2.1 第一個 Servlet

從這章開始會正式學習 Servlet/JSP，如果想紮穩基礎的話，要先從 Servlet 開始了解，正如第 1 章談過的，JSP 終究會轉譯為 Servlet，了解 Servlet，JSP 也就學了一半了，而且不會被看似奇怪的 JSP 錯誤搞得糊理糊塗。

首先要準備開發環境，使用 Apache Tomcat 作為容器，本書除了介紹 Servlet/JSP 之外，也會一併介紹整合開發環境（Integrated Development Environment）的使用，簡稱 IDE，畢竟在了解 Servlet/JSP 的原理與撰寫之外，了解如何善用 IDE 這樣的產能工具來增進程式撰寫效率也是必要的，也能符合業界需求。

2.1.1 準備開發環境

第 1 章曾經談過，抽象層面來說，Web 容器是 Servlet/JSP 唯一認得的 HTTP 伺服器，開發工具的準備中，自然就要有 Web 容器的存在，本書使用 Apache Tomcat 9 作為 Web 容器，這是支援 Servlet 4.0 的版本，可以在這邊下載：

- tomcat.apache.org/download-90.cgi

在這邊建議下載頁面中的 Core 版本，如果是在 Windows 環境中，**請留意 Windows 版本是 32-bit 或是 64-bit**，本書的範例環境是 Windows 10 64-bit 版本，因此會使用下載頁面中的「64-bit Windows zip」。

在第 1 章中看過這張圖：

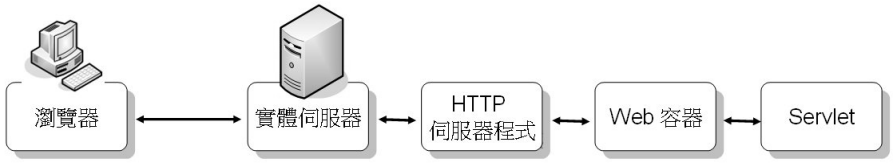

圖 2.1 從請求到 Servlet 處理的線性關係

Tomcat 提供的主要是 Web 容器的功能，而不是 HTTP 伺服器的功能，然而為了給開發者便利，下載的 Tomcat 會附帶一個簡單的 HTTP 伺服器，相較於真正的 HTTP 伺服器而言，Tomcat 附帶的 HTTP 伺服器功能太過簡單，僅作開發用途，不建議日後直接上線服務。

接著準備 IDE，本書會使用 Eclipse，這是業界普遍採用的 IDE 之一，可以在這邊下載：

- www.eclipse.org/downloads/eclipse-packages/

請下載頁面中的 Eclipse IDE for Java EE Developers，同樣地，若是 Windows 作業系統，請確定是 32-bit 或是 64-bit，本書會下載基於 Eclipse OXYGEN.2 Release (4.7.2)的 Eclipse IDE for Java EE Developers 之 64-bit 版本。

當然，必須有 Java 執行環境，**Java EE 8 搭配的版本為 Java SE 8**，如果還沒安裝，可以在這邊下載：

- www.oracle.com/technetwork/java/javase/downloads/

撰寫本書的這個時間點上，Java SE 的最新版本其實是 9，然而，Java SE 9 的重大特性之一為模組化平臺，JDK9/JRE9 為此也有著重大變動，有些開放原始碼專案針，必須對 Java SE 9 的模組化設計進行更新，才能順利於 Java SE 9 上執行，**為了避免相容性的困擾，建議仍是下載 Java SE 8 進行安裝**。

提示 »» 本書中會運用到 JDK7/8 的一些新特性，像是 NIO2、Lambda 語法、Stream、新日期時間 API 等，這些特性可以參考《Java SE 9 技術手冊》，該書也包含了 Java SE 9 模組化平臺的介紹。

底下總結本書目前使用到的基礎環境：

- Java SE 8
- Eclipse IDE for Java EE Developers
- Tomcat 9

本書預設你已經有 Java SE 的基礎，有能力自行安裝 JDK8，如果願意，可以配合本書的環境配置，本書製作範例時，將 Eclipse 與 Tomcat 都解壓縮在 C:\workspace 中，如下圖所示：

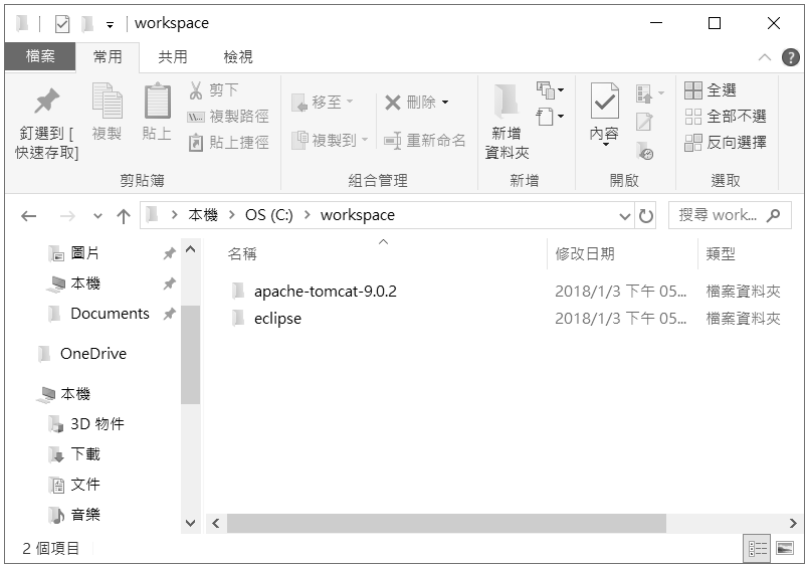

圖 2.2 範例基本環境配置

注意》》 如果想放在別的資料夾中，請不要放在有中文或空白字元的資料夾中，Eclipse 或 Tomcat 對此會有點感冒。

接著要在 Eclipse 中新增 Tomcat 為伺服器執行時期環境（Server Runtime Environments），以便之後開發的 Servlet/JSP 可執行於 Tomcat 實現的 Web 容器上。請按照以下步驟執行：

Lab

1. 執行 eclipse 資料夾中的 eclipse.exe。
2. 出現「Eclipse Launcher」對話方塊時，將「Workspace:」設定為「C:\workspace」，按下「Launch」。
3. 執行選單「Window/Preferences」，在出現的「Preferences」對話方塊中，展開左邊的「Server」節點，並選擇其中的「Runtime Environment」節點。
4. 按下右邊「Server Runtime Environments」中的「Add」按鈕，在出現的「New Server Runtime Environment」中選擇「Apache Tomcat v9.0」，按下「Next」按鈕。
5. 按下「Tomcat installation directory」旁的「Browse」，選取 C:\workspace 中解壓縮的 Tomcat 資料夾，按下「確定」。

