第1章：簡介&安裝

jQuery 介紹

jQuery 是一套跨瀏覽器的 JavaScript 函式庫，簡化 HTML 與 JavaScript 之間的操作。由約翰·雷西格（John Resig）在 2006 年 1 月的 BarCamp NYC 上釋出第一個版本。全球前 10,000 個存取最高的網站中，有 65% 使用了 jQuery，是目前最受歡迎的 JavaScript 函式庫。

jQuery 是開源軟體，使用 MIT 授權條款授權。jQuery 的語法設計使得許多操作變得容易，如操作文件（document）、選擇文件物件模型（DOM）元素、建立動畫效果、處理事件、以及開發 Ajax 程式。jQuery 也提供了給開發人員在其上建立外掛程式的能力。這使開發人員可以對底層互動與動畫、進階效果和進階主題化的元件進行抽象化。模組化的方式使 jQuery 函式庫能夠建立功能強大的動態網頁以及網路應用程式。

jQuery 用途

- 可以讓你的 JavaScript 程式碼更簡短、與各種瀏覽器品牌或版本相容性更
- 處理你的 HTML 碼，例如呈現或隱藏等工作
- 處理事件，處理例如使用者點按某個按鈕或其它的滑鼠的動作
- 動畫，例如讓你的網頁的某個部份具有淡出淡入等動畫特效
- AJAX，進行伺服器端的要求，與伺服器端的資料交互應用

為什麼要學 jQuery

- jQuery 可以減少你的開發時間與程式碼問題，減少程式碼的字數與提升閱讀性
- jQuery 易於整合 ASP.net、PHP、Python 等技術
- jQuery 容易學習：相較於 Angular、React 或 Vue 等技術，jQuery 很容易學習與使用，只要有 JavaScript 語言基礎就可以
- jQuery 使用人口很多，資源豐富：網路上文件、範例資源很多，也有很多現成的外掛可以使用

jQuery 安裝

jQuery 可以使用以下兩種安裝方式：

- 從官網下載
- 使用 CDN 外連方式
- 使用 npm 模組安裝方式

從官網下載

官方網站提供了兩種的檔案，一種是經過壓縮的、供於營運使用的，另一種是未經壓縮、供於開發使用的，可與 HTML 檔案放在主機上，引用然後使用。

- 下載網址：https://jquery.com/download/

相關的使用程式碼範例：

```
<!DOCTYPE html>
<html>
  <head>
    <title>jQuery使用範例</title>
    <script src="jquery-3.3.1.min.js"></script>
  </head>
  <body></body>
</html>
```

CDN 外連使用

CDN 外連使用是一種常見的使用方式，但會需要要求網頁必須在有網路連線的情況下使用，常用的 CDN 有以下幾個：

- 官網：https://code.jquery.com/
- Google CDN：https://developers.google.com/speed/libraries/#jquery
- Microsoft CDN：https://docs.microsoft.com/en-us/aspnet/ajax/cdn/overview#jQuery_Releases_on_the_CDN_0

相關的使用程式碼範例：

```
<!DOCTYPE html>
<html>
  <head>
    <title>jQuery使用範例</title>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
  </head>
  <body></body>
</html>
```

註：使用 CDN 方式請儘量使用大公司(微軟、Google)所提供的，另外也有可能因為一段使用時間過後，會有某版本已經更換或失連的問題，這要特別注意。

npm 模組使用方式

npm 模組通常使用於開發階段，直接使用 Node 中所提供的 npm 工具來安裝 jquery 模組，使用的指令如下：

```
npm install jquery
```

第2章：基本使用語法

jQuery Document Ready

由於 HTML 碼載入的順序是從上至下，jQuery 或 JavaScript 載入時如果有需要進行抓取 DOM 元素的參照，在 DOM 元素還未被載入時抓取會產生錯誤，所以必須確保 jQuery 或 JavaScript 程式碼在 HTML 中的 DOM 元素完全被載入後，再開始執行。

jQuery 提供了 Document Ready (文件已準備好了) 的語法，可以相容於各種瀏覽器品牌。常見有兩種語法：

```
$(document).ready(function() {
  alert("DOM is Ready");
});
```

或是

```
// 簡寫法 $( document ).ready()
$(function() {
  alert("DOM is Ready");
});
```

註：如果不透過 Document Ready 的語法，另一種方式就是將 JavaScript 或 jQuery 的程式碼或引用，放置在 HTML 檔案的最下方。

jQuery 基本語法

基本的語法結構：$(selector).action()

- $ 是 jQuery 使用的定義/存取符號(相當於 jQuery)
- (selector) 代表查詢(尋找)某個 HTML 的元素
- action() 代表要對此元素執行某個動作或行為

範例：

- $(this).hide() - 隱藏目前的元素
- $("p").hide() - 隱藏所有的p元素
- $(".test").hide() - 隱藏所有的包含 class="test" 的元素(CSS 類別)
- $("#test").hide() - 隱藏包含 id="test" 的元素(只會有一個)

第3章：選取器(Selector)與修飾篩選

選取器(Selector)

基本的選取器語法基礎有三種。

元素選取器(element)

`$("p").hide()` - 隱藏所有的p元素

範例：

```
$(document).ready(function() {
  $("button").click(function() {
    $("p").hide();
  });
});
```

id 選取器

`$("#test").hide()` - 隱藏包含 id="test" 的元素(只會有一個)

```
$(document).ready(function() {
  $("button").click(function() {
    $("#test").hide();
  });
});
```

類別選取器(class)

`$(".test").hide()` - 隱藏所有的包含 class="test" 的元素(CSS 類別)

```
$(document).ready(function() {
  $("button").click(function() {
    $(".test").hide();
  });
});
```

