使用 JSTL

7 CHAPTER

學習目標

- 了解何謂 JSTL
- 使用 JSTL 核心標籤庫
- 使用 JSTL 格式標籤庫
- 使用 JSTL XML 標籤庫
- 使用 JSTL 函式標籤庫

7.1 簡介 JSTL

需要依某條來決定顯示網頁片段，或是需要使用迴圈顯示表格內容，然而，HTML 或 JSP 本身並沒有什麼<if>標籤，更沒什麼<for>標籤達到這個目的。

這些跟頁面呈現相關的邏輯判斷標籤，可由 JSTL（JavaServer Pages Standard Tag Library）提供。JSTL 提供的標籤庫分作五個大類：

- 核心標籤庫

  提供條件判斷、屬性存取、URI 處理及錯誤處理等標籤。

- I18N 相容格式標籤庫

  提供數字、日期等的格式化功能，以及區域（Locale）、訊息、編碼處理等國際化功能。

- SQL 標籤庫

  提供基本的資料庫查詢、更新、設定資料來源（DataSource）等功能，這會在第 9 章說明 JDBC 時再介紹。

- XML 標籤庫

  提供 XML 剖析、流程控制、轉換等功能。

- 函式標籤庫

  提供常用字串處理的函式標籤庫。

JSTL 是另一標準規範，並非在 JSP 的規範當中，可以透過〈JavaServer Pages Standard Tag Library[1]〉找到 JSTL 的原始碼。如果想取得 JSTL 的 JAR 檔案，可以在〈Apache Taglibs Downloads[2]〉下載，撰寫本書的這個時間點，可以找到 JSTL 1.2.5，需要下載 taglibs-standard-spec-1.2.5.jar 與 taglibs-standard-impl-1.2.5.jar，前者是 JSTL 標準介面與類別，後者是實作。

可以將 JSTL 的兩個 JAR 檔案，放置到 Web 應用程式的 WEB-INF/lib 資料夾。如果需要 API 文件說明，可以在〈Apache Standard Taglib 1.2 API[3]〉找到。

在 Eclipse 中，雖然可以直接將 JAR 檔案複製至專案的 /WEB-INF/lib 資料夾，不過專案各自擁有 JAR 檔案，管理上會很麻煩。可以將 JAR 檔案統一放置在某個資料夾中，再透過 Eclipse 的「Deployment Assembly」設定使用 JAR 檔案，在建立新專案後，可以按照以下步驟進行操作：

1. 在專案上按右鍵，執行「Properties」，在出現的專案屬性對話方塊上，選擇「Deployment Assembly」。
2. 按下「Web Deployment Assembly」右邊的「Add」按鈕，在出現的「New Assembly Directive」對話方塊中，選擇「Archives from File System」後按下「Next」。
3. 按下「Add」按鈕，選擇檔案系統中的 JAR 檔案，按下「Finish」按鈕。
4. 按下「Web Deployment Assembly」的「OK」按鈕。
5. 在專案的「Java Resources/Libraries」節點，可以發現「Web App Libraries」已設定 JAR 檔案。

---

1 JSTL：www.oracle.com/technetwork/java/index-jsp-135995.html

2 Apache Taglibs Downloads：tomcat.apache.org/download-taglibs.cgi

3 Apache Standard Taglib 1.2 API：tomcat.apache.org/taglibs/standard/apidocs/

JSTL 從 Java EE 5 開始就沒有什麼顯著的特性變更，從今日眼光來說，只能說是提供了基本功能，部分標籤也顯得過時，有許多開放原始碼自訂標籤庫可以取代 JSTL；然而在某些場合，還是會遇到 JSTL，而在學習自訂標籤庫時，JSTL 也是個可模仿的對象，因此在本書中仍保留對 JSTL 的介紹。

JSTL 標籤種類也蠻多的，本章將先說明 JSTL 核心標籤庫、格式標籤庫 XML 標籤庫與函式標籤庫，第 9 章說明 JDBC 後再說明 SQL 標籤庫。

如果你真的不需要 JSTL，就銜接本書後續內容而言，只需要認識 7.2 核心標籤庫的內容就可以了。

要使用 JSTL 標籤庫，必須在 JSP 中使用 **taglib** 指示元素，定義前置名稱與 uri 參考，例如使用核心標籤庫的話，可以如下定義：

```
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
```

前置名稱設定了標籤庫在 JSP 中的名稱空間，以避免與其他標籤庫發生衝突，慣例上使用 JSTL 核心標籤庫時，會使用 c 作為前置名稱。uri 參考告知容器，如何參考 JSTL 標籤庫實作（如 6.3.5 節定義 TLD 時的作用，可先參考該節內容，第 8 章說明自訂標籤時還會看到相關說明）。

7.2　核心標籤庫

JSTL 核心標籤庫主要包括流程處理標籤，像是<c:if>、<c:forEach>等，可處理頁面呈現邏輯，錯誤處理標籤可捕捉例外，網頁匯入、重新導向標籤提供比原有<jsp:include>、<jsp:forward>更進一步的功能，屬性處理標籤可提供比原有<jsp:setProperty>更多的設定，其他還有輸出處理標籤、URI 處理標籤等，可用於處理頁面邏輯。

