

使用 XMLHttpRequest 物件

我們已經討論了動態 Web 應用程式的發展歷史，並簡要介紹了 Ajax，下面再來討論問題的關鍵：如何使用 XMLHttpRequest 物件。儘管與其說 Ajax 是一種技術，不如說是一種技巧，但如果沒有對 XMLHttpRequest 的廣泛支持，Google Suggest 和 Ta-da List 可能不會像我們看到的有今天這樣的發展，而你可能也不會看到手上的這本書！

XMLHttpRequest 最早是在 IE 5 中以 ActiveX 組件形式實現的。由於只能在 IE 中使用，所以大多數開發人員都沒有用 XMLHttpRequest，直到最近，Mozilla 1.0 和 Safari 1.2 把它採用為業界標準（de facto standard），情況才有改觀。需要重點說明的是，XMLHttpRequest 並不是一個 W3C 標準，不過許多功能已經涵蓋在一個新提案中：DOM Level 3 載入和儲存規範（DOM Level 3 Load and Save Specification）。因為它不是標準，所以在不同瀏覽器上的表現也稍

有區別，不過大多數方法和屬性都得到了廣泛的支援。當前，Firefox、Safari、Opera、Konqueror 和 Internet Explorer 都以類似的方式實現了 XMLHttpRequest 物件的行為。

前面已經說過，如果大量使用者還是在使用較舊的瀏覽器存取網站或應用程式，就要三思了。第 1 章討論過，在這種情況下，如果要使用 Ajax 技術，要麼需要開發一個候選網站，要麼你的應用程式應該能妥善地降級。大多數使用統計表明，在當前使用的瀏覽器中只有極少數不支援 XMLHttpRequest，所以一般情況下不會存在這個問題。不過，還是應該查看 Web 日誌，確定你的使用者在使用什麼樣的客戶端來存取網站。

2.1 XMLHttpRequest 物件概述

在使用 XMLHttpRequest 物件發送請求和處理回應之前，必須先用 JavaScript 建立一個 XMLHttpRequest 物件。由於 XMLHttpRequest 不是一個 W3C 標準，所以可以採用多種方法使用 JavaScript 來建立 XMLHttpRequest 的實例。Internet Explorer 把 XMLHttpRequest 實現為一個 ActiveX 物件，其它瀏覽器（如 Firefox、Safari 和 Opera）把它實現為一個原生 JavaScript 物件。由於存在這些差別，JavaScript 程式碼中必須包含有關的邏輯，從而使用 ActiveX 技術或者使用原生 JavaScript 物件技術來建立 XMLHttpRequest 的一個實例。

很多人可能還記得從前的那段日子，那時不同瀏覽器上的 JavaScript 和 DOM 實現簡直千差萬別，聽了上面這段話之後，這些人可能又會不寒而慄。幸運的是，在這裏為了明確該如何建立 XMLHttpRequest 物件的實例，並不需要那麼詳細地編寫程式碼來區別瀏覽器類型。你要做的只是檢

查瀏覽器是否提供對 ActiveX 物件的支援。如果瀏覽器支援 ActiveX 物件，就可以使用 ActiveX 來建立 XMLHttpRequest 物件。否則，就要使用原生 JavaScript 物件技術來建立。程式碼清單 2-1 示範了撰寫跨瀏覽器的 JavaScript 程式碼來建立 XMLHttpRequest 物件實例是多麼簡單。

程式碼清單 2-1 建立 XMLHttpRequest 物件的一個實例

```
var xmlhttp;  
  
function createXMLHttpRequest() {  
    if (window.ActiveXObject) {  
        xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");  
    }  
    else if (window.XMLHttpRequest) {  
        xmlhttp = new XMLHttpRequest();  
    }  
}
```

