本章內容：

- 資訊共享
- 控制權共用

11

Servlet 協同運作

有多種辦法讓同一伺服器內的 servlet 之間能互相溝通。讓 servlet 之間能彼此溝通，不外乎是為了達成下列兩種合作模式：

資訊共享

這種模式涉及共享同一組狀態或資源的兩個或多個 servlet。例如，一組負責管理線上商店的 servlet 之間，可以共享該商店的產品資訊（庫存量、型錄...），或共享同一個資料庫連線。進程追蹤（session tracking，請參考第七章）算是資訊共享的一種特例。

控制權共用

這種模式讓兩個或多個 servlet 共想同一個 request 的控制權。例如，由某個 servlet 專門負責接收 request，讓另一個 servlet 負責部份（或全部）的 request 處理動作。

在過去（Servlet API 2.1 版之前），還有另一種合作模式：直接運用（direct manipulation）。在這種合作模式中，servlet 可透過 getServlet() 來取得另一個 servlet 的直接參照指標（direct reference），藉此直接呼叫該 servlet 的方法。在 Servlet API 2.1 版之後，這種合作模式已不復存在，getServlet() 也被棄置（deprecated），其定義被改成一律傳回 null。這樣做的理由是 web server 可能在任何時刻結束任何一個 servlet，所以只有 server 有權保有 servlet 的直接參照指標。況且所有可借助 getServlet() 來完成的事，都有更好、更安全的替代辦法 — 而這正是本章的主題。

11.1 資訊共享

Servlet 經常藉由共享某些資訊來互相分工合作。這些共享資訊可能是狀態資訊、共用的資源、資源產地（resource factory）、或是任何東西。在 Servlet API 2.0 版以前，servlet 並沒有內建資訊分享的機制。而在本書的第一版（依據 API 2.0 規格而寫），我們示範了幾種很有創意的解決方案，甚至將共享資訊放在 System properties list！很幸運地，現在已不再需要這些投機取巧的苦差事。因為從 Servlet API 2.1 開始，ServletContext 類別的功能已被強化，可以當成一個共享的資訊庫。

11.1.1 同一 Web 應用系統之內的資訊共享

Servlet 可呼叫 getServletContext() 來取得其 Web 應用系統的 ServletContext 物件。Servlet 可將這個 ServletContext 物件當成 Hashtable 或 Map 之類的物件來使用。ServletContext 物件提供下列方法：

```
public void ServletContext.setAttribute(String name, Object o)
public Object ServletContext.getAttribute(String name)
public Enumeration ServletContext.getAttributeNames()
public void ServletContext.removeAttribute(String name)
```

setAttribute() 方法將指定的物件（Object o）繫結（bind）到指定的名稱（String name），若存在任何同名的連繫關係，新的「物件－名稱」對應關係會取代舊的對應。屬性名稱的命名方式，應該依循套件名稱的命名慣例，以免發生新名蓋寫舊名的狀況。當然，java.*、javax.* 和 com.sun.* 這幾個套件名稱已被保留，不可使用。

getAttribute() 方法傳回指定名稱（String name）所對應的 Object。若該屬性不存在，則傳回 null。如同第四章《擷取資訊》的討論，getAttribute() 方法也可取得伺服器專屬的固定屬性（比方說 javax.servlet.context.tempdir）。

getAttributeNames() 傳回一個 Enumeration 物件，這物件包含所有已繫結的屬性之名稱。如果傳回空的 Enumeration 物件，表示目前沒有任何繫結關係。

removeAttribute() 方法移除所指定名稱（String name）與其對應物件的繫結關係，若指定的屬性不存在，則不進行任何動作。適時移除已沒有用的屬性，可以減低記憶體的耗用量。

應用實例

舉一個有趣的例子。想像有一組銷售洋春捲（burritos）【編註】的 servlet，它們需要共用「本日特價品」的資訊。負責主導的「本日特價品」的 servlet，可用範例 11-1 所示的技巧來設定「本日特價品」：