7.2.1　流程處理標籤

當 JSP 必須根據某條件安排內容輸出時，可以使用流程標籤。例如想依使用者名稱、密碼請求參數，來決定是否顯示某畫面，或想用表格輸出多筆資料等。

首先介紹**<c:if>**標籤的使用（假設標籤前置使用"c"），這個標籤可根據運算式的結果，決定是否顯示本體內容。直接來看個範例：

JSTL login.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<!DOCTYPE html">
<html>
    <head>
        <meta charset="UTF-8">
        <title>登入頁面</title>
    </head>
    <body>
        <c:if test="${param.name == 'momor' && param.password == '12345678'}">
            <h1>${param.name} 登入成功</h1>
        </c:if>
    </body>
</html>
```

<c:if>標籤 **test** 屬性可以放置 EL 運算式，如果運算式結果是 true，會將<c:if>本體輸出。就上例來說，若發送的請求參數中，名稱與密碼正確，就會顯示名稱與登入成功的訊息。

提示»» 為了避免流於語法說明的瑣碎細節，本章不會試圖說明 JSTL 標籤所有屬性，在需要的時候，可參考 JSTL 的線上文件說明或 JSTL 規格書 JSR52。

<c:if>標籤僅在 test 結果為 true 時顯示本體內容，不過沒有對應的<c:else>標籤。想在某條件式成立時顯示某內容，不成立顯示另一內容，可使用**<c:choose>**、**<c:when>**及**<c:otherwise>**標籤。

JSTL login2.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<jsp:useBean id="user" class="cc.openhome.User"  />
<jsp:setProperty name="user" property="*" />
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>登入頁面</title>
    </head>
    <body>
        <c:choose>
```

```
            <c:when test="${user.valid}">
                <h1>
                    <jsp:getProperty name="user" property="name"/>登入成功
                </h1>
            </c:when>
            <c:otherwise>
                 <h1>登入失敗</h1>
            </c:otherwise>
        </c:choose>
    </body>
</html>
```

這個範例改寫自 6.2.2 節的使用者登入範例。在 6.2.2 節時，使用了 Scriptlet 撰寫 Java 程式碼，判斷使用者是否發送正確名稱密碼，分別顯示登入成功或失敗畫面，使用<c:choose>、<c:when>及<c:otherwise>標籤，就可以不使用 Scriptlet 來實現需求。

<c:when> 及<c:otherwise>必須放在<c:choose>中。當<c:when>的 **test** 運算為 true 時，輸出<c:when>本體內容，而不理會<c:otherwise>的內容。<c:choose>中可以有多個<c:when>標籤，此時會從上往下測試，如果有個<c:when>標籤 test 運算為 true 就輸出本體內容，忽略後續的<c:when>與<c:otherwise>。若<c:when>測試都不成立，會輸出<c:otherwise>的內容。

如果打算產生一連串的資料輸出。例如有個留言版程式，使用 JavaBean 從資料庫中取得留言，留言可能有數十則，以陣列方式傳回：

JSTL MessageService.java

```
package cc.openhome;

public class MessageService {
    // 放些假資料，假裝這些資料是來自資料庫
    private Message[] fakeMessages = {
         new Message("caterpillar", "caterpillar's message!"),
         new Message("momor", "momor's message!"),
         new Message("hamimi", "hamimi's message!")
    }

    public Message[] getMessages() {
        return fakeMessages;
    }
}
```

Message 物件 name 與 text 屬性，各代表留言者名稱與留言文字，你打算使用表格來顯示每則留言，若不想使用 Scriptlet，可以使用 JSTL 的**<c:forEach>**標籤實現需求。例如：

JSTL message.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<!DOCTYPE html>
<jsp:useBean id="messageService" class="cc.openhome.MessageService"/>
<html>
    <head>
        <meta charset="UTF-8">
        <title>留言版</title>
    </head>
    <body>
        <table style="text-align: left; width: 100%;" border="1">
            <tr>
                <td>名稱</td><td>訊息</td>
            </tr>
            <c:forEach var="message" items="${messageService.messages}">
                <tr>
                    <td>${message.name}</td><td>${message.text}</td>
                </tr>
            </c:forEach>
        </table>
    </body>
</html>
```

<c:forEach>標籤 **items** 屬性可以是陣列、Collection、Iterator、Enumeration、Map 與字串，每次循序從 items 指定的物件取出一個元素，指定給 **var** 屬性設定之名稱，接著就可以在<c:forEach>標籤中，使用該名稱取得元素。這個範例的執行畫面如下所示：

圖 7.1 <c:forEach>範例網頁執行結果

關於國際化

國際化的三個重要觀念是**地區（Locale）**資訊、**資源包（Resource bundle）**與**基礎名稱（Base name）**。

地區資訊代表了特定的地理、政治或文化，地區資訊可由一個語言編碼（Language code）與可選的地區編碼（Country code）來指定，其中語言編碼是 ISO 639[4]定義，由兩個小寫字母代表，例如"fr"表示法文（French），"zh"表示中文（Chinese）。地區編碼由兩個大寫字母表示，定義在 ISO 3166[5]，例如"IT"表示義大利（Italy）、"TW"表示臺灣（Taiwan）。

在 3.3.2 略提過地區（Locale）資訊的對應類別 Locale，在建立 Locale 時，可以指定語言編碼與地區編碼，例如建立代表臺灣正體中文的 Locale：

```
Locale locale = new Locale("zh", "TW");
```

資源包中包括了特定地區的資訊，先前介紹的 ResourceBundle 物件，就是 JVM 中資源包的代表物件，同一組訊息但不同地區的各個資源包，會共用相同的基礎名稱，使用 ResourceBundle 的 