可以看到，建立 XMLHttpRequest 物件相當容易。首先，要建立一個全域變數 xmlhttp 來儲存這個物件的參考。createXMLHttpRequest 方法完成建立 XMLHttpRequest 實例的實際工作。這個方法中只有簡單的分支邏輯（選擇邏輯）來確定如何建立物件。對 window.ActiveXObject 的呼叫會傳回一個物件，也可能傳回 null，if 陳述會把呼叫傳回的結果看作是 true 或 false（如果傳回物件則為 true，傳回 null 則為 false），以此指示瀏覽器是否支援 ActiveX 控制項，相應地得知瀏覽器是不是 Internet Explorer。如果確實是，則通過實例化 ActiveXObject 的一個新實例來建立 XMLHttpRequest 物件，並傳入一個字串指示要建立何種類型的 ActiveX 物件。在這個例子中，為建構式提供的字串是 Microsoft.XMLHTTP，這說明你想建立 XMLHttpRequest 的一個實例。

如果 window.ActiveXObject 呼叫失敗（傳回 null），JavaScript 就會轉到 else 陳述分支，確定瀏覽器是否把 XMLHttpRequest 實現為一個原生

JavaScript 物件。如果存在 `window.XMLHttpRequest`，就會建立 `XMLHttpRequest` 的一個實例。

由於 JavaScript 具有動態型別特性，而且 `XMLHttpRequest` 在不同瀏覽器上的實現是相容的，所以可以用同樣的方式存取 `XMLHttpRequest` 實例的屬性和方法，而不論這個實例建立的方法是甚麼。這就大大簡化了開發過程，而且在 JavaScript 中也不必撰寫特定於瀏覽器的邏輯。

2.2 方法和屬性

表 2-1 顯示了 `XMLHttpRequest` 物件的一些基本方法。不要擔心，稍後就會詳細介紹這些方法。

表 2-1 標準 `XMLHttpRequest` 操作

| 方法 | 描述 |
|--|--|
| <code>abort()</code> | 停止當前請求 |
| <code>getAllResponseHeaders()</code> | 把 HTTP 請求的所有回應標頭作為鍵/值對傳回 |
| <code>getResponseHeader("header")</code> | 傳回指定標頭的字串值 |
| <code>open("method", "url")</code> | 建立對伺服器的呼叫。 <code>method</code> 參數可以是 GET、POST 或 PUT。 <code>url</code> 參數可以是相對 URL 或絕對 URL。這個方法還包括 3 個可選的參數 |
| <code>send(content)</code> | 向伺服器發送請求 |
| <code>setRequestHeader("header", "value")</code> | 把指定標頭設定為所提供的值。在設定任何標頭之前必須先呼叫 <code>open()</code> |

下面來更詳細地討論這些方法。

- `void open(string method, string url, boolean asynch, string username, string password)`：這個方法會建立對伺服器的呼叫。這是初始化一個請求的純腳本方法。它有兩個必要的參數，還有 3 個可選參數。要提供呼叫的特定方法（GET、POST 或 PUT），還要提供所呼叫資源的 URL。另外還可以傳遞一個 Boolean 值，指示這個呼叫是非同步的還是同步的。預設值為 true，表示請求本質上是非同步的。如果這個參數為 false，處理就會等待，直到從伺服器傳回回應為止。由於非同步呼叫是使用 Ajax 的主要優勢之一，所以倘若將這個參數設定為 false，從某種程度上講與使用 XMLHttpRequest 物件的初衷不太相符。不過，前面已經說過，在某些情況下這個參數設定為 false 也是有用的，比如在持久儲存頁面之前可以先驗證使用者的輸入。最後兩個參數不說自明，允許你指定一個特定的使用者名稱和密碼。
- `void send(content)`：這個方法實際向伺服器發出請求。如果請求聲明為非同步的，這個方法就會立即傳回，否則它會等待直到接收到回應為止。可選參數可以是 DOM 物件的實例、輸入串流，或者字串。傳入這個方法的內容會作為請求本體的一部分發送。
- `void setRequestHeader(string header, string value)`：這個方法為 HTTP 請求中一個給定的標頭設定值。它有兩個參數，第一個字串表示要設定的標頭，第二個字串表示要在標頭中放置的值。需要說明，這個方法必須在呼叫 `open()` 之後才能呼叫。