6. 在按下「Finish」按鈕後，應該會看到以下的畫面，按下「Apply and Close」完成配置：

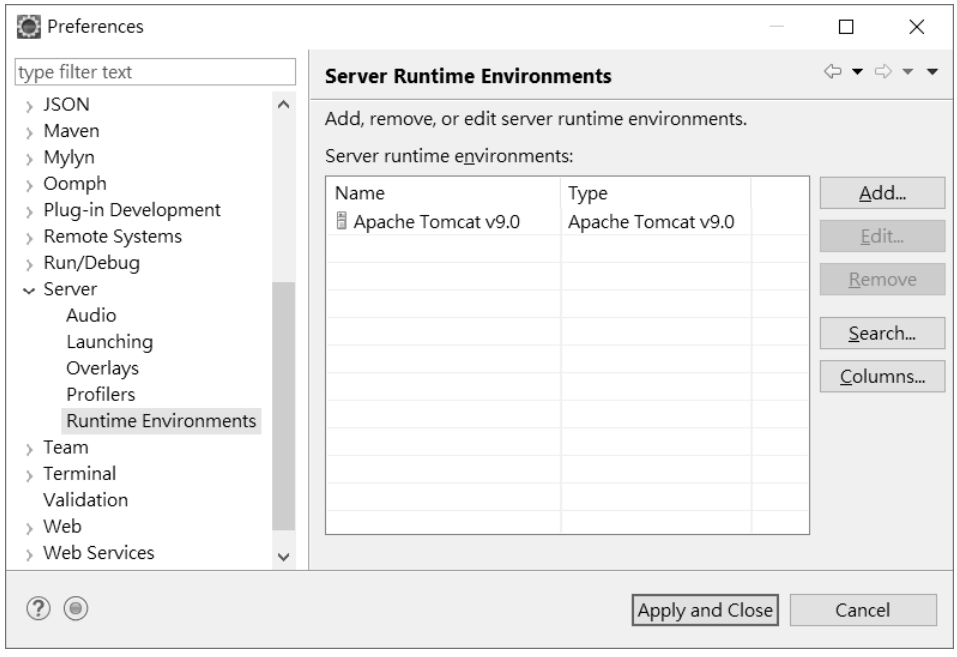

圖 2.3 新增 Tomcat 為伺服器執行時期環境

接著要配置工作區（Workspace）預設的文字檔案編碼，在沒有進一步設定的情況下，Eclipse 會使用作業系統預設的文字檔案編碼，在 Windows 上就是 MS950，在這邊建議使用 UTF-8，除此之外，CSS、HTML、JSP 等相關編碼設定，也建議都設為 UTF-8，這可以避免日後遇到一些編碼處理上的問題。請按照以下的步驟進行：

1. 執行選單「Window/Preferences」，在出現的「Preferences」對話方塊中，展開左邊的「General/Workspace」節點。
2. 在右邊的「Text file encoding」選擇「Other」，在下拉選單中選擇「UTF-8」，按下「Apply」按鈕。
3. 展開左邊的「Web」節點，選擇「CSS Files」，在右邊的「Encoding」選擇「UTF-8」，按下「Apply」按鈕。

4. 選擇「HTML Files」，在右邊的「Encoding」選擇「UTF-8」，按下「Apply」按鈕。
5. 選擇「JSP Files」，在右邊的「Encoding」選擇「UTF-8」，按下「Apply」按鈕。
6. 按下「Preferences」對話方塊的「Apply and Close」完成設定。

2.1.2 第一個 Servlet 程式

接著可以開始撰寫第一個 Servlet 程式了，程式將使用 Servlet 接收使用者名稱並顯示招呼語。由於 IDE 是產能工具，會使用專案來管理應用程式相關資源，在 Eclipse 中第一步是建立「Dynamic Web Project」，之後在專案中建立第一個 Servlet。請按照以下步驟進行操作：

1. 執行選單「File/New/Dynamic Web Project」，在出現的「New Dynamic Web Project」對話方塊中，輸入「Project name」為「FirstServlet」。
2. 確定「Target runtime」為方才設定的「Apache Tomcat v9.0」，按下「Finish」按鈕。
3. 展開新建專案中的「Java Resources」節點，在「src」上按右鍵，執行「New/Servlet」。
4. 在「Create Servlet」對話方塊的「Java package」輸入「cc.openhome」，「Class name」輸入「Hello」，按下「Next」按鈕。
5. 選擇「URL mappings」中的「Hello」，按右邊的「Edit」按鈕並改為「/hello」後，按下「OK」按鈕。
6. 按下「Create Servlet」的「Finish」按鈕。

在第 2 個步驟中有個「Dynamic web module version」設定，撰寫本書的這個時間點上，僅支援至 3.1，這並不妨礙本書範例之進行（除了自動產生的 web.xml 要做點修改外，之後會看到），未來 Eclipse 應該會有支援 4.0 的版本。

接著就可以來撰寫第一個 Servlet 了，在建立的「Hello.java」中，編輯以下的內容：

FirstServlet Hello.java

```
package cc.openhome;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/hello")
public class Hello extends HttpServlet {      ← ❶ 繼承 HttpServlet
    @Override
    protected void doGet(      ← ❷ 重新定義 doGet()
            HttpServletRequest request, HttpServletResponse response)
                throws ServletException, IOException {
                                                          ❸ 設定回應內容類型
        response.setContentType("text/html;charset=UTF-8");

        String name = request.getParameter("name");      ← ❹ 取得請求參數
                                                          "
        PrintWriter out = response.getWriter();      ← ❺ 取得回應輸出物件
        out.print("<!DOCTYPE html>");
        out.print("<html>");
        out.print("<head>");
        out.print("<title>Hello</title>");
        out.print("</head>");
        out.print("<body>");
        out.printf("<h1> Hello! %s!%n</h1>", name);      ← ❻ 跟使用者說 Hello!
        out.print("</body>");
        out.print("</html>");
    }
}
```

範例中繼承了 **HttpServlet**❶，並重新定義了 **doGet()**方法❷，當瀏覽器使用 GET 方法發送請求時，會呼叫此方法。

在 doGet()方法上可以看到 **HttpServletRequest** 與 **HttpServletResponse** 兩個參數，容器接收到瀏覽器的 HTTP 請求後，會收集 HTTP 請求中的資訊，並分別建立代表請求與回應的 Java 物件，而後在呼叫 doGet()時，將這兩個物件當作參數傳入。可以從 HttpServletRequest 物件中取得有關 HTTP 請求相關資訊，在範例中是透過 HttpServletRequest 的 **getParameter()**並指定請求參數名稱，來取得使用者發送的請求參數值❹。

由於 `HttpServletResponse` 物件代表對瀏覽器的回應，因此可以藉由其 **`setContentType()`** 設定正確的內容類型❸，範例中是告知瀏覽器，回應要以 `text/html` 解析，而採用的字元編碼是 `UTF-8`。接著使用 `getWriter()` 方法取得代表回應輸出的 **`PrintWriter`** 物件❺，藉由 `PrintWriter` 的 `println()` 方法來對瀏覽器輸出回應的文字資訊，在範例中是輸出 HTML 以及根據使用者名稱說聲 Hello! ❻。

提示 »» 在 Servlet 的 Java 程式碼中，以字串輸出 HTML，當然是很笨的行為，別擔心，在談到 JSP 時，會有個有趣的練習，將 Serlvet 轉為 JSP，從中明瞭 Servlet 與 JSP 的對應。