選取器取回的元素的型態

以上三種常用的選取器方式，不論是哪一種，就算只能得到一個元素的參照元件值，都會將匹配到的元素以 陣列 型態返回，也就是說你可以像下面這樣取得被匹配到元素的個數：

```
// 取得.test元素的個數
$(".test").length;
```

所以這個方式可以用來判斷目前使用的選取器語法有沒有得到任何的元素參照，範例如下：

```
// 測試是否有這個選取有包含任何元素
if ($(".test").length) {
  //...
}
```

選取器得到的元素轉回 JavaScript 元素

要由 jQuery 的選取器得到的結果，回傳一個 JavaScript 的元素參照，可以用以下的語法：

```
var elem = $("#a")[0];
```

或

```
var elem1 = $("#a").get(0);
```

JavaScript 元素 轉為 jQuery 選取器得到的元素

直接使用選取器的語法來轉換，如下範例：

```
var elem1 = $(document.getElementById("foo"));
```

小結範例

```
// non-jquery -> jquery
var a = document.getElementById("some-link"); // 單個元素
var $a = $(a);

// jquery -> non-jquery
a = $a[0]; // jquery元素會回傳一個符合所有條件的陣列
```

註：錢號($)實際上是一個可以在 JavaScript 使用的合法變數命名符號，也可以用於變數名稱開頭。

註： `document.getElementById` 只能用於取得具有某個 id 值的元素，而 jQuery 的選取器可以用於 id 或 class 兩種，注意一種有用井號(#)，另一種沒有。

修飾篩選選取元素

jQuery 的選取器有很多可用的修飾篩選的語法，可以讓開發者可精確地選取元素列於其下。

階層式篩選選取(Hierarchy)

```
// parent child
// 所有在id為myId之內的a元素
$("#myId a");

// parent > child
// 所有在id為myId之內直接地(下一層)的a元素(不包含在子元素中的子元素)
$("#myId > a");

// prev + next
// 選取接在a元素之後的下一個span元素
$("a + span");

// prev ~ siblings
// 選取所有與 <div> 元素位於同階層級的  <p> 元素
$("div ~ p");
```

類別篩選選取(Class)

此類型與 CSS 類別有對應的關係。

```
// .class1,.class2
// 選取類別1或類別2的元素
$(".white,.black");

// div .class
// 選取某種包含某類別的元素
$("div .myClass");
```

```
// elementId > element > element > element
// 選取直接子元素
$("#myData > ul > li > a");
```

索引篩選選取(Index)

```
// :first
// 選取在DOM中的第1個div
$("div:first");

// :last
// 選取在DOM中的最後1個div
$("div:last");

// :even
// 選取在DOM中的偶數tr
$("tr:even");

// :odd
// 選取在DOM中的奇數tr
$("tr:odd");

// :eq(n)
// 選取在tr元素中的第3個td(索引值的第1位是0，所以是第3位置)
$("tr td:eq(2)");
```

子元素篩選選取

```
// :first-child
// 選取所有div內的第一個的span子元素，相當於:nth-child(1)，注意可能會回傳多個
$("div span:first-child");

// :last-child
// 選取所有div內的最後一個的span子元素，注意可能會回傳多個
$("div span:first-child");

// :nth-child(n)
// 選取ul內的第2個的li子元素，注意n自1開始計算，可能會回傳多個
$("ul li:nth-child(2)");
```

屬性篩選選取

```
// [attribute]
// 具有某個屬性的元素
// 選取具有id屬性的div元素
$("div[id]");

// [attribute==value]
// 具有某個屬性，而且值是指定為某固定的值
// 選取具有value屬性，而且值是指定為Hot Fuzz的input元素
$("input[value='Hot Fuzz']");

// [attribute!=value]
// 具有某個屬性，而且值不是指定為固定的值
// 選取具有name屬性，而且值不是指定為newsletter的input元素
$("input[name!='newsletter']");
```

> 注意：雙引號("")號中要使用字串要用單引單('')，或是用跳脫符號

> 註：屬性選擇器實際上是 CSS 標準之一，但 jQuery 的語法更多方式，以上只有列出常見的幾種

內容篩選選取

```
// :contains(text)
// 包含某文字
// 選取有包含文字John的div元素
$("div:contains('John')");

// :has(selector)
// 包含某個selector(元素、類別、或id)
// 選取有包含p元素的div元素
$("div:has(p)");

// :empty
// 元素其中包含的內容為空白(無文字、無子元素…等)
// 選取其中沒有包含文字的p元素
$("p:empty");

// :parent
// 元素其中有包含某些內容(文字、子元素…等)，empty的反義方法
// 選取其中有包含文字(或子元素)的td元素
$("td:parent");
```

第4章：DOM樹狀結構的遍歷/穿越(Traversing)

jQuery 中的遍歷/穿越(Traversing)的意思是，以每個元素之間的關係為基礎，用來尋找(選取)HTML 元素。一開始是使用一個選取，然後進行遍歷直到找到你需要的元素。遍歷的方法非常的多，這會與元素與元素之間的關係有關，可以視情況使用。基本的遍歷關係有三種：

- parents：父母
- children：子女
- siblings：手足(兄弟姐妹)

Parents(父母)

常用的是 parent 和 parents 方法。

```
<div class="grandparent">
  <div class="parent">
    <div class="child"><span class="subchild"></span></div>
  </div>
  <div class="surrogateParent1"></div>
  <div class="surrogateParent2"></div>
</div>


// 選取一個元素的直接父母元素:

// 回傳 [ div.child ]
$("span.subchild").parent();

// 選擇所有的父母元素，額外符合給定的選取器:

// 回傳 [ div.parent ]
$("span.subchild").parents("div.parent");

// 回傳 [ div.child, div.parent, div.grandparent ]
$("span.subchild").parents();
```