範例 11-1：設定本日特價品

```
import java.io.*;
import java.util.*;
import javax.servlet.*;

import javax.servlet.http.*;

public class SpecialSetter extends HttpServlet {

  public void doGet(HttpServletRequest req, HttpServletResponse res)
                              throws ServletException, IOException {
    res.setContentType("text/plain");
    PrintWriter out = res.getWriter();
```

編註　我們不知道 burritos 的正式譯名，但是知道其作法與外觀很像春捲，所以姑且稱為「洋春捲」— 反正這與本書主題無關，所以就不講究這麼多了。

範例 11-1：設定本日特價品（續）

```
      ServletContext context = getServletContext();
      context.setAttribute("com.costena.special.burrito", "Pollo Adobado");
      context.setAttribute("com.costena.special.day", new Date());

      out.println("The burrito special has been set.");
    }
  }
```

設定好之後，同伺服器上的其它 servlet，就可利用範例 11-2 的程式碼，來取得並顯示本日特價品的資訊：

範例 11-2：取得本日特價品

```
import java.io.*;
import java.text.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;


public class SpecialGetter extends HttpServlet {

  public void doGet(HttpServletRequest req, HttpServletResponse res)
                              throws ServletException, IOException {
    res.setContentType("text/html");
    PrintWriter out = res.getWriter();

    ServletContext context = getServletContext();
    String burrito = (String)
      context.getAttribute("com.costena.special.burrito");
    Date day = (Date)
      context.getAttribute("com.costena.special.day");

    DateFormat df = DateFormat.getDateInstance(DateFormat.MEDIUM);
    String today = df.format(day);
```

```
    out.println("Our burrito special today (" + today + ") is: " + burrito);
  }
}
```

11.1.2 不同 Web 應用系統之間的資訊共享

利用 ServletContext 分享資訊的好處，在於每一個 Web 應用系統都有各自專用的資訊貯存區；因此，就算在同一個伺服器上安裝（deploy）同一個 Web 應用系統兩次，也不必擔心會發生同名屬性的衝突。然而，有些時候我們需要不同的 web context 也能共享同一組資訊，在這種情況下，我們有兩種選擇：第一，使用外部的資訊儲藏庫，像是 singleton（單一個體）或資料庫。或者，利用一個特殊的掛勾（hook）來直接存取另一個環境（context）。我們將解釋第二種作法。

Servlet 可以使用自身環境的 getContext() 來取得同伺服器上的其它環境之代碼（handle）：

```
public ServletContext ServletContext.getContext(String uripath)
```

這方法傳回含有指定的 URI 路徑（String uripath）的 ServletContext。其中的 uripath 必須是絕對路徑（以 / 開頭），也就是以伺服器的文件根目錄（document root）為路徑起點。這方法讓 servlet 可存取自身環境之外的其它環境（context）。在著重安全或分散式環境（distributed enviornment，參見第十二章《Enterprise Servlets 和 J2EE》），servlet container 可能不管你指定的 uripath 為何，一律傳回 null 值。

繼續沿用先前的洋春捲範例。假設有另外一家提供外食遞送服務的公司，他們網站上的 servlet 也需要存取食品公司的「今日特價品」資訊。假設食品公司的洋春捲應用系統環境（context）是以 /burritostore 為路徑起點，而且伺服器的安全防護控制也不禁止「環境間通訊」（intercontext communciation）；在這種情況下，任何 servlet 都可用以下程式碼取得該食品公司的「今日特價品」資訊：

```
ServletContext myContext = getServletContext();
ServletContext otherContext =
    myContext.getContext("/burritostore/index.html");
String burrito = otherContext.getAttribute("com.costena.special.burrito");
Date day = (Date)otherContext.getAttribute("com.costena.special.day");
```

目前只能透過 URI 路徑來取得不同環境（context）的參照指標，因此，如果該食品公司決定將洋春捲應用系統搬到別處，上述範例就跟著失效了。或許未來可以依據「名稱」來取得環境。