接著要來執行 Servlet 了，瀏覽器要對這個 Servlet 進行請求，同時附上請求參數。請按照以下的步驟進行：

1. 在「Hello.java」上按右鍵，執行「Run As/Run on Server」。
2. 在「Run on Server」對話方塊中，確定「Server runtime environment」為先前設定的「Apache Tomcat v9.0」，按下「Finish」按鈕。
3. 在 Tomcat 啟動後，Eclipse 也會啟動內嵌的瀏覽器，接著可以使用底下網址來進行請求：

   `http://localhost:8080/FirstServlet/hello?name=Justin`

如上操作之後，就會看到以下的畫面：

圖 2.4 第一個 Servlet 程式

提示 »» Eclipse 內建的瀏覽器功能其實陽春，可以執行選單「Window/Web Browser」來選擇「Run on Server」時啟動的瀏覽器。

Tomcat 預設會使用 8080 埠號，注意到網址列中，請求的 Web 應用程式路徑是 FirstServlet 嗎？預設專案名稱就是 Web 應用程式環境路徑（Context root），那為何請求的 URI 必須是/hello 呢？記得 Hello.java 中有這麼一行嗎？

```
@WebServlet("/hello")
```

這表示，如果請求的 URI 是/hello，就會由 Hello 來處理請求，關於 Servlet 的設定，還有更多的細節，事實上，由於到目前為止，借助了 IDE 的輔助，有許多細節都被省略了，接下來得先來討論這些細節。

2.2　在 Hello 之後

在 IDE 中撰寫了 Hello，並成功執行出應有的結果，那這一切是如何串起來的，IDE 又代勞了哪些事情？你在 IDE 的專案管理中看到的檔案組織結構真的是應用程式上傳之後的結構嗎？

記得！Web 容器是 Servlet/JSP 唯一認得的 HTTP 伺服器，你要了解 Web 容器會讀取哪些設定？又要求什麼樣的檔案組織結構？Web 容器對於請求到來，又會如何呼叫 Servlet？IDE 很方便，但不要過份依賴 IDE！

2.2.1　關於 HttpServlet

注意到 Hello.java 中 import 的語句區段：

```
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

如果要編譯 Hello.java，類別路徑（Classpath）中必須包括 Servlet API 的相關類別，如果使用的是 Tomcat，這些類別會封裝在 Tomcat 資料夾的 lib 資料夾中的 **servlet-api.jar**。假設 Hello.java 位於 src 資料夾下，並放置於對應套件的資料夾之中，則可以如下進行編譯：

```
% cd YourWorkspace/FirstServlet
% javac -classpath Yourlibrary/YourTomcat/lib/servlet-api.jar -d ./classes
src/cc/openhome/Hello.java
```

底線部份必須修改為實際的資料夾位置，編譯出的.class 檔案會出現於 classes 資料夾中，並有對應的套件階層（因為使用 `javac` 時下了`-d` 引數）。事實上，如果遵照 2.1 節的操作，Eclipse 就會自動完成類別路徑設定，以及在存檔時嘗試編譯等細節，展開「Project Explorer」中的「Libraries/Apache Tomcat v9.0」節點，就會看到相關 JAR（Java ARchive）檔案的類別路徑設定。

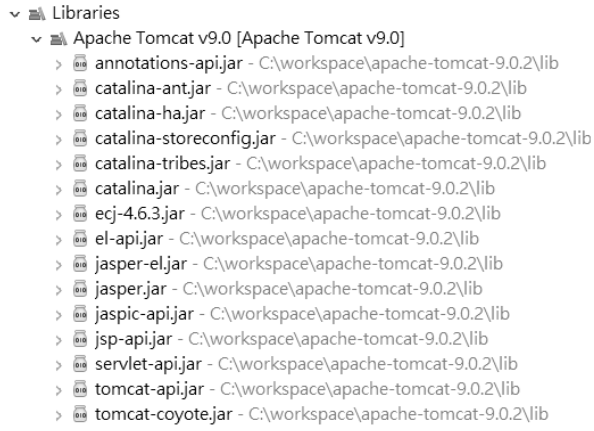

圖 2.5 IDE 會為你設定專案的類別路徑

為什麼要在繼承 `HttpServlet` 之後重新定義 `doGet()`？又為什麼 HTTP 請求為 `GET` 時會自動呼叫 `doGet()`呢？首先來討論範例中看到的相關 API 架構圖：

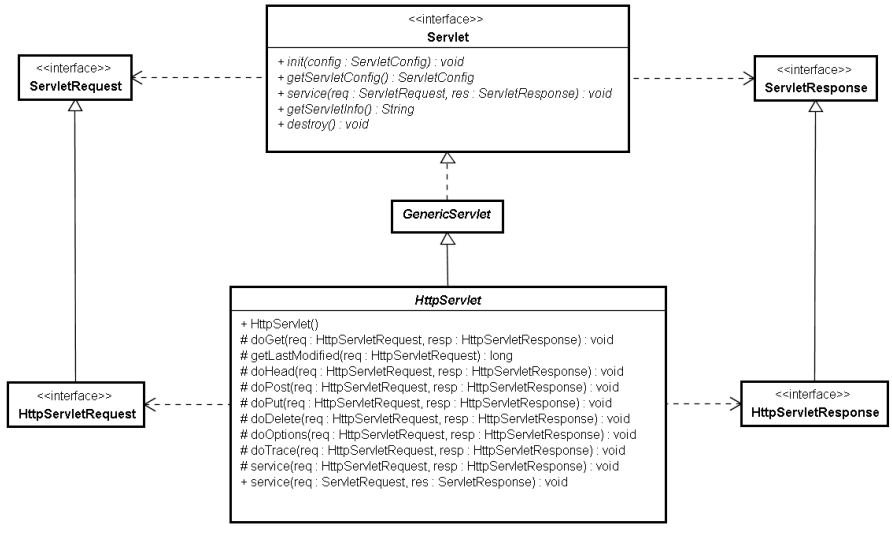

圖 2.6 `HttpServlet` 相關 API 類別圖

首先看到 **Servlet** 介面，它定義了 Servlet 的基本行為，例如與 Servlet 生命週期相關的 **init()**、**destroy()** 方法、提供服務時要呼叫的 **service()** 方法等。

實作 Servlet 介面的類別是 GenericServlet 類別，它還實作了 ServletConfig 介面，將容器呼叫 init() 方法時傳入的 ServletConfig 實例封裝起來，而 service() 方法直接標示為 abstract 而沒有任何實作。在本章中將暫且忽略 GenericServlet（第 5 章會加以討論）。

在這邊只要先注意到一件事，GenericServlet 沒有規範任何有關 HTTP 的相關方法，而是由繼承它的 HttpServlet 定義。在最初定義 Servlet 時，並不限定它只能用於 HTTP，所以沒有將 HTTP 相關服務流程定義在 GenericServlet 之中，而是定義在 HttpServlet 的 service() 方法中。

提示 »» 可以注意到套件的設計，與 Servlet 定義相關的類別或介面都位於 javax.servlet 套件之中，像是 Servlet、GenericServlet、ServletRequest、ServletResponse 等。而與 HTTP 定義相關的類別或介面都位於 javax.servlet.http 套件之中，像是 HttpServlet、HttpServletRequest、HttpServletResponse 等。

HttpServlet 的 service() 方法中的流程大致如下：