Children(子女)

children 方法只會找直接的子女元素，但 find 方法會進行更深層的遍歷，不論是子女元素中的子女都會尋找符合的。

```
<div class="grandparent">
  <div class="parent">
    <div class="child"><span class="subchild"></span></div>
  </div>
  <div class="surrogateParent1"></div>
  <div class="surrogateParent2"></div>
</div>


// 選取直接的子女元素:

// 回傳 [ div.parent, div.surrogateParent1, div.surrogateParent2 ]
$("div.grandparent").children("div");

// 尋找所有在選取中符合條件的所有元素:

// 回傳 [ div.child, div.parent, div.surrogateParent1, div.surrogateParent2 ]
$("div.grandparent").find("div");
```

Siblings(手足)

主要有找前一個元素的 .prev()，找下一個元素的 .next() 以及一起找前後元素的 .siblings()。

因為.prev()與.next()都是尋找單個元素為主要用途，所以又有.nextAll()、.nextUntil()、 .prevAll() 與 .prevUntil() 這幾個可以視情況使用。但要注意它們的回傳都是陣列類型。

```
<div class="grandparent">
  <div class="parent">
    <div class="child"><span class="subchild"></span></div>
  </div>
  <div class="surrogateParent1"></div>
  <div class="surrogateParent2"></div>
</div>


// 選取目前選取元素的下一個手足:

// 回傳 [ div.surrogateParent1 ]
$("div.parent").next();

// 選取目前選取元素的前一個手足:

// 回傳 [] ，因為目前沒有前一個手足元素
$("div.parent").prev();

// 選取目前選取元素的前後手足:

// 回傳 [ div.surrogateParent1, div.surrogateParent2 ]
$("div.parent").siblings();
```

第5章：遍歷/穿越(Traversing) - 過濾(Filter)

jQuery 有提供一系列方法幫助開發者方便的「過濾出」要的目標元素。以下介紹幾個常見的方法。

清單項目元素使用的 first(), last(), eq()

```
<ul>
  <li>list item 1</li>
  <li>list item 2</li>
  <li>list item 3</li>
  <li>list item 4</li>
  <li>list item 5</li>
</ul>
```

```
// 選取第一個項目li元素
$("li")
  .first()
  .css("background-color", "red");

// 選取最後一個項目li元素
$("li")
  .first()
  .css("background-color", "blue");

// 選取索引值為2的項目li元素(索引值是由0開始計算，所以是第3個)
$("li")
  .eq(2)
  .css("background-color", "green");
```

過濾特定的內容或元素 - is(), has(), not(), filter()

is(), has(), not()從字義上就很容易理解是在作什麼過濾用途的，is(), filter()可以使用較為複雜的過濾函式來檢測。

```
<ul>
  <li>list item 1</li>
  <li>list item 2</li>
  <li>list item 3</li>
  <li>list item 4</li>
  <li>list item 5</li>
</ul>
```

```
// 項目2, 4
$("li")
  .not(":even")
  .css("background-color", "red");

// 項目1, 3, 5
$("li")
  .filter(":even")
  .css("background-color", "red");
```

迭代/迴圈 - .each()

.each()的設計是為了要對一個 jQuery 對象中的所有元素，也就是 jQuery 的物件進行遍歷。有一個重點是，回調函式是在當前 DOM 元素為上下文的語境中觸發的。因此關鍵字 this 總是指向這個元素。

語法原型： .each(function(index, Element))

範例：

```
<ul>
  <li>foo</li>
  <li>bar</li>
</ul>
```

```
$( "li" ).each(function( index ) {
  console.log( index + ": " + $(this).text() );
});
```

> 註：元素的.each()方法與另一個重要的$.each()方法(或稱為 jQuery.each)用途相類似，但使用的對象不同。

第6章：DOM 元素的操作處理

jQuery 的基本的 DOM 元素處理方法，通常都可以是兩用的，意思是說如果其中沒有加傳入參數，指的是要獲取目前的數值(getting)。如果其中有加上傳入參教，指的是要設定目前的數值(setting)。也就是 getting 與 setting 都使用同一種方法。

連鎖語法(Chaining)

當你在某個選取元素參照後面呼叫(執行)某個方法，這個方法又會回傳某個 jQuery 的物件(選取的元素參照)，你可以繼續呼叫(執行)某個方法，不需要使用分號(;)來中斷語句，這種語法稱之為連鎖語法(Chaining)。範例如下：

```
$("#content")
  .find("h3")
  .eq(2)
  .html("new text for the third h3!");
```

jQuery 提供了.end() 方法，讓開發者可以在連鎖的中途，回復到最一開始的(原始的)選取元素，範例如下：

```
$("#content")
  .find("h3")
  .eq(2)
  .html("new text for the third h3!")
  .end() // 回復所以選取在#content中的h3元素們
  .eq(0)
  .html("new text for the first h3!");
```

連鎖語法是一種相當有用的、簡便的語法，但仍然需要小心使用，使用太過度會造成難以除錯、有可能會影響效能的情況。

獲取/設置資訊的相關方法

- .html() – 獲取/設置 HTML 內容
- .text() – 獲取/設置文字內容，HTML 部份會被拿掉
- .attr() – 獲取/設置屬性值
- .width() – 獲取/設置寬度(整數值)
- .height() – 獲取/設置高度(整數值)
- .position() – 獲取位置(物件值)，這個方法只能獲取(getter)
- .val() – 獲取/設置表單元素的值