類別載入器的考量

由於「類別載入器」（class loader）的關係，若放入環境（context）中的是「自製物件」（custom object），將使得不同環境之間的物件共享變得很困難。原因是，每個 Web 應用系統的類別檔，都是放在自己的 WEB-INF 目錄下，因此，不同 Web 應用程式的類別檔，是各自以不同的 class loader 載入的；而來自某一應用程式的 class loader，無法找出存放在其它應用程式的類別；這導致一個不幸後果：若物件的 .class 檔只存在於 WEB-INF/classes 或 WEB-INF/lib 目錄中，則任何其它應用程式都不能夠輕易使用該物件；任何想將該物件鑄型（cast）成適當型態的嚐試，都將以 NoClassDefFoundError 悲劇收場。

將 .class 檔複製到其它應用程式的作法也不可行，因為由不同 class loader 所載入的類別，就算類別定義相同，也會被視為不同的類別。不信邪的人可以試試，這樣做只會換成另一種悲劇收場而已：ClassCastException。

這問題有三種可能的解法。首先，將共享物件的 .class 檔複製到伺服器的「標準類別路徑」（standard classpath），以 Tomcat 3.2 而言，這是指 *server_root*/classes/ 目錄，若是 Tomcat 3.3，此目錄為 *server_root*/lib/common/。這使得所有 Web 應用系統所共享的「根本類別載入器」（primordial class loader）可以找到共享物件的 .class 檔。第二種解法，是避免對傳回的物件實施「鑄型」（cast），而改用 reflection 技術（一種讓 Java 類別可在執行期自我檢視及管理的技術）來呼叫物件本身的方法；這種解法雖然不夠高竿，但是當你無權控管整個伺服環境時，卻是一個可行的辦法。第三種解法，是將傳回的物件鑄型成一個宣告了相關方法的界面，並且將該界面的 .class 檔放置於伺服器的標準類別路徑中。如此一來，除了那個界面之外，每個類別的 .class 檔皆可留在各自的 Web 應用系統。當然，這種作法需要“重新研擬”（refactoring）共享物件的類別，但如果需要保持 Web 應用系統的原本實作成品，這卻不失為一個可行之道。

11.2 控制權共用

對於更機動的合作模式，servlet 可以共享 request 的控制權。第一，servlet 可以轉遞（forward）整個 request、由其它 servlet 進行某些必要處理、然後再將 request 交給另一個元件。第二，servlet 可在自己 response 放入（include）一些其它元件產生的內容，基本上，這形同以程式手法達到 SSI（server-side include）的效果。在概念上，如果將結果網頁當成一個「畫面」（screen），那麼「forward」的動作可比喻成『將整個畫面交給另一個 servlet 全權處理』；而「include」的動作，就好像在畫面的某區域注入其它元件產生的內容。

這種分工能力讓 servlet 更有彈性、達成更理想的任務劃分。利用分工概念，servlet 可將它的 response 當成各種 Web 伺服元件所產生的結果之集合。對於 JavaServer Pages（JSP）而言，這種能力尤其重要。在 JSP，時常由一個 servlet 負責預先處理 request，然後將 request 交給另一個 JSP 網頁來完成 response（見第十八章《JavaServer Pages》）。

11.2.1 取得 RequestDispatcher

為了支援 request 的分工處理，Servlet API 2.1 版引進一個 `javax.servlet.RequestDispatcher` 界面。servlet 使用自己的 request 物件的 `getRequestDispatcher()` 方法來取得一個 `RequestDispatcher` 個體，所傳回的 `RequestDispatcher` 物件可用來將 request 分派給 `uripath` 所指定的其它元件（servlet、JSP、或靜態檔案）：

```
public RequestDispatcher ServletRequest.getRequestDispatcher(String uripath)
```