在所有這些方法中，最有可能用到的就是 `open()` 和 `send()`。XMLHttpRequest 物件還有許多屬性，在設計 Ajax 互動時這些屬性非常有用。

- `void abort()`：顧名思義，這個方法就是要停止請求。
- `string getAllResponseHeaders()`：這個方法的核心功能對 Web 應用程式開發人員應該很熟悉了，它傳回一個字串，其中包含 HTTP 請求的所有回應標頭，標頭包括 `ContentLength`、`Date` 和 `URI`。

- `string getResponseHeader(string header)`：這個方法與 `getAllResponseHeaders()` 是對應的，不過它有一個參數表示你希望得到的指定標頭值，並把這個值作為字串傳回。

除了這些標準方法，XMLHttpRequest 物件還提供了許多屬性，如表 2-2 所示。處理 XMLHttpRequest 時可以大量使用這些屬性。

表 2-2 標準 XMLHttpRequest 屬性

| 屬性 | 描述 |
|---------------------------------|---|
| <code>onreadystatechange</code> | 每個狀態改變時都會觸發這個事件處理器，通常會呼叫一個 JavaScript 函式 |
| <code>readyState</code> | 請求的狀態。有 5 個可取值：0 = 未初始化，1 = 正在載入，2 = 已載入，3 = 互動中，4 = 完成 |
| <code>responseText</code> | 伺服器的回應，表示為一個字串 |
| <code>responseXML</code> | 伺服器的回應，表示為 XML。這個物件可以剖析為一個 DOM 物件 |
| <code>status</code> | 伺服器的 HTTP 狀態碼（200 對應 OK，404 對應 Not Found（未找到），等等） |
| <code>statusText</code> | HTTP 狀態碼的相應文字（OK 或 Not Found（未找到）等等） |

2.3 互動範例

看到這裏，你可能想知道典型的 Ajax 互動是什麼樣。圖 2-1 顯示了 Ajax 應用程式中標準的互動模型。

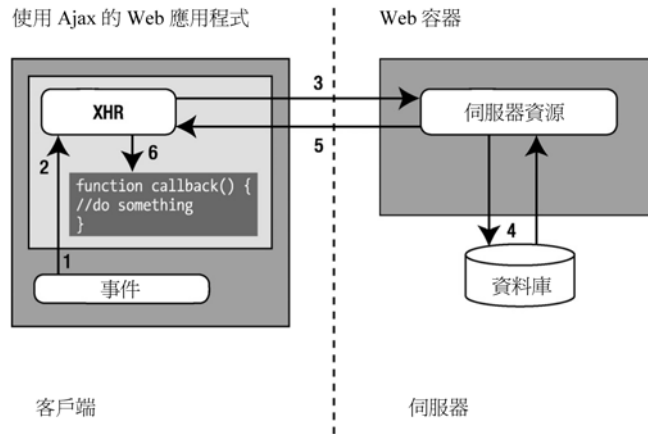


圖 2-1 標準 Ajax 互動

不同於標準 Web 客戶中所用的標準請求/回應方法，Ajax 應用程式的做法稍有差別。

1. 一個客戶端事件觸發一個 Ajax 事件。從簡單的 `onchange` 事件到某個特定的使用者動作，很多這樣的事件都可以觸發 Ajax 事件。可以有如下的程式碼：

```
<input type="text" d="email" name="email" onblur="validateEmail()"/>
```