範例：

```
// .html() 方法設置所有 h1 元素的 html 內容為"hello world"
$("h1").html("hello world");

// .html() 方法會回到第1個h1元素的html內容
$("h1").html();
// > "hello world"
```

> 注意：上面的例子，getter 和 setter 的影響或回傳情況不太一樣。

> 註：Setter 會回傳 jQuery 物件，所以在作連鎖語法(Chaining)可以這樣使用。但 Getter 回傳的就看這方法是要得到什麼東西，所以是不能使用連鎖語法(Chaining)。

移動、拷貝與移除元素

jQuery 提供了兩種不同角度的處理元素的位置方式，它們分別是：

- 選取的元素是被進行操作的，然後對另外的元素相對作動作
- 選取的元素是固定位置的元素，然後對另外的元素來相對這個選取元素作操作

範例：

```
// 在#foo元素後面插入p
$("#foo").after("p");

// 相等於
$("p").insertAfter("#foo");
```

插入/移動相關方法

- .insertAfter()
- .after() 外部插入，後面
- .insertBefore()
- .before() 外部插入，前面
- .appendTo()
- .append() 內部插入，後面
- .prependTo()
- .prepend() 內部插入，前面

範例：

```
// 例如 HTML 元素
// <p>I would like to say: </p>

$("p").append("<strong>Hello</strong>");
// 得到的結果
//<p>I would like to say: <strong>Hello</strong></p>

$("p").after("<b>Hello</b>");
// 得到的結果
// <p>I would like to say: </p><strong>Hello</strong>
```

註：如果在上面這些方法的傳入參數中帶入 "jQuery"元素 或 "DOM" 元素物件則代表要移動它們，而不是插入(或新增)。

拷貝元素

使用 .clone 方法可以進行元素的拷貝。

範例：

```
// 拷貝列表中的第一個項目元素，然後加到列表的最後一個
$("#myList li:first")
  .clone()
  .appendTo("#myList");
```

刪除元素

- .remove() 移除
- .detach() 分離(先移除，之後有可能會再用到先作保留)
- .empty() 移除其中包含的子元素(或內容)

建立新元素

使用與選取器相同的語法，其中傳入參數即可包含 HTML 元素的字串，範例如下：

```
// 從一段HTML字串中建立新元件
$("<p>This is a new paragraph</p>");
$('<li class="new">new list item</li>');

// 或是再使用屬性物件作為第二個傳入參數
$("<a/>", {
```

```
  html: "This is a <strong>new</strong> link",
  class: "new",
  href: "foo.html"
});
```

操作屬性

HTML 標記的屬性可以透過 attr()方法來進行操作，這個方法可以進行很複雜的操作，目前可以先學習它的設置方式，基本上有兩種。

針對單個屬性：

```
// 設置.
$("#myDiv a:first").attr("href", "newDestination.html");

// 獲取
$("#myDiv a:first").attr("href");
```

針對多個屬性的設置：

```
// 設置
$("#myDiv a:first").attr({
  href: "newDestination.html",
  rel: "nofollow"
});
```

attr() 方法也可以用像 data-id 這種屬性，不過 jQuery 另外提供了 data() 方法，這兩種用法不完全相同，請不要混用。範例如下：

```
// 範例HTML
// <a data-id="123">link</a>

// 使用 .attr()
$(this).attr("data-id"); // 回傳字串 "123"

// 使用 .data()
$(this).data("id"); // 回傳數字 123
```

第7章：CSS、樣式與尺寸規格

CSS 樣式

.css()可以很方便的設置、獲取樣式資訊。

```
// 獲取 CSS 屬性

$("h1").css("fontSize"); // 回傳 "19px".

$("h1").css("font-size"); // 與上面相同結果


// 設置 CSS 屬性

// 設置單一個屬性
$("h1").css("fontSize", "100px");

// 設置多個屬性
$("h1").css({
  fontSize: "100px",
  color: "red"
});
```

CSS 類別

CSS 類別的套用、移除或檢測，有各自獨立的方法可以使用。

```
var h1 = $("h1");

h1.addClass("big");
h1.removeClass("big");
h1.toggleClass("big");

if (h1.hasClass("big")) {
  // ...
}
```

尺寸規格

```
// 基本的尺寸規格的方法

// 設置所有h1元素的寬度
$("h1").width("50px");

// 獲取第一個h1元素的寬度
$("h1").width();

// 設置所有h1元素的高度
$("h1").height("50px");

// 獲取第一個h1元素的高度
$("h1").height();

// 回傳一個物件，裡面有包含位置的資訊
// 這個位置是第一個h1元素，相對於它的父母元素的位置
$("h1").position();
```

第8章：工具方法

jQuery 提供了許多工具方法，這些方法通常以 jQuery 或$為開頭，在程式碼中處處可以使用。

$.trim()

移除文字字串的前後空白：

```
$.trim("    lots of extra whitespace    ");
```

$.each()

迭代/迴圈於陣列或物件：

```
$.each(["foo", "bar", "baz"], function(idx, val) {
  console.log("element " + idx + " is " + val);
});

$.each({ foo: "bar", baz: "bim" }, function(k, v) {
  console.log(k + " : " + v);
});
```

檢測類型使用

```
$.isArray([]); // true
$.isFunction(function() {}); // true
$.isNumeric(3.14); // true

$.type(true); // "boolean"
$.type(3); // "number"
$.type("test"); // "string"
$.type(function() {}); // "function"
```

第9章：事件處理

jQuery 主要使用.on()方法，但也設計了各種對應不同事件的.click(), .focus(), .blur(), .change()簡便方法。

.on()與.off()方法

語法原型： .on(events [, selector][, data], handler)