所指定的 `uripath` 可以是「相對路徑」，但不能擴展到目前 servlet context 之外。不過，你可利用先前討論過的 `getContext()` 方法，將 request 分派到目前環境之外 — 但僅限於同一伺服器內的環境，你不能將 request 交給另一個伺服器的環境。若指定的 `uripath` 是以 `/` 開頭，則會被解釋成『以目前環境的文件根目錄為路徑起點』。如果 `uripath` 包含查詢字串，則 request 會被加到受委任元件的參數集合的開頭。若有任何原因讓 servlet container 無法傳回 `RequestDispatcher` 物件，這方法會傳回 `null`。

有趣的是，在 ServletContext 類別中有一個同名方法：

```
public RequestDispatcher ServletContext.getRequestDispatcher(String path)
```

這兩個同名方法的差異，在於 ServletContext 類別的版本（於 API 2.1 版引進）只接絕對式 URL（以 / 開頭）；而 ServletRequest 類別的版本（於 API 2.2 版引進）則能接受相對式 URL 與絕對式 URL。因此，現在已經沒有理由繼續使用 ServletContext 的版本，它的存在只是為了保持回溯相容性，而且可以考慮棄置（deprecation）它（雖然未經正式宣佈）。

除了可用 URL 路徑來表示資源之外，你也可以在 Web 應用系統的佈署描述檔（web.xml）定義資源（指 servlet 與 JSP 網頁）的名稱（請參考第三章《Servlet 生命週期》）。既然如此，若你不想用 URL 來取得 RequestDispatcher 物件，可以改用 ServletContext 類別的 getNamedDispatcher() 方法，以資源名稱來取得 RequestDispatcher 物件：

```
public RequestDispatcher ServletContext.getNamedDispatcher(String name)
```

使用資源名稱的好處，讓我們可隱藏一些沒必要開放給大眾存取的資源；而 ServletContext.getNamedDispatcher() 讓我們可將 request 分派到這類資源。若有任何因素使得 context 無法傳回 RequestDispatcher 物件，這方法會傳回 null。

RequestDispatcher 提供了兩個方法，分別是 forward() 和 include()。前者（forward()）將 request 完整交給受委任的資源處理；後者（include()）將受委任資源的輸出加到 calling servlet 的 response，但仍將控制權保留在 calling servlet 自己。

11.2.2 委任式分工

forward() 方法可將 servlet 的 request 轉交給同伺服器上的另一個資源。利用這方法，servlet 可先對 request 做些必要的前置處理，然後由另一個資源產生 response。不同於 sendRedirect() 方法，forward() 是完全在伺服器內運作，用戶端不會知道伺服器內所發生的事。轉遞 request 時，有兩種辦法可將資訊傳遞給受委任者：附加一個查詢字串；或者使用 setAttribute() 方法設定一組申請屬性（request attribute）。範例 11-3 示範一個負責搜尋工作的 servlet，在它完成搜尋之後，它將搜尋結果轉交給另一個網頁來產生輸出畫面。

範例 11-3：搜尋引擎後端作業

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class SearchLogic extends HttpServlet {


  public void doGet(HttpServletRequest req, HttpServletResponse res)
                              throws ServletException, IOException {

    // 不需要設定 content type, 也用不到 writer

    // 取得要搜尋的字串
    String search = req.getParameter("search");

    // 計算含有搜尋字串的 URL
    String[] results = getResults(search);

    // 將搜尋結果當成一個申請屬性
    req.setAttribute("results", results);

    // 將 request 委任給負責產生畫面的資源
    String display = "/servlet/SearchView";
    RequestDispatcher dispatcher = req.getRequestDispatcher(display);
    dispatcher.forward(req, res);
  }


  // 實際使用這方法時, 應該要在這方法呼叫真正的搜尋引擎,
  // 並傳回更詳盡的搜尋結果, 而非只有 URL 而已
  String[] getResults(String search) {
    return new String[] { "http://www.abc.com",
                          "http://www.xyz.com" };
  }