```
protected void service(HttpServletRequest req,
                       HttpServletResponse resp)
    throws ServletException, IOException {
    String method = req.getMethod(); // 取得請求的方法
    if (method.equals(METHOD_GET)) { // HTTP GET
        // 略...
        doGet(req, resp);
        // 略 ...
    } else if (method.equals(METHOD_HEAD)) { // HTTP HEAD
        // 略 ...
        doHead(req, resp);
    } else if (method.equals(METHOD_POST)) { // HTTP POST
        // 略 ...
        doPost(req, resp);
    } else if (method.equals(METHOD_PUT)) { // HTTP PUT
        // 略 ...
}
```

當請求來到時，容器會呼叫 Servlet 的 service() 方法，而可以看到，HttpServlet 的 service() 中定義的，基本上就是判斷 HTTP 請求的方式，再分別呼叫 doGet()、doPost() 等方法，若想針對 GET、POST 等方法進行處理，才會在繼承 HttpServlet 之後，重新定義相對應的 doGet()、doPost() 方法。

注意》》 這其實是使用了設計模式（Design Pattern）中的 Template Method 模式。不建議也不應該在繼承了 `HttpServlet` 之後，重新定義 `service()` 方法，這會覆蓋掉 `HttpServlet` 中定義的 HTTP 預設處理流程。

2.2.2 使用@WebServlet

撰寫好 Servlet 之後，接下來要告訴 Web 容器有關於這個 Servlet 的一些資訊。自 Java EE 6 的 Servlet 3.0 之後，可以使用標註（Annotation）來告知容器哪些 Servlet 會提供服務以及額外資訊。例如在先前的 Hello.java 中，就有底下的標註：

```
@WebServlet("/hello")
public class Hello extends HttpServlet {
```

只要 Servlet 上有設定**`@WebServlet`**標註，容器就會自動讀取當中的資訊。上面的`@WebServlet`告訴容器，若請求的 URI 是`/hello`，就由 `Hello` 的實例提供服務。

標註`@WebServlet`時可以提供供更多資訊，例如，修改先前的 Hello.java 如下：

FirstServlet Hello.java

```
package cc.openhome;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet(
    name="Hello",
    urlPatterns={"/hello"},
    loadOnStartup=1
)
public class Hello extends HttpServlet {
    ...略
}
```

上面的`@WebServlet`告知容器，`Hello`這個 Servlet 的名稱是 Hello，這是由 **`name`** 屬性指定，若瀏覽器請求的 URI 是`/hello`，就由這個 Servlet 來處理，這是由 **`urlPatterns`** 屬性來指定。在 Java 應用程式中使用標註時，沒有設定的屬性通常會有預設值，例如，**若沒有設定`@WebServlet`的 `name` 屬性時，預設值會是 Servlet 的類別完整名稱**。

當應用程式啟動後，並沒有建立所有的 Servle 實例。容器會在首次接到請求需要某個 Servlet 服務時，才將對應的 Servlet 類別實例化、進行初始動作，接著再處理請求。這意謂著第一次請求該 Servlet 的瀏覽器，必須等待 Servlet 類別實例化、進行初始動作之後，才真正得到請求的處理。

如果希望應用程式啟動時，可先將 Servlet 類別載入、實例化並作好初始化動作，可以使用 **`loadOnStartup`** 設定，設定大於 `0` 的值（預設值`-1`），表示啟動應用程式後就初始化 Servlet（而不是實例化幾個 Servlet）。**數字代表了 Servlet 的初始順序，容器必須保證有較小數字的 Servlet 先初始化**，在使用標註的情況下，如果有多個 Servlet 在設定 `loadOnStartup` 時使用了相同的數字，容器實作廠商可以自行決定要如何載入哪個 Servlet。

2.2.3　使用 web.xml

使用標註來定義 Servlet，是從 Java EE 6 的 Servlet 3.0 之後才有的功能，目的是在簡化設定，在舊的 Servlet 版本中，必須於 Web 應用程式的 WEB-INF 資料夾中，建立 **web.xml** 檔案定義 Servlet 相關資訊，當然，就算可以使用標註，必要的時候，仍然能使用 web.xml 檔案來定義 Servlet。

例如，可以在先前的 FirstServlet 專案的「Project Explorer」中，找到「Deployment Descriptor:FirstServlet」節點，按右鍵執行「Generate Deployment Descriptor Stub」，這會在「WebContent/WEB-INF」節點中，建立一個 web.xml 檔案，就本書使用的 Eclipse 版本來說，會有以下的預設內容：

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd" version="3.1">
  <display-name>FirstServlet</display-name>
```

```
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>default.html</welcome-file>
    <welcome-file>default.htm</welcome-file>
    <welcome-file>default.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

像這樣的檔案稱之為**部署描述檔（Deployment Descriptor，有人簡稱 DD 檔）**。由於先前「Dynamic web module version」設定是採用 3.1，因此自動產生的 web.xml 在 XSD 檔案及版本設定上預設為 3.1 版本，**在 Servlet 4.0 中，XSD 檔案應該是 `web-app_4_0.xsd`，而 `version` 會是 `4.0`**。

提示 »» 可以在〈Java EE: XML Schemas for Java EE Deployment Descriptors〉[1]中找到各版本的XML Shema。

在產生的 web.xml 中，可以看到`<display-name>`，這定義了 Web 應用程式的名稱，若工具程式有支援，可以採用此名稱來代表 Web 應用程式，**`<display-name>` 不是 Web 應用程式環境根目錄**，在 Servlet 4.0 之前，並沒有規範如何定義 Web 應用程式環境根目錄，因而各廠商實作會有各自定義的方式。

提示 »» Tomcat可以在META-INF/context.xml中設定環境根目錄，可參考〈The Context Container〉[2]。

Tomcat 預設會使用應用程式資料夾作為環境根目錄，在 Eclipse 中，也可以於專案上按右鍵，執行「Properties」，在「Web Project Settings」裏設定環境根目錄：

[1] Java EE: XML Schemas for Java EE Deployment Descriptors：www.oracle.com/webfolder/technetwork/jsc/xml/ns/javaee/index.html

[2] The Context Container：tomcat.apache.org/tomcat-9.0-doc/config/context.html

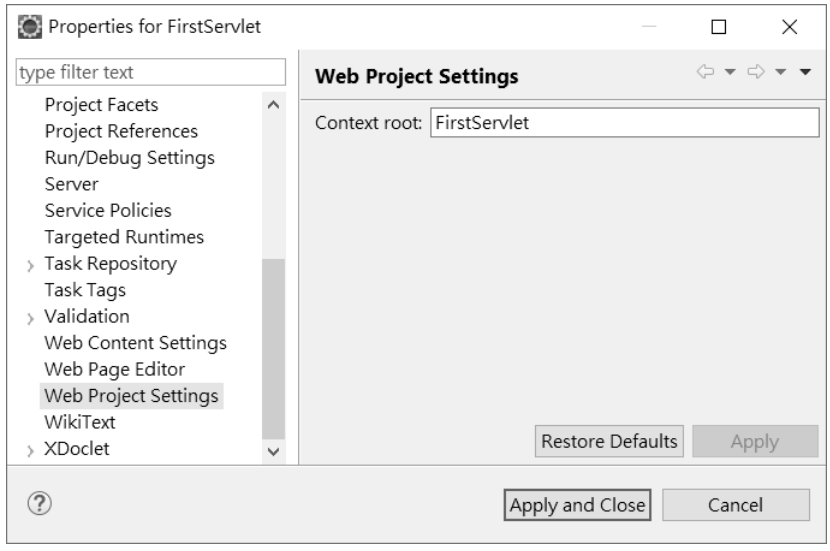

圖 2.7 Eclipse 中設定環境根目錄

從 Servlet 4.0 開始，可以在 web.xml 中使用`<default-context-path>`來建議預設環境路徑，然而考量到既有容器實作的相容性，容器實作廠商可以不理會這個設定。

至於`<welcome-file-list>`定義的檔案清單，是在瀏覽器請求路徑沒有指定特定檔案時，會看看路徑中是否有清單中的檔案，如果有的話，就會作為預設頁面回應。

除了定義整個 Web 應用程式必要的資訊之外，**web.xml 中的設定可用來覆蓋 Servlet 中的標註設定**，因此可以使用標註來作預設值，而 web.xml 作為日後更改設定值之用，例如：

FirstServlet web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd" version="4.0">
        ...略
    <servlet>
        <servlet-name>Hello</servlet-name>
        <servlet-class>cc.openhome.Hello</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
```

```
    <servlet-mapping>
        <servlet-name>Hello</servlet-name>
        <url-pattern>/helloUser</url-pattern>
    </servlet-mapping>
</web-app>
```