範例：

```
// 使用簡便的click方法
$("p").click(function() {
  console.log("You clicked a paragraph!");
});

// 相對於使用 `.on()` 方法
$("p").on("click", function() {
  console.log("click");
});
```

移除事件是使用.off()方法，以下為範例：

```
var foo = function() {
  console.log("foo");
};
var bar = function() {
  console.log("bar");
};

$("p")
  .on("click", foo)
  .on("click", bar);

$("p").off("click", bar); // foo 仍然還是有綁字click事件
```

事件處理函式

事件處理函式可以自動得到一個傳入參數，即為 Event 物件，這個物件對於各種不同事件的操作相當重要。分列幾個重要的屬性或方法如下：

- .preventDefault()：停止預設的動作，例如讓表單不送出、連結不連出等等
- type：事件的類型，例如 click 等等
- data：資料，事件發生時一併綁定的資料
- target：目標對象(DOM 元素)

由於事件處理函數中的 this 為被觸發的「DOM 元素」，而非 jQuery 物件，這個特性很常使用於事件處理函數中，如下的範例：

```
$("a").click(function(eventObject) {
  var elem = $(this);
  if (elem.attr("href").match(/evil/)) {
    eventObject.preventDefault();
    elem.addClass("evil");
  }
});
```

綁定多種事件

```
// 多種事件，同一個處理函式
$("input").on(
  "click change", // 綁定多種事件
```

```
  function() {
    console.log("An input was clicked or changed!");
  }
);


// 多種事件，各自有不同的處理函式
$("p").on({
  click: function() {
    console.log("clicked!");
  },
  mouseover: function() {
    console.log("hovered!");
  }
});
```

只執行一次的事件

.one()方法可以定義在某些特殊情況下，只會被執行一次的事件，如下面的範例：

```
$("p").one("click", firstClick);

function firstClick() {
  console.log("你剛點按了第一次!");

  // 這裡設定了另一種事件處理，是第一次點按後，接下來的事件處理會使用這個
  $(this).click(function() {
    console.log("你剛已經點按過了!");
  });
}
```

第10章：AJAX

異步程序概念

異步 Callback(回調)

我需要再次強調，並非所有的使用 callbacks(回調)函式的 API 都是異步執行的。在 JavaScript 中，除了 DOM 事件處理中的回調函式 9 成 9 都是異步執行的，語言內建 API 中使用的回調函式不一定是異步執行的，也有同步執行的例如 `Array.forEach` ，要讓開發者自訂的 callbacks(回調)的執行轉變為異步，有以下幾種方式:

- 使用計時器(timer)函式: `setTimeout` , `setInterval`
- 特殊的函式: `nextTick` , `setImmediate`
- 執行 I/O: 監聽網路、資料庫查詢或讀寫外部資源
- 訂閱事件

> 註: 執行 I/O 的 API 通常會出現在伺服器端(Node.js)，例如讀寫檔案、資料庫互動等等，這些 API 都會經過特別的設計。瀏覽器端只有少數幾個。

Deferred 物件

jQuery 中的 ajax 相關方法，裡面就有 Deferred(延期)的設計了。Deferred(延期)算是很早就被實作的一種技術，最知名的是由 jQuery 函式庫在 1.5 版本(2011 年左右)中實作的 Deferred 物件，用於註冊多個 callbacks(回調)進入 callbacks(回調)佇列，呼叫 callbacks(回調)佇列，以及在成功或失敗狀態轉接任何同步或異步函式，這個目的也就是 Deferred 物件，或是 Promise 物件實作出來的原因。

Deferred 的設計在 jQuery 中 API 豐富而且應用廣泛，尤其是在 ajax 相關方法中，因為它的語法易用而且功能強大，更是受到很多程式設計師的歡迎。jQuery 所設計的 Deferred 物件中其中有一個 `deferred.promise()` 可以回傳 Promise 物件。歷經多次的改版，現在在 3.0 版本中的 jQuery.Deferred 物件，與現在的 Promises/A+與 ES6 Promises 標準，已經是相容的設計可以交互使用，你可以把它視為是 Promise 的超集或擴充版本。

如果你有需要使用外部函式庫或框架中關於 Promise 的設計，以及使用它們裡面豐富的方法與樣式，建議你不妨先花點時間了解一下 ES6 中的 Promise 特性，畢竟這是內建的語言特性，對於一般的異步程式設計也許已經很足夠。

異步程式設計與 Promise

> promise 物件的設計就是針對異步函式的執行結果所設計的，要不就是用一個回傳值來變成已實現狀態，要不就是用一個理由(錯誤)來變成已拒絕狀態

同步程序你應該很熟悉了，大部份你寫的程式碼都是同步的程序。一步一步(一行一行)接著執行。異步程序有一些你可能用過， `setTimeout` 、 `XMLHttpRequest` (AJAX)之類的 API，或是 DOM 事件的處理，在設計上就是異步的。

我們關心的是以函式(方法)的角度來看異步或同步，函式相當於包裹著要要一起來作某件事的程序語句，雖然函式內的這些程序有可能是同步的也有可能是呼叫到異步的其他函式。JavaScript 中的程式執行的設計是以函式為執行上下文(EC)的一個單位，也只有函式可以進入異步的執行流程之中。

> 同步執行函式的結果要不就是回傳一個值，要不然就是執行到一半發生例外，中斷目前的程式然後拋出例外。

異步的函式結果又會是什麼？要不然就最後回傳一個值，要不然就執行到一半發生例外，但是異步的函式發生錯誤時怎麼辦，可以馬上中斷程式然後拋出例外嗎？不行。那該怎麼作？只能用別的方式來處理。也就是說異步的函式，除了與同步函式執行方式不同，它們對於錯誤的處理方式也要用不同的方式。