}
```

此 servlet 的任務，是成為線上搜尋引擎的中樞。它搜尋 search 參數所指定的字串，然後將找到的 URL 儲存在 request 的 results 屬性中，然後將 request 轉交給展示元件來產生輸出畫面；在此，我們固定使用 /servlet/SearchView 這個展示元件，但是在實務上，應該可以依照使用者偏好的語言、網站色調、初階或進階顯示方式 ...等等資訊而改變展示元件的路徑（或參數）。

要將 request 委任出去的 servlet，必須遵守一些較嚴格的規則：

- 只能設定 HTTP header 及狀態碼，而不能送出任何 response body 到用戶端（那是include 的動作）。也就是說，只有在送出 response 之前，才能呼叫 forward()。
- 如果已送出 response，呼叫 forward() 時會發生 IllegalStateException。
- 如果尚未送出 response，但是應答緩衝區內有資料，則 forward() 的過程中會自動清掉緩衝區。
- 此外，你不能自作主張，替換新的 request 與 response 物件。你只能呼叫當初傳給 calling servlet 的服務方法（doGet() 和 doPost()）的 request 與 response 物件的 forward()；而且 forward() 只能在同一個 handler thread 內被呼叫。

受委任（接收 request）的元件，其寫法與任何其它元件的寫法一樣，如範例 11-4 所示：

範例 11-4：搜尋引擎前端作業

```
import java.io.*;

import javax.servlet.*;
import javax.servlet.http.*;


public class SearchView extends HttpServlet {

  public void doGet(HttpServletRequest req, HttpServletResponse res)
                              throws ServletException, IOException {
    res.setContentType("text/plain");
```

```
    PrintWriter out = res.getWriter();

    // Get the search results from a request attribute
    String[] results = (String[]) req.getAttribute("results");

    if (results == null) {
      out.println("No results.");
      out.println("Did you accidentally access this servlet directly?");
    }
    else {
      out.println("Results:");
      for (int i = 0; i < results.length; i++) {
        out.println(results[i]);
      }
    }

    out.println();
    out.println("Request URI: " + req.getRequestURI());
    out.println("Context Path: " + req.getContextPath());
    out.println("Servlet Path: " + req.getServletPath());
    out.println("Path Info: " + req.getPathInfo());
    out.println("Query String: " + req.getQueryString());
  }
}
```

如果你打算自己測試這個 servlet**，請注意到該** servlet **印出來的路徑資訊已經被調整過，它已不是原本的** request **資訊，而是用來取得** dispatcher **的路徑（【編註：也就是範例** 11-3 **的** Display **字串】）：**

```
Request URI: /servlet/SearchView
Context Path:
Servlet Path: /servlet/SearchView
Path Info: null
Query String: null
```

調整路徑的原因，在於當你將 request **轉交給另一個** servlet **時，受託的** servlet **所得到的路徑資訊，應該就像它被直接呼叫時所得到的路徑資訊一樣，從收到** request **的那一刻起，受託的** servlet **就有完整的控制權。這也就是為什麼在呼叫** forward() **之前，必須先清空應答緩衝區，而且不能送出任何** response**。**

使用名稱分派資源

使用“路徑”來指定委任對象，會遇到一個問題：如果目標元件是某路徑上的一個伺服元件，那麼，用戶端也能用同樣路徑取得該元件。為了安全起見，你可能會刻意讓元件無法由外界存取（比方說，取消 URL 中的 /servlet），不使用路徑而改用“名稱”來指定委任對象。你可利用 getNamedDispatcher() 來辦到這件事。在範例 11-3，其中用來指定委任對象的程式碼，可以改寫成這樣：