在上例中，若有瀏覽器請求/helloUser，就由 Hello 這個 Servlet 來處理，這分別是由**<servlet-mapping>**中的**<url-pattern>**與**<servlet-name>**來定義，而 Hello 名稱的 Servlet，實際上是 cc.openhome.Hello 類別的實例，這分別是由**<servlet>**中的**<servlet-name>**與**<servlet-class>**來定義。

如果有多個 Servlet 在設定<load-on-startup>時使用了相同的數字，會依照在 web.xml 中設定的順序來初始 Servlet。

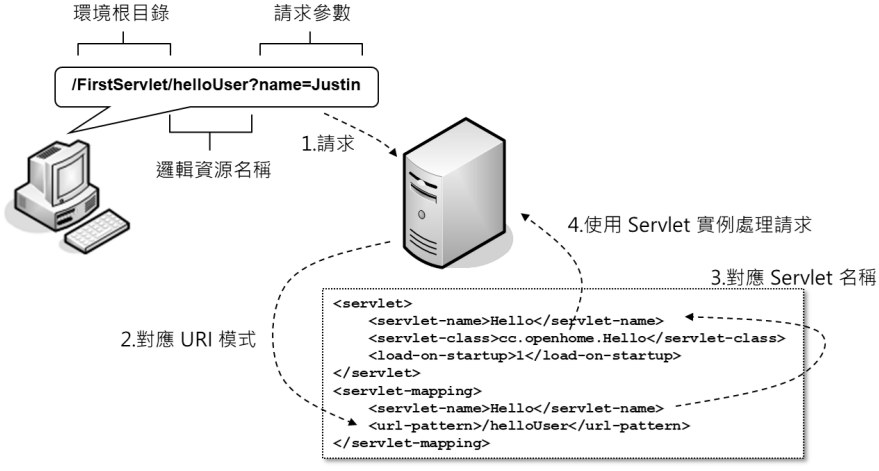

圖 2.8 Servlet 的請求對應

由於 web.xml 中<servlet-name>設定的名稱也是 Hello，與 Hello.java 中的 @WebServlet 標註的 name 屬性設定值相同，**在 Servlet 名稱相同的情況下，web.xml 中的 Servlet 設定會覆蓋@WebServlet 標註設定**，現在必須使用/helloUser（而不是使用/hello）請求 Servlet 了。

無論是使用@WebServlet 標註，或是使用 web.xml 設定，瀏覽器請求的 URI 都只是個邏輯上的名稱（Logical Name），請求/hello 並不一定指 Web 網站上有個實體檔案叫 hello，而會由 Web 容器來對應至實際處理請求的程式實體名

稱（Physical Name）或檔案。如果願意，也可以用個像 hello.view 或甚至 hello.jsp 之類的名稱來偽裝資源。

目前為止可以知道， Servlet 在 web.xml 會有三個名稱設定：`<url-pattern>` 設定的邏輯名稱、`<servlet-name>`註冊的 Servlet 名稱、以及`<servlet-class>`設定的實體類別名稱。

注意》》 除了可將`@WebServlet` 的設定當預設值，web.xml 用來覆蓋預設值之外，想一下，在 Servlet 3.0 之前，只能使用 web.xml 設定時的問題：寫好了一個 Servlet 並編譯完成，現在要寄給同事或客戶，還得說明如何在 web.xml 設定。在 Servlet 3.0 之後，只要使用`@WebServlet` 設定標註資訊，寄給同事或客戶後，他就只要將編譯好的 Servlet 放到 WEB-INF/classes 資料夾中就可以了（稍後就會談到這個資料夾），部署上簡化了許多。

2.2.4　檔案組織與部署

IDE 為了管理專案資源，會有其專案專屬的檔案組織，那並不是真正上傳至 Web 容器之後該有的架構，Web 容器要求應用程式部署時，必須遵照以下結構：

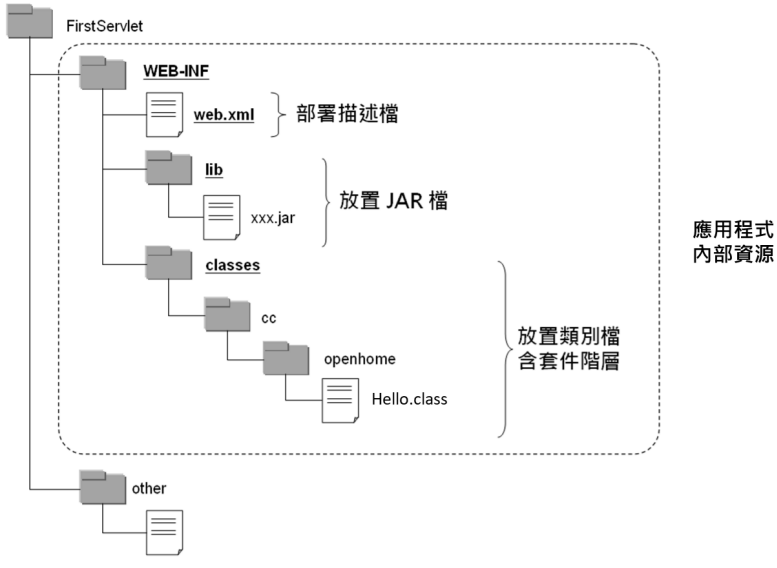

圖 2.9 Web 應用程式檔案組織

上圖有幾個重要的資料夾與檔案位置必須說明：

- WEB-INF

  這個資料夾名稱是固定的，而且一定是位於應用程式根資料夾下，放置在 WEB-INF 中的檔案或資料夾，對外界來說是封閉的，也就是瀏覽器無法使用 HTTP 的任何方法「直接」請求 WEB-INF 中的檔案或資料夾。若有這類需要，必須透過 Servlet/JSP 的請求轉發（Forward），不想讓瀏覽器直接存取的資源，可以放置在這個資料夾下。