> 異步執行函式的結果要不就是帶有回傳值的成功，要不就是帶有回傳理由的失敗。

以一個簡單的比喻來說，你開了一間冰店，可能有些原料是自己作的，但也有很多配料或食材是由別人生產的。同步函式就像你自己作配料的流程，例如自己製作大冰塊、煮紅豆湯之類的，每個步驟都是你自己監管品質，中間如果發生問題(例外)，例如作大冰塊的冰箱壞了，你也可以第一時間知道，而且需要你自己處理，但作大冰塊這件事就會停擺，影響到後面的工作。異步函式是另一種作法，有些配料是向別的工廠叫貨，例如煉乳或黑糖漿，你可以先打電話請工廠進行生產，等差不多時間到了，這些工廠就會把貨送過來。當工廠發生問題時，你可能只是接獲工廠通知，你能作的後續處理有可能是要同意延期交貨或是改向別的工廠叫貨。

Promise 物件的設計就是針對異步函式的執行結果所設計的，promise 物件最後的結果要不然就用一個回傳值來 fulfilled(實現)，要不然就用一個理由(錯誤)來 rejected(拒絕)。

你可能會認為這種用失敗(或拒絕)或成功的兩分法結果，似乎有點太武斷了，但在許多異步的結構中，的確是用成功或失敗來作為代表，例如 AJAX 的語法結構。promise 物件用實現(解決)與拒絕來作為兩分法的分別字詞。對於有回傳值的情況，沒有什麼太多的考慮空間，必定都是實現狀態，但對於何時才算是拒絕的狀態，這有可能需要仔細考量，例如以下的情況:

好的拒絕狀態應該是:

- I/O 操作時發生錯誤，例如讀寫檔案或是網路上的資料時，中途發生例外情況
- 無法完成預期的工作，例如 `accessUsersContacts` 函式是要讀取手機上的聯絡人名單，因為權限不足而失敗
- 內部錯誤導致無法進行異步的程序，例如環境的問題或是程式開發者傳送錯誤的傳入值

壞的拒絕狀態例如:

- 沒有找到值或是輸出是空白的情況，例如對資料庫查詢，目前沒有找到結果，回傳值是 0。它不應該是個拒絕狀態，而是帶有 0 值的實現。
- 詢問類的函式，例如 `hasPermissionToAccessUsersContacts` 函式詢問是否有讀取手機上聯絡人名單的權限，當回傳的結果是 false，也就是沒有權限時，應該是一個帶有 false 值的實現。

不同的想法會導致不同的設計，舉一個明確的實例來說明拒絕狀態的情境設計。jQuery 的 `ajax()` 方法，它在失敗時會呼叫 `fail` 處理函式，失敗的情況除了網路連線的問題外，它會在雖然伺服器有回應，但是是屬於失敗類型的 HTTP 狀態碼時，也算作是失敗的狀態。但另一個可以用於類似功能的 Fetch API 並沒有， `fetch` 使用 Promise 架構，只有在網路連線發生問題才會轉為 rejected(拒絕)狀態，只要是伺服器有回應都算已實現狀態。

> 註: 在 JavaScript 中函式的設計，必定有回傳值，沒寫只是回傳 undefined，相當於 `return undefined`

AJAX 與 XMLHttpRequest 說明

AJAX這個技術名詞的出現是在十年前(2005)，其中內容包含 XML、JavaScript 中的 XMLHttpRequest 物件、HTML 與 CSS 等等技術的整合應用方式，這個名詞並非專指某項特定技術或是軟體，Google 在時所推出的 Gmail 服務與地圖服務，獲得很大的成功，當時這個技術名詞以此作為主要的案例說明。實際上這個技術的實現是在更早之前(2000 年之前)，一開始是微軟公司實作了一個 Outlook 與郵件伺服器溝通的介面，後來把它整合到 IE5 瀏覽器上。在 2006 年 XMLHttpRequest 正式被列入 W3C 標準中，現在已被所有的瀏覽器品牌與新版本所支援。

所謂的 AJAX 技術在 JavaScript 中，即是以XMLHttpRequest物件(簡稱為 XHR)為主要核心的實作。正如它的名稱，它是用於客戶端對伺服器端送出 httpRequest(要求)的物件，使用的資料格式是 XML 格式(但後來 JSON 格式才是最為流行的資料格式)。流程即是建立一個 XMLHttpRequest(XHR)物件，打開網址然後送出要求，成功時最後由回調函式處理伺服器傳回的 Response(回應)。整體的流程是很簡單的，但經過這麼長久的使用時間(11 年)，它在使用上產生不少令人頭痛的問題，例如:

- API 設計得過於高階(簡單)，所有的輸出與輸入、狀態，都只能與這個 XHR 物件溝通取得，進程狀態是用事件來追蹤。
- XHR 是使用以事件為基礎(event-based)的模組來進行異步程式設計。
- 跨網站的 HTTP 要求(cross-site HTTP request)與 CORS(Cross-Origin Resource Sharing)不易實作。
- 對非文字類型的資料處理上不易實作。
- 除錯不易。

XHR 在使用上都是像下面的範例程式碼這樣，其實你可以把它視作一種事件處理的結構，大小事都是依靠 XHR 物件來作，語法中並沒有把每件事情分隔得很清楚，而比較像是擠在一團:

```
function reqListener() {
  const data = JSON.parse(this.responseText);
  console.log(data);
}

function reqError(err) {
  console.log("Fetch Error :-S", err);
}

const oReq = new XMLHttpRequest();
oReq.onload = reqListener;
oReq.onerror = reqError;
oReq.open("get", "./sample.json", true);
oReq.send();
```