```
// Forward to a display page
String display = "searchView";
RequestDispatcher dispatcher = req.getNamedDispatcher(display);
dispatcher.forward(req, res);
```

使用 getNamedDispatcher() 時要注意一件事，由於不使用 URI 路徑，因此無法附加查詢字串，而且也不會調整路徑資訊。

11.2.3 「轉遞」（Forward）與「重導」（Redirect）的比較

到底應該使用 forward() 或是 sendRedirect()？這兩種方法其實各有用途。若有一個負責處理事務邏輯的元件，必須與其它元件共享處理結果時，使用 forward() 的效果最好。另一方面，若要將用戶端帶到另一個網頁，則最好使用 sendRedirect()。不過，對於簡單的重新導引，一般都傾向以 forward() 來代替 sendRedirect()。因為 forward() 的動作全在伺服器內完成，執行速度較快；而 sendRedirect() 卻需要與用戶端來回通訊。不幸的是，forward() 有一個致命傷：在處理相對式 URL 時會出問題。請參考範例 11-5，假設這個 servlet 的任務是將 request 轉交到該站台的首頁：

範例 11-5：直接回家，不繞路

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HomePageForward extends HttpServlet {
```

```
    public void doGet(HttpServletRequest req, HttpServletResponse res)

                                throws ServletException, IOException {
      RequestDispatcher dispatcher = req.getRequestDispatcher("/index.html");
      dispatcher.forward(req, res);
    }
  }
```

如果將這個 servlet 擺放到 Tomcat 伺服器，然後在用戶端以 http://server:8080/servlet/HomePageForward 來存取它，你將發現 index.html 的所有影像連結都斷了。這是因為 index.html 網頁內的 <IMG> 標籤以「相對路徑」來指定影像檔的位置，而用戶端瀏覽器並不知道路徑起點已經改變了，仍舊以 /servlet/HomePageForward/ 為路徑起點，所以才會造成所有相對路徑全錯的狀況。另一方面，若是使用 sendRedirect()，由於這會讓伺服器對用戶端送出一個 HTTP Redirect 訊息，讓用戶端自己向新網頁發出 request，因此瀏覽器能知道正確的路徑起點。除此之外，若要將 request 委任給其它環境（context）下的資源，使用 sendRedirect() 也比較容易些，因為這不需要到 getContext() 方法。因此，我們的建議是，如果可能，請儘量使用 sendRedirect()；在必要狀況下才使用 forward()。

11.2.4 調用式分工

RequestDispatcher 物件的 include() 方法可將其它資源的內容“引入”目前的 response 中，其效果就像是用程式手段達成的 SSI（server-side include）。相較於 forward() 方法，呼叫 include() 的 servlet 並不會交出 response 的控制權，而且還可在調用來的內容前後加上自己的內容。範例 11-6 示範 servlet 如何將來自其它資源的項目引入目前的 response：

範例 11-6：引入一個項目

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class NileBooks extends HttpServlet {
```

範例 11-6：引入一個項目（續）

```
    public void doGet(HttpServletRequest req, HttpServletResponse res)
                                throws ServletException, IOException {
      res.setContentType("text/html");
      PrintWriter out = res.getWriter();

      out.println("<HTML><HEAD><TITLE>歡迎來到尼羅河網路書店</TITLE></HEAD>");
      out.println("<BODY>");

      // 顯示在線上型錄中的物品
      out.println("沒得選，僅此一本：");

      RequestDispatcher dispatcher =
        req.getRequestDispatcher("/servlet/NileItem?item=0596000405");
      dispatcher.include(req, res);

      out.println("小子，看你順眼，這次算你八折就好！");