- web.xml

  Web 應用程式部署描述檔，一定要放在 WEB-INF 資料夾。

- lib

  放置 JAR（Java Archive）檔案的資料夾，一定是位於 WEB-INF 資料夾之中。

- classes

  放置編譯過後.class 檔案的資料夾，放在 WEB-INF 資料夾，編譯過後的類別檔案，必須有與套件名稱相符的資料夾結構。

如果使用 Tomcat 作為 Web 容器，可以將符合圖 2.9 的 FirstServlet 整個資料夾複製至 Tomcat 資料夾下 webapps 資料夾，然後至 Tomcat 的 bin 資料夾下，執行 startup 指令來啟動 Tomcat，接著使用以下 URI 來請求應用程式（假設 URI 模式為/helloUser）：

```
http://localhost:8080/FirstServlet/helloUser?name=caterpillar
```

實際上在部署 Web 應用程式時，會將應用程式封裝為一個 WAR（Web Archive）檔案，副檔名為*.war。WAR 檔案可使用 JDK 所附的 `jar` 工具程式來建立。例如，如圖 2.9 的方式組織好 Web 應用程式檔案之後，可進入 FirstServlet 資料夾，然後執行以下指令：

```
jar cvf ../FirstServlet.war *
```

這會在 FirstServlet 資料夾外建立 FirstServlet.war 檔案，在 Eclipse 中，則可以直接專案中，按右鍵執行「Export/WAR file」匯出 WAR 檔案。

12.1 使用 Gradle

若要使用Spring，在Spring 3.x或之前的版本中，可以在Spring官方網站[1]直接下載JAR檔案，然而**從 4.x開始，推薦使用Gradle或Maven下載，Spring本身是使用Gradle來管理**，為了要能更方便地使用Spring，認識Gradle是必要的課題。

12.1.1 下載、設定 Gradle

在 Java 中要開發應用程式，必須撰寫原始碼、編譯、執行，過程中必須指定類別路徑、原始碼路徑，相關應用程式檔案必須使用工具程式建構，以完成封裝與部署，嚴謹的應用程式還有測試等階段。

像這類的工作，IDE 解決了部份問題，然而，對於重複需要自動化的流程，單靠 IDE 提供之功能不易解決，因而 Java 的世界中，提供有建構工具來輔助開發人員，在建構工具中元老級的專案是 Ant（Another Neat Tool），使用 Ant 在專案結構上有很大的彈性，然而彈性的另一面就是鎖碎的設定。

另一方面，類似專案會有類似慣例流程，如果能提供預設專案及相關慣例設定，對於開發將會有所幫助，這就是 Maven 後來興起的原因之一，除了提供預設專案及相關慣例設定之外，對於 Java 中程式庫或框架相依性問題，Maven 也提供了集中式貯藏室（Central repository） 解決方案；對於相依性管理問題 Ant 也結合了 Ivy 來進行解決。

然而無論是Ant Ivy、Maven，主要都使用XML進行設定，設定繁鎖，而且有較高的學習曲線，Gradle[2]結合了Ant與Maven的一些好的概念，並使用Groovy語言作為腳本設定，在設定上有了極大簡化，並可以輕易地與Ant、Maven進行整合，種種優點吸引了不少開發者，有些重大專案，像是Spring、Hibernate等，也宣佈改用Gradle 做為建構工具。

---

[1] Spring：spring.io
[2] Gradle：gradle.org

接下來要介紹的，就是Gradle的下載與設定，可以在〈Gradle | Release[3]〉下載Gradle的zip壓縮版本，撰寫本節時的版本是 4.5.1，解壓縮之後會有個gradle-4.5.1 資料夾，其中bin資料夾放置了`gradle`執行檔，為了便於使用，可以在`PATH`環境變數中增加該bin資料夾的路徑，之後開啟文字模式，就可使用`gradle -v`得知Gradle版本：

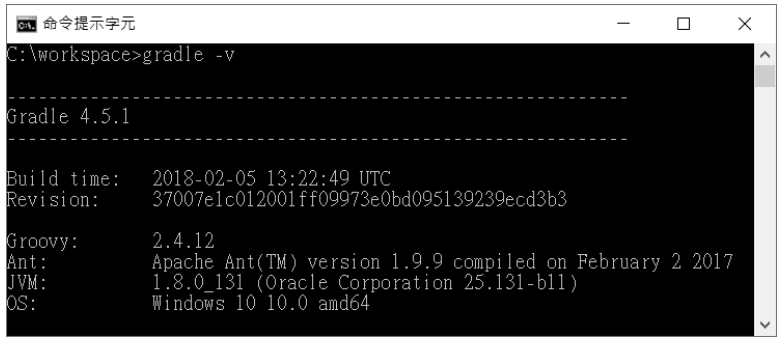

圖 12.1 使用 `gradle -v`

12.1.2 簡單的 Gradle 專案

在文字模式中編譯、執行 Java 應用程式有些麻煩，實際上在編譯.java 原始碼時，如果有多個.java 檔案及已經編譯完成的.class 檔案，必須指定`-classpath`、`-sourcepath` 等，這些都可以讓 Gradle 來代勞。

首先可以建立一個 HelloWorld 資料夾，Gradle 的慣例期待.java 原始碼會置放在 src\main\java 資料夾，依套件階層放置，假設在 HelloWorld\src\main\java\cc\openhome 中有個 Main.java：

HelloWorld Main.java

```
package cc.openhome;

public class Main {
    public static void main(String[] args) {
        System.out.printf("Hello, %s%n", args[0]);
    }
}
```

[3] Gradle | Release：gradle.org/releases/

接著需要在專案資料夾中建立一個 build.gradle 檔案：

HelloWorld build.gradle

```
apply plugin: 'java'
apply plugin:'application'
mainClassName = "cc.openhome.Main"

run {
    args username
}
```

'java'的 plugin 為 Gradle 專案加入了 Java 語言的原始碼編譯、測試與打包（Bundle）等能力；'application'的 plugin 擴充了語言常用的相關任務，像是執行應用程式等；mainClassName 指出了從哪個位元碼檔案的 main 開始執行。run 這個任務中，使用 args 指定了執行位元碼檔案時給定的命令列引數。