在今天瀏覽器功能相當強大，以及網站應用功能複雜的時代，XHR 早就已經不敷使用，它在架構上明顯的有太多的問題，尤其在很多功能的應用情況，程式碼會顯得複雜且不易維護。除非你是有一定要使用原生 JavaScript 的強迫症，要不然現在要作 AJAX 功能時，程式設計師並不會使用原生 XHR 物件來撰寫，大部份時候會使用外部函式庫。因為一個 AJAX 的程式，並不是單純到只有對 XHR 的要求與回應這麼簡單，例如你可能會對伺服器要求一份資料，當成功得到資料後，後面還有需要進一步的資料處理流程，這樣就會涉及到異步程式的執行結構，原生 XHR 並沒有提供可用的方式，它只是單純的作與伺服器互動那件事而已。

XHR Level 2(第 2 級)

XHR 並不是沒有在努力進步，在約 5 年前已經有制定 XHR 的第 2 級新標準，但它仍然與原有 XHR 向下相容，所以整體的模型架構並沒有重大的改變，只是針對問題加以補強或是擴充。目前 XHR 第 2 級在 9 成以上的瀏覽器品牌新版本都已經支援全部的功能，除了 IE 系列要版本 10 之後才支援，以及 Opera Mini 瀏覽器完全不支援，還有一小部份功能在不同瀏覽器上實作細節會有所不同。XHR 第 2 級(5 年前)相較於原有的 XHR(11 年前)多加了以下的功能，這也是現在我們已經可以使用到的 XHR 的新特性:

- 指定回應格式
- 上傳文件與 blob 格式檔案
- 使用 FormData 傳送表單
- 跨來源資源共享(CORS)
- 監視傳輸的進程

不過，XHR 第 2 級的新標準並沒有太引人注目的新功能，它比較像是解決長期以來的一些嚴重問題的補強版本。而且，XHR 在原本上的設計就是這樣，常被批評的是它的語法結構不論在使用與設定都相當的零亂。補強或擴充都還是跳脫不了基本的結構，現今是 HTM5、CSS3 與 ES6 的時代，有許多新的技術正在蓬勃發展，說句實在話，就是 XHR 技術已經舊掉了，當時的設計不符合現在時代需求了，這也無關對或錯。

jQuery 中的作法

外部函式庫例如 jQuery 很早就看到 XHR 物件中在使用的問題，使用像 jQuery 的函式庫來撰寫 AJAX 相關功能，不光是在解決不同瀏覽器中的不相容問題，或是提供簡化語法這麼簡單而已。jQuery 它擴充了原有的 XHR 物件為 jqXHR 物件，並加入類似於 Promise 的介面與 Deferred Object(延遲物件)的設計。

為何要加入類似 Promise 的介面？可以看看它的說明中，是為了什麼而加入的？

> 這些方法可以使用一個以上的函式傳入參數，當 `$.ajax()` 的要求結束時呼叫它們。這可以讓你在單一個(request)要求中指定多個 callbacks(回調)，甚至可以在要求完成後指定多個 callbacks(回調)。 ~譯自 jQuery 官網The jqXHR Object

原生的 XHR 根本就沒有這種結構，Promise 的結構基本上除了是一種異步程式設計的架構，它也可以包含錯誤處理的流程。簡單地來說，jQuery 的目標並不是只是簡化語法或瀏覽器相容性而已，它的目標是要"**取代以原生 XHR 物件的 AJAX 語法結構**"，雖然本質上它仍然是以 XHR 物件為基礎。

jQuery 作得相當成功，十分受到程式設計師們的歡迎，它的語法結構相當清楚，可閱讀性與設定彈性相當高，常讓人忘了原來的 XHR 是有多不好使用。在 Promise 還沒那麼流行的前些年，裡面就已經有類似概念的設計。加上現在的新版本(3.0)已經支援正式的 Promise 標準，說實在沒什麼理由不去使用它。以下是 jQuery 中 ajax 方法的範例:

```
// 使用 $.ajax() 方法
$.ajax({
  // 進行要求的網址(URL)
  url: "./sample.json",

  // 要送出的資料 (會被自動轉成查詢字串)
  data: {
    id: "a001"
  },

  // 要使用的要求method(方法)，POST 或 GET
  type: "GET",

  // 資料的類型
  dataType: "json"
})
  // 要求成功時要執行的程式碼
  // 回應會被傳遞到回調函式的參數
  .done(function(json) {
```

```
  $("<h1>")
    .text(json.title)
    .appendTo("body");

  $("<div class='content'>")
    .html(json.html)
    .appendTo("body");
})
// 要求失敗時要執行的程式碼
// 狀態碼會被傳遞到回調函式的參數
.fail(function(xhr, status, errorThrown) {
  console.log("出現錯誤，無法完成!");
  console.log("Error: " + errorThrown);
  console.log("Status: " + status);
  console.dir(xhr);
})
// 不論成功或失敗都會執行的回調函式
.always(function(xhr, status) {
  console.log("要求已完成!");
});
```

把原生的 XHR 用 Promise 包裹住，的確是一個好作法，有很多其他的函式庫也是使用類似的作法，例如axios與SuperAgent，相較於 jQuery 的多功能，這些是專門只使用於 AJAX 的函式庫，另外這些函式庫也可以用在伺服器端，它們也是有一定的使用族群。

