網頁中逐層提升。當你點擊一個段落內的連結時，`click` 事件首先在連結上發生，然後父元素（段落）發生 `click` 事件，接下來是 `<body>` 與 `<html>`。換句話說，元素上發生的事件會一路向上傳遞到每個父元素上。

這項事實在事件委派狀況很有用。如第 171 頁所述，有時你會想要將事件套用在還不存在的 HTML 上－例如待辦事項的工作，它們只會在網頁載入與訪客新增項目後出現。雖然你無法新增 `click` 事件到還不存在的 `<li>` 標籤上，但可以將 `click` 事件加入到已經存在的父元素如 `<ul>` 或 `<ul>` 的父元素 `<div>` 上。

如同第 164 頁所述，每個事件都有個記錄許多資訊的事件物件。此例中最重要的資訊是 `target` 屬性，它代表收到事件的特定 HTML 標籤。舉例來說，當你點擊連結時，該連結是點擊的目標。由於事件泡泡的緣故，父標籤會 " 聽到 " 事件，然後判別哪個子元素是目標。

因此，在事件委派中，你可以讓父元素傾聽事件－如同 `<ul>` 傾聽 `click` 事件。然後它檢查哪個標籤才是真正的目標。舉例來說，如果 `<li>` 是目標，則執行特定的程式來處理－例如待辦事項範例。

此外，你會使用 JavaScript 在載入網頁時隱藏所有的回答。為何不使用 CSS 來隱藏回答？舉例來說，設置答案的 CSS 的 display 屬性為 none 也是一種隱藏的方式。這種方式的問題是對沒有開啟 JavaScript 的使用者來說：網頁載入時看不到答案，也無法點擊問題以顯示回答。要讓網頁對有開啟與沒開啟 JavaScript 的訪客都能看到，最好是以 JavaScript 來隱藏網頁內容。

注意：下載練習檔案資訊見第 12 頁。

程式設計

1. 以文字編輯器開啟 *chapter05* 下的 *faq.html*。

此檔案已經連結 jQuery 檔，且已經寫好 $(document).ready() 函式（第 160 頁）。首先在檔案載入時隱藏所有的回答。

2. 在 $(document).ready() 函式輸入 $('.answer').hide();。

每個回答都是放在 class 為 answer 的 中。這一行程式選出 並加以隱藏（hide() 函式見第 182 頁）。存檔後以瀏覽器檢視，回答應該不可見。

注意：元素以 JavaScript 而不是 CSS 隱藏的原因是某些訪客可能會關閉 JavaScript，無法看到此練習的酷炫效果，但至少能夠看到回答。

下一步是判斷哪些元素需要傾聽事件。訪客點擊題目時會顯示回答，因此你必須選出 FAQ 中的每一個問題。此頁中的每個題目均是網頁中的 <h2> 標籤。

3. 輸入下面粗體部分：

```
<script src="../_js/jquery.min.js></script>
<script>
$(document).ready(function() {
  $('.answer').hide();
 $('.main h2')
}); // end of ready()
</script>
```

這是基本的選擇器，會選出每個 class 為 main（不會影響其他的標題）的 <h2> 標籤。現在可以加入事件處理程序。click 事件很合適，但需要做一些設置。對這個範例來說，每個點擊不是將回答顯示就是將它隱藏，看起來是個條件陳述的工作。基本上，你會檢查 <h2> 之下回答的 是否為隱藏：若是則顯示，不是則隱藏。

4. 輸入下面粗體部分以加上事件處理程序給 <h2>：

```
$(document).ready(function() {
  $('.answer').hide();
 $('.main h2').click(function() {

 }); // end click
}); // end of ready()
```

這段程式加上事件處理程序給 <h2>。// end click 註解並不是必要的，但如第 58 頁所提，註解可以幫助標示 });歸屬。加上這一段程式後，無論不具名函式中寫了什麼，都會在訪客點擊 <h2> 時執行。

5. 在不具名函式中間輸入：

```
var $answer = $(this).next('.answer');
```

此處建構一個變數－ $answer －來儲存 jQuery 物件。如同第 159 頁所述，$(this) 指向目前回應事件的元素－此例中的某一個 <h2> 標籤。jQuery 提供數個函式讓網頁結構之間遊走更為容易，.next() 函式找出目前標籤的下一個標籤，換句話說，它找出 <h2> 後面的標籤。.next() 可透過傳入額外的選擇器以更精準的搜尋－ .next('.answer') 找出 <h2> 之後第一個 class 為 answer 的標籤。

換句話說，這行程式儲存 <h2> 後面的 指標，該 內容為題目的回答。將它存在變數中是因為接下來在函式中會有多次的存取：判斷回答是否隱藏，若回答隱藏中則加以顯示，若顯示中則加以隱藏。每次以 $( ) 存取 jQuery 時就會讓瀏覽器執行一堆 jQuery 中的程式碼，因此叫用 $('this').next('.answer') 會讓 jQuery 執行一些工作。相較於重複這些步驟，可以將選取結果儲存在變數中，以變數來指向要顯示或隱藏的 。

當你需要反覆使用同一個 jQuery 選取結果時，將第一次選取結果儲存在變數中反覆使用是個好主意。這會讓程式更有效率，避免瀏覽器執行不必要的工作，讓瀏覽器反應更好。

以 $ 符號開頭的變數（如 $answer）表示儲存的是 jQuery 物件（執行 $( ) 函式的結果）。並不是一定要在變數前面加上 $；這只是 jQuery 程式設計的通用慣例，以表示變數中的物件可以用在 jQuery 函式上（例如 .hide()）。