接著可以在 HelloWorld 資料夾中如下執行 Gradle：

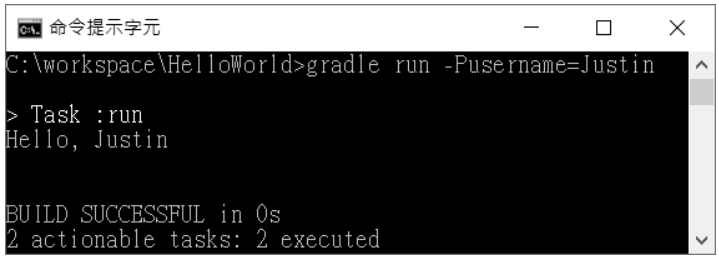

圖 12.2 執行 Java 程式

-Puserame=Justin 指定了 build.gradle 中 username 參數的值為"Justin"，編譯出來的.class 檔案會放置在 build\classes\main 中，不過不用自行建立資料夾 Gradle 會自行建立。

12.1.3 Gradle 與 Eclipse

方才的範例中，必須自行建立.java 對應的套件資料夾，若能結合 IDE 的話會省事許多，目前最新版本的 Eclipse 內建了 Gradle 的支援，最簡單的方式使用 Eclipse 內建的 Gradle Project：

1. 執行選單「File/New/Project…」，在出現的「New Project」對話方塊中，選擇「Gradle/Gradle Project」後按下「Next >」。
2. 在「New Gradle Project」的「Project name」中輸入「Mail」，按下「Finish」按鈕。
3. 展開新建專案中的「Project and External Dependencies」節點，可以看到 Gradle Project 預設相依的程式庫 JAR 檔案。

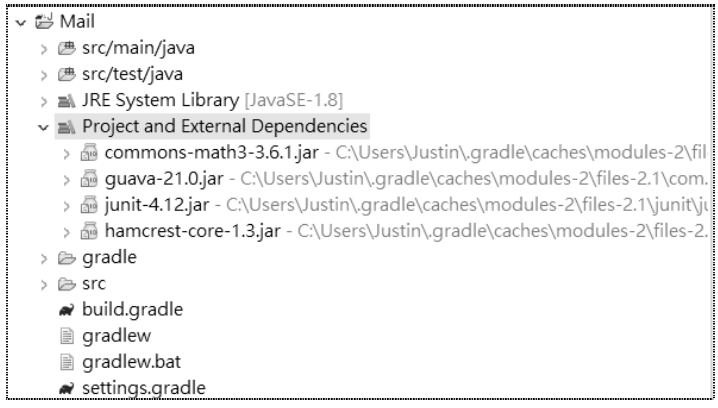

圖 12.3 預設的 Gradle Project

會有這些相依的 JAR 檔案，是因為在預設的 build.gradle 中，已經撰寫了一些定義：

```
apply plugin: 'java-library'

repositories {
    // 預設的相依程式庫來源
    jcenter()
}

dependencies {
    // 這邊可以宣告相依的程式庫資訊
    api 'org.apache.commons:commons-math3:3.6.1'
    implementation 'com.google.guava:guava:21.0'
    testImplementation 'junit:junit:4.12'
}
```

Gradle 會自動下載這些相依的 JAR 檔案，如圖 12.3 所示，預設會將 JAR 檔案儲存在使用者資料夾的.gradle\caches 之中，並自動設定好專案的類別路徑資訊，如果需要新的程式庫，例如 JavaMail，可以在 build.gradle 中定義：

Mail build.gradle

```
apply plugin: 'java-library'

repositories {
    jcenter()
}

dependencies {
    // https://mvnrepository.com/artifact/com.sun.mail/javax.mail
    compile group: 'com.sun.mail', name: 'javax.mail', version: '1.6.0'

    api 'org.apache.commons:commons-math3:3.6.1'
    implementation 'com.google.guava:guava:21.0'
    testImplementation 'junit:junit:4.12'
}
```

接著在專案上按右鍵執行「Gradle/Refresh Gradle Project」，就會下載相依的 JAR 檔案，如果程式庫還有相依於其他程式庫，相關的 JAR 檔案也會一併下載，這就是使用 Gradle 的好處，不用為了程式庫間複雜的相依性而焦頭爛額。

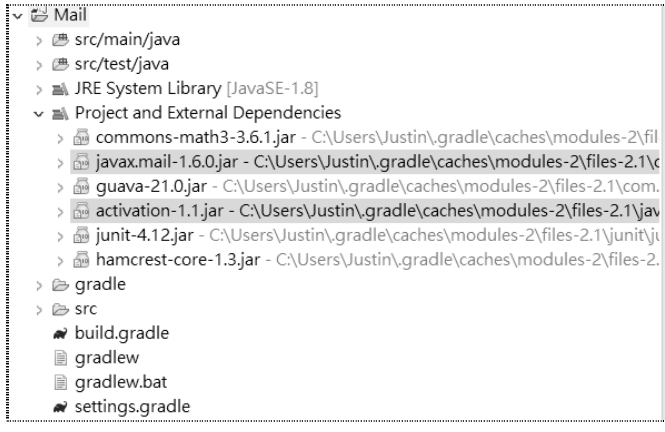

圖 12.4 自動下載相依的程式庫 JAR 檔案

可以在 src/main/java 之中建立類別撰寫一些程式，例如簡單地寄送郵件：

Mail Main.java

```
package cc.openhome;

...略

public class Main {
    ...略
```

```
    public static void main(String[] args) {
        try {
            Message message = createMessage(
                    "from@gmail.com",
                    "to@gmail.com", "測試", "這是一封測試");
            Transport.send(message);
            System.out.println("郵件傳送成功");
        } catch (MessagingException e) {
            throw new RuntimeException(e);
        }

    }

    ...略
}
```

想要執行程式的話，同樣只需要在原始碼上按右鍵執行「Run As/Java Application」就可以了。

如果是既有的 Java 應用程式專案，可以直接在專案上按右鍵執行「Configure/Add Gradle Nature」，讓既有專案支援最基本的 Gradle 特性，接著在專案上按右鍵執行「New/File」建立 build.gradle 檔案，在其中撰寫定義，例如：

Mail2 build.gradle

```
apply plugin: 'java-library'

// 設置 Java 源碼所在目錄
sourceSets.main.java.srcDir 'src'

repositories {
    jcenter()
}

dependencies {
    // https://mvnrepository.com/artifact/com.sun.mail/javax.mail
    compile group: 'com.sun.mail', name: 'javax.mail', version: '1.6.0'
}
```

同樣地，接著在專案上按右鍵執行「Gradle/Refresh Gradle Project」，就會下載相依的 JAR 檔案，如果程式庫還有相依於其他程式庫，相關 JAR 檔案也會一併下載。

想要執行程式的話，同樣只需要在原始碼上按右鍵執行「Run As/Java Application」就可以了。

提示 在 Eclipse 中匯入 Gradle 專案之後，記得在專案上按右鍵執行「Gradle/Refresh Gradle Project」，重新整理專案中相依的程式庫資訊。

12.2 認識 Spring 核心

在學會使用 Gradle 之後，接下來就可以試著使用 Spring 框架，然而，整個 Spring 框架是非常龐大的，試圖完全掌握沒有意義，這一節將從 Spring 的核心開始認識，初步運用 Spring 來解決一些問題。

接下來並不會全面地介紹 Spring，這不是全書設定的目標，**會介紹 Spring 的原因主要是作為一個銜接**，希望在接下來的章節之後，你有能力自行探討更多有關 Spring 的課題。