JSON 說明

JSON（JavaScript Object Notation）是一種由道格拉斯·克羅克福特(Douglas Crockford)設計、輕量級的資料格式。它以文字為基礎，且易於閱讀。JSON 資料格式與語言無關，脫胎於 JavaScript，但目前很多程式語言都支援 JSON 格式資料的生成和解析。JSON 的官方 MIME 類型是 application/json，副檔名是 .json。

JSON 使用了物件、陣列、數字、字串、布林值(true/false)、 null 的語法。它以 JavaScript 語言中的物件字面為基礎設計，但並不完全相同，JSON 並不完全是 JavaScript 的子集(參考)。

> 注意：JSON中不能使用 `undefined` 值，而且也不能用 `function` (函式)。

JSON 格式

JSON 主要以"鍵/值(key/value)"的格式來描述數值，一組數值稱為一個鍵值對(key/value pair)

- 鍵(Key): 使用雙引號包含的字串
- 值(Value): 可以是字串(string)、數字(number)、布林(boolean)、陣列(array)或物件(object)
- 鍵值對(Key/Value Pair): 鍵與值間使用冒號(:)分隔，鍵值對之間使用逗號(,)分隔

如下面的範例:

```
"foo" : "bar"
```

陣列

```
"foo" : {
  "bar" : "Hello",
  "baz" : [ "quuz", "norf" ]
}
```

物件

```
"foo" : {
  "bar" : "Hello"
}
```

JavaScript 中的相關方法

JSON.parse

將一個 JSON 字串轉變為 JavaScript 中的數值，如下範例:

```
JSON.parse('{}') // {}
JSON.parse('true') // true
JSON.parse('"foo"') // "foo"
JSON.parse('[1, 5, "false"]') // [1, 5, "false"]
JSON.parse('null') // null
```

JSON.parse方法如果傳入的字串有語法上的錯誤，會直接造成程式中斷，所以需要使用try...catch語句 來捕獲例外(錯誤)，如下範例:

```
if(response) {
    try {
        a = JSON.parse(response);
    } catch(e) {
        alert(e); // error in the above string (in this case, yes)!
```

```
    }
}
```

JSON.stringify

將一個 JavaScript 中的數值轉變為 JSON 字串，如下範例:

```
JSON.stringify({}) // '{}'
JSON.stringify(true) // 'true'
JSON.stringify('foo') // '"foo"'
JSON.stringify([1, 'false', false]) // '[1,"false",false]'
JSON.stringify({ x: 5 }) // '{"x":5}'
JSON.stringify({ x: [10, undefined, function() {}, Symbol('')] })
// '{"x":[10,null,null,null]}'
```

json-server 用法說明

本文說明json-server的用法。

如果不想在電腦中安裝，可以試試它的免費線上測試用版本: jsonplaceholder

步驟一: 安裝 json-server

首先是安裝它，直接用在命令列工具(終端機)中用 npm 工具就可以安裝，裝在全域即可:

```
npm install -g json-server
```

步驟二: 準備 db.json 資料庫檔案

`db.json` 檔案格式就類似像下面這樣，它與 JavaScript 中的物件格式很像，不過它是 JSON 的格式檔案，是有一些差異的。而且是個純文字檔而已:

```
{
  "items": [
    {
      "id": 1482513391121,
      "title": "聽演唱會",
      "isCompleted": true
    }
  ]
}
```

步驟三: 啟動 json-server

啟動它也是用命令列工具(終端機)的指令，當然前提是你要先建立一個 `db.json` 檔案，這個檔案中在我們上面這個範例中有附上(名稱可能是 xxx.json 的檔案，請依不同名稱輸入指令)，命令列(終端機)的對應目錄在 db.json 檔案的同目錄路徑中:

```
json-server --watch --port 5555 db.json
```

這個指令代表要啟動一個在埠號為 5555 的 json-server 伺服器，之後就可以用瀏覽器打開 http://localhost:5555/items

json-server 伺服器的網址後加上 items 會自動只列出 items 裡面的 json 資料，它稱之為 Routes(路由)。

> 註: 我們一般在開發測試 React 應用的是 3000 埠，json-server 伺服器是 5555 埠，相當於在同一個電腦啟動了兩個不同的伺服器。

> 註: 上面的 5555 是隨便取的埠號，你要用 8888 或 9999 也可以，不過埠號有一定的範圍就是。

步驟四: 使用 REST API

因為 json-server 伺服器是個用 REST API 的伺服器，而且又有小型資料庫，又該如何新增、讀取、更新...裡面的資料？

首先你要先理解REST是什麼，也就是在傳資料時的方法(method)值，指定不同值時分別要作不同的事情之用，例如下面幾個:

- POST = 新增
- GET = 讀取
- PUT = 更新
- DELETE = 刪除
- PATCH = 取代部份資料

下面就幾種常用的要作某些事情的範例:

載入所有資料

載入所有資料，預設是用 id 由小至大(ASC)排序:

> GET /items

用 id 排序，由大至小:

> GET /items?_sort=id&_order=desc

新增一筆資料

> POST /items

資料範例:

```
{
  "id": 4,
  "title": "專心學React",
  "isCompleted": false
}
```

更新一筆資料

> PUT /items/4

資料範例:

```
{
  "title": "React學好了",
  "isCompleted": true
}
```

> PATCH /items/4

資料範例:

```
{
  "isCompleted": false
}
```

註：PATCH 與 PUT 類似，但 PATCH 可只針對其中一個資料欄位進行更動，PUT 除了 id 欄位，其它的資料均需要再給定，不論有沒有更動。

刪除一筆資料

> DELETE /items/4

其它說明

json-server 對於要獲取資料有很多的參數值可以使用，例如針對某個欄位進行排序，取出一定的範圍等等，對於開發特定應用時有很多的幫助。詳情請看json-server的說明文件。