注意： .next() 函式只是能夠幫助你遊走網頁的 DOM 的許多 jQuery 函式（或稱*方法*）之一。其他函式見 *http://docs.jquery.com/Traversing*，你也可以參考第 537 頁關於 DOM 的函式。

---

6. 加上 if/else 述句：

```
$(document).ready(function() {
  $('.answer').hide();
 $('.main h2').click(function() {
  var $answer = $(this).next('.answer');
  if ( ) {

  } else {

  }
 }); // end click
}); // end of ready()
```

有經驗的程式設計師通常不會這樣輸入 if/else 述句，但一段段建構程式可以幫助學習。接著檢查回答是否隱藏中。

7. 在條件陳述的括號中輸入 $answer.is(':hidden')：

```
 $(document).ready(function() {
  $('.answer').hide();
 $('.main h2').click(function() {
  var $answer = $(this).next('.answer');
  if ($answer.is(':hidden')) {

  } else {

  }
 }); // end click
}); // end of ready()
```

此處使用步驟 5 所建構的 $answer，此變數帶有使用者點擊的 <h2> 後面的 class 為 answer 的元素。記住，它是帶有該題目的回答的 。

你以 jQuery 的 is() 來判斷該元素是否隱藏中。is() 函式檢查元素是否與選擇器相符。傳入 CSS 或 jQuery 的選擇器，該函式會檢查物件是否與指定的選擇器相符。若相符，則函式回傳 true；若不相符則回傳 false。太完美了，條件陳述需要的正是 true 或 false 值（第 66 頁）！

此處的 :hidden 選擇器是只有 jQuery 可用的特殊選擇器，它能辨識隱藏中的元素。此例中，你檢查回答目前是否隱藏，如果隱藏則加以顯示。

8. 加入下面的程式：

```
$(document).ready(function() {
  $('.answer').hide();
 $('.main h2').click(function() {
  var $answer = $(this).next('.answer');
  if ($answer.is(':hidden')) {
   $answer.slideDown();
  } else {

  }
 }); // end click
}); // end of ready()
```

slideDown() 是 jQuery 的動畫函式之一（下一章會討論），它以滑動的方式隱藏元素。此時你可以檢查程式，存檔後以瀏覽器檢視。點擊網頁上的題目，它的回答應該會顯示（如果沒有，檢查程式錯字並以第 18 頁所提方式除錯）。

觀察網頁，你會發現有藍色的 + 符號出現在每個標題前。此加號通常用來標示 " 這裡還有更多的訊息 "。若要標示可以將回答隱藏，將加號替換為減號。這可以透過 <h2> 的樣式來完成。

9. 在前一步驟程式碼後面輸入：

```
$(this).addClass('close');
```

要記得 $(this) 是收到事件的元素（第 159 頁），此例中是 <h2> 標籤。因此程式在回答顯示時加上 close 這個 class。減號是以背景圖定義在樣式表中（可見 CSS 能夠簡化 JavaScript 程式設計）。

接下來你會完成轉置效果的後半部－第二次點擊時隱藏回答。

10. 在條件陳述的 else 部分加入下面粗體部分：

```
<script src="../_js/jquery.min.js></script>
<script>
$(document).ready(function() {
```

```
  $('.answer').hide();
 $('.main h2').click(function() {
  var $answer = $(this).next('.answer');
  if ($answer.is(':hidden')) {
   $answer.slideDown();
   $(this).addClass('close');
  } else {
   $answer.fadeOut();
   $(this).removeClass('close');
  }
 }); // end click
}); // end of ready()
</script>
```

這一部分的程式隱藏回答。你可以使用 `slideUp()` 以有趣的滑動方式隱藏元素，此例中使用的是 `fadeOut()` 函式（第 158 頁有說明）。

最後，你會將名稱為 `close` 的 class 移出 `<h2>` 標籤：這讓加號重新出現在標題左側。

存檔並檢視，現在你點擊題目時不只會出現回答，問題的圖示也會改變（圖 5-7）。

小技巧：完成之後將 `slideDown()` 與 `slideUp()` 換成 `fadeIn()` 與 `fadeOut()`。你喜歡哪一種效果？

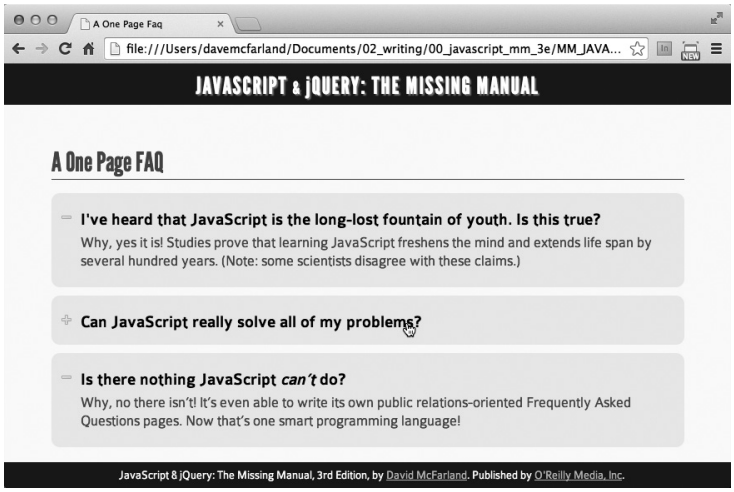

圖 5-7：
使用幾行 JavaScript 就可以在滑鼠點擊時做出顯示與隱藏元素的功能。