12.2.1 相依注入

在先前的微網誌應用程式中，為了要能建構 `UserService` 實例，必須建構 `AccountDAOJdbcImpl`、`MessageDAOJdbcImpl` 實例，而為了要能建構這兩個實例，必須先建構 `DataSource` 實例：

```
package cc.openhome;

import org.h2.jdbcx.JdbcDataSource;
...略

public class Main {
    public static void main(String[] ags) {
        // JdbcDataSource 實現了 DataSource 介面
        JdbcDataSource dataSource = new JdbcDataSource();
        dataSource.setURL(
             "jdbc:h2:tcp://localhost/c:/workspace/SpringDI/gossip");
        dataSource.setUser("caterpillar");
        dataSource.setPassword("12345678");

        AccountDAO acctDAO = new AccountDAOJdbcImpl(dataSource);
        MessageDAO messageDAO = new MessageDAOJdbcImpl(dataSource);

        UserService userService = new UserService(acctDAO, messageDAO);
```

```
        userService.messages("caterpillar")
                    .forEach(message -> {
                        System.out.printf("%s\t%s%n",
                            message.getLocalDateTime(),
                            message.getBlabla());
                    });
    }
}
```

物件的建立與相依注入（Dependency Injection）當然是必要的關切點，只不過當過程太過冗長，模糊了商務流程之時，應該適當地將之分離，也許建立一個工廠方法會比較好：

```
package cc.openhome;
...略
public class Service {
    public static UserService getUserService() {
        JdbcDataSource dataSource = new JdbcDataSource();
        dataSource.setURL(
            "jdbc:h2:tcp://localhost/c:/workspace/SpringDI/gossip");
        dataSource.setUser("caterpillar");
        dataSource.setPassword("12345678");

        AccountDAO acctDAO = new AccountDAOJdbcImpl(dataSource);
        MessageDAO messageDAO = new MessageDAOJdbcImpl(dataSource);

        UserService userService = new UserService(acctDAO, messageDAO);
        return userService;
    }
}
```

那麼要取得 `UserService` 實例，就只要如下撰寫：

```
package cc.openhome;

import cc.openhome.model.UserService;

public class Main {
    public static void main(String[] ags) {
        UserService userService = Service.getUserService();

        userService.messages("caterpillar")
                    .forEach(message -> {
                        System.out.printf("%s\t%s%n",
                            message.getLocalDateTime(),
                            message.getBlabla());
                    });
    }
}
```

如此之來，程式碼的流程清晰了，而且即使是不懂 JDBC 或 DataSource 等的開發者，只要透過這樣的方式，也可以直接取得 UserService 進行操作。

上面的 Service 當然是特定用途，隨著打算開始整合各種程式庫或方案，你會遇到各種物件建立與相依設定需求，為此，你可能會重構 Service，使之越來越通用，像是可透過組態檔來進行相依設定，甚至成為一個通用於各式物件建立與相依設定的容器，實際上這類容器，在 Java 的世界中早已存在，且有多樣性的選擇，而最有名的實現之一就是 Spring 框架。

12.2.2 使用 Spring 核心

Lab

為了要能使用 Spring 核心，必須在 build.gradle 中設定，由於會使用到 H2 資料庫驅動程式，這也可以一併透過 Gradle 來管理相關的 JAR：

SpringDI build.gradle

```
apply plugin: 'java-library'

repositories {
    jcenter()
}

dependencies {
    testImplementation 'junit:junit:4.12'

    // https://mvnrepository.com/artifact/com.h2database/h2
    compile('com.h2database:h2:1.4.196')

    // https://mvnrepository.com/artifact/org.springframework/spring-context
    compile('org.springframework:spring-context:5.0.3.RELEASE')
}
```

由於 H2 的 JdbcDataSource 原始碼並不在專案的控制之內，這可以在建立 Spring 設定檔時，一併撰寫在其中：

SpringDI AppConfig.java

```
package cc.openhome;

import javax.sql.DataSource;

import org.h2.jdbcx.JdbcDataSource;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
```

```
import org.springframework.context.annotation.Configuration;

@Configuration
@ComponentScan
public class AppConfig {
    @Bean
    public DataSource getDataSource() {
        JdbcDataSource dataSource = new JdbcDataSource();
        dataSource.setURL(
            "jdbc:h2:tcp://localhost/c:/workspace/gossip/gossip");
        dataSource.setUser("caterpillar");
        dataSource.setPassword("12345678");
        return dataSource;
    }
}
```

Spring 支援多種設定方式，最方便的一種是透過標註，`@Configuration` 告知 Spring，這個 `AppConfig` 是作為設定檔使用，由於它也是個 Java 類別，使用它作為設定檔的好處之一是，可以在其中撰寫 Java 程式碼，如同 `getDataSource()` 中示範的。

提示 »» `@Configuration` 標註的類別，在透過 Spring 的處理之後，在角色上真的就像是個設定檔，在某些行為上並不是 Java 程式該有的行為，這部份細節請參考 Spring 專書。

由 Spring 管理的實例稱為 Bean，`@Bean` 告訴 Spring，`getDataSource()` 傳回的實例會作為 Bean 元件，至於其他的 Bean，像是 `AccountDAOJdbcImpl`、`MessageDAOJdbcImpl`、`UserService` 等實例，實際上也可以寫在 `AppConfig` 裏，然而由於它們的原始碼在控制之中，更方便的作法是透過 Spring 來自動掃描與自動綁定。

為了能 Spring 能自動掃描 Bean 的存在，`AppConfig` 上標註了 `@ComponentScan`，如此 Spring 會自動掃描同一套件以及其子套件下，是否有 Bean 元件的存在。

接著可以來處理 `AccountDAOJdbcImpl` 的自動綁定：

SpringDI AccountDAOJdbcImpl.java

```
package cc.openhome.model;
...略
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

@Component
public class AccountDAOJdbcImpl implements AccountDAO {
```

```
    @Autowired
    private DataSource dataSource;
    ...略
}
```

在這邊看到 DataSource 的 dataSource 值域上標註了@Autowired，當 Spring 在自身管理的 Bean 中發現了相同類型的實例，會自動設定給 dataSource，而 AccountDAOJdbcImpl 本身也會被 Spring 作為 Bean 管理，因此可以使用@Component 來標註，表示這也是個 Bean 元件。

MessageDAOJdbcImpl 也做了類似的標註：

SpringDI MessageDAOJdbcImpl.java

```
package cc.openhome.model;
...略

@Component
public class MessageDAOJdbcImpl implements MessageDAO {
    @Autowired
    private DataSource dataSource;
    ...略
}
```

接下來 UserService 也是類似，只不過自動綁定的對象包括了 AccountDAO 與 MessageDAO 的實例：

SpringDI UserService.java

```
package cc.openhome.model;
...略

@Component
public class UserService {
    @Autowired
    private final AccountDAO acctDAO;

    @Autowired
    private final MessageDAO messageDAO;

    ...略
}
```

如上設定與標註之後，就可以透過 Spring 來取得 DataSource、AccountDAOJdbcImpl、MessageDAOJdbcImpl 或 UserService 實例了，然而，必須要有個物件來讀取 AppConfig 設定檔：