2. 建立 XMLHttpRequest 物件的一個實例。使用 `open()` 方法建立呼叫，並設定 URL 以及所希望的 HTTP 方法（通常是 GET 或 POST）。請求實際上透過一個 `send()` 方法呼叫觸發。可能的程式碼如下所示：

```
var xmlhttp;
function validateEmail() {
    var email = document.getElementById("email");
    var url = "validate?email=" + escape(email.value);
```

```
if (window.ActiveXObject) {
    xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
}
else if (window.XMLHttpRequest) {
    xmlhttp = new XMLHttpRequest();
}
xmlhttp.open("GET", url);
xmlhttp.onreadystatechange = callback;
xmlhttp.send(null);
}
```

3. 向伺服器做出請求。可能呼叫 servlet、CGI 腳本，或者任何伺服器端技術。
4. 伺服器可以做你想做的事情，包括存取資料庫，甚至存取另一個系統。
5. 請求傳回到瀏覽器。Content-Type 設定為 text/xml——XMLHttpRequest 物件只能處理 text/html 類型的結果。在另外一些更複雜範例中，回應可能涉及更廣，還包括 JavaScript、DOM 管理以及其它相關的技術。需要說明的是，你還需要設定另外一些標頭，使瀏覽器不會在本地快取結果。為此可以使用下面的程式碼：

```
response.setHeader("Cache-Control", "no-cache");
response.setHeader("Pragma", "no-cache");1
```

6. 在這個範例中，XMLHttpRequest 物件配置為處理傳回時要呼叫 callback() 函式。這個函式會檢查 XMLHttpRequest 物件的 readyState 屬性，然後查看伺服器傳回的狀態碼。如果一切正常，callback() 函式就會在客戶端上做些有意思的工作。以下就是一個典型的回呼方法：

```
function callback() {
    if (xmlhttp.readyState == 4) {
        if (xmlhttp.status == 200) {
            //do something interesting here
        }
    }
}
```

¹ Pragma 和 Cache-Control 它們不是做相同的事嗎？是，它們是做相同的事，不過定義 Pragma 是為了保證向後相容。

可以看到，這與正常的請求/回應模型有所不同，但對 Web 開發人員來說，並不是完全陌生的。顯然，在建立和建立 XMLHttpRequest 物件時還可以做些事情，另外當“回呼（callback）”函式完成了狀態檢查之後也可以有所作為。一般地，你會把這些標準呼叫包裝在一個程式庫中，以便在整個應用程式中使用，或者可以使用 Web 上提供的程式庫。這個領域還很新，但是在開放原始碼社群中已經如火如荼地展開了大量的工作。

通常，Web 上提供的各種框架和工具套件負責基本的連接和瀏覽器抽象，有些還增加了使用者介面元件。有一些純粹基於客戶端，還有一些需要在伺服器上工作。這些框架中的很多只是剛開始開發，或者還處於發佈的早期階段，隨著新的程式庫和新的版本的定期出現，情況還在不斷發生變化。這個領域正在日漸成熟，最具優勢的將脫穎而出。一些比較成熟的程式庫包括 libXmlRequest、RSLite、sarissa、JSON（JavaScript Object Notation）、JSRS、DWR（Direct Web Remoting）和 Ruby on Rails。這個領域日新月異，所以應該適當地配置你的 RSS 供應器，及時收集有關 Ajax 的所有網站上的資訊！

2.4 GET 與 POST

你可能想瞭解 GET 和 POST 之間有什麼區別，並想知道什麼時候使用它們。從理論上講，如果請求是冪等的（idempotent）就可以使用 GET，所謂冪等是指多個請求傳回相同的結果。實際上，相應的伺服器方法可能會以某種方式修改狀態，所以一般情況下這是不成立的。這只是一種標準。更實際的區別在於承載（payload）的大小，在許多情況下，瀏覽器和伺服器會限制用於向伺服器發送資料的 URL 長度。一般來講，可以使用 GET 從伺服器獲取資料；換句話說，要避免使用 GET 呼叫改變伺服器上的狀態。