      out.println("</BODY></HTML>");
    }
  }
```

如同 forward()，使用 include() 時，也可利用「查詢字串」或使用 setAttribute() 設定「request 屬性」，將資訊傳給受呼叫的資源；使用 request 屬性的好處是可以傳遞物件，若使用「查詢字串」參數，就只能夠傳遞字串。範例 11-7 示範如何利用 setAttribute() 讓資源傳回我們要的內容：

範例 11-7：引入多個項目

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class NileBooks extends HttpServlet {

  public void doGet(HttpServletRequest req, HttpServletResponse res)
                                throws ServletException, IOException {
```

```
    res.setContentType("text/html");
    PrintWriter out = res.getWriter();

    out.println("<HTML><HEAD><TITLE>歡迎來到尼羅河網路書店</TITLE></HEAD>");
    out.println("<BODY>");

    // 顯示在線上型錄中的物品
    RequestDispatcher dispatcher =
      req.getRequestDispatcher("/servlet/NileItem");

    out.println("就這一本, 如何？");
    req.setAttribute("item", Book.getBook("0596000405"));
    dispatcher.include(req, res);

    //  移除使用過的 item
    req.removeAttribute("item");

    out.println("或者, 這本怎麼樣？");
    req.setAttribute("item", Book.getBook("0395282659"));
    dispatcher.include(req, res);

    out.println("小子, 你的眼光實在不錯, 一律算八折！");

    out.println("</BODY></HTML>");
  }
}

// 一個簡單的 Book 類別
class Book {
  String isbn;
  String title;
  String author;

  private static Book JSERVLET =
    new Book("0596000405", "Java Servlet 程式設計", "Hunter");

  private static Book HOBBIT =
    new Book("0395282659", "The Hobbit", "Tolkien");
```

範例 11-7：引入多個項目（續）

```
    // 模擬一次資料庫查詢
    public static Book getBook(String isbn) {
      if (JSERVLET.getISBN().equals(isbn)) {
        return JSERVLET;
      }
      else if (HOBBIT.getISBN().equals(isbn)) {
        return HOBBIT;
      }
      else {
        return null;
      }
    }

    private Book(String isbn, String title, String author) {
      this.isbn = isbn;
      this.title = title;
      this.author = author;
    }

    public String getISBN() {
      return isbn;
    }

    public String getTitle() {
      return title;
    }

    public String getAuthor() {
      return author;
    }
  }
```

在這裡，servlet 傳遞的是完整的 Book 物件，而不只是 ISBN 字串（國際書號）。在實際應用上，書籍資訊應該是來自資料庫（通常是以 ISBN 為查詢條件），不應該像本例的作法，將這些資料寫死在程式碼裡。

NileItem servlet 可以呼叫 req.getAttribute("item") 來取得 item 屬性，如範例 11-8 所示：

範例 11-8：顯示一個項目

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class NileItem extends HttpServlet {

  public void doGet(HttpServletRequest req, HttpServletResponse res)
                              throws ServletException, IOException {

    // 不設定內容型態 (content type)

    PrintWriter out = res.getWriter();

    Book book = (Book) req.getAttribute("item");

    out.println("<BR>");
    if (book != null) {

      out.println("<I>" + book.getTitle() + "</I>");
      out.println(" by " + book.getAuthor());
    }
    else {
      out.println("<I>沒找到書</I>");
    }
    out.println("<BR>");
  }
}
```

被引用的 servlet，因為沒有能力修改狀態碼，也不能變更隨 response 送出的 HTTP header（就算嘗試改變也沒有效果），所以其輸出內容可放在網頁的任何位置。

不同於 forward()，在被引用的 servlet 裡，request 的路徑資訊與參數並不會改變，而會與 calling servlet 一致。若被引用 servlet 需要得到它自身的路徑資訊與參數，就只能透過下列由伺服器指定的屬性來取得：

```
javax.servlet.include.request_uri
javax.servlet.include.context_path
javax.servlet.include.servlet_path
javax.servlet.include.path_info
javax.servlet.include.query_string
```

傳給 include() 方法的 request 及 response 參數，必須是 calling servlet 的服務方法（doGet() 及 doPost()）所收到的物件相同，而且必須在同一個 handler thread 內呼叫 include()。

最後，呼叫方與被呼叫方所用的輸出機制必須一致。換句話說，如果呼叫方使用 PrintWriter，則被引入的資源也應該使用 PrintWriter；或呼叫方使用的是 OutputStream，則被引入的資源也同樣要使用 OutputStream。如果兩方所用的輸出機制不同，則被引入的 servlet 會發生 IllegalStateException 異常。如果可能，所有文字輸出都應該採用 PrintWriter，而這應該不是什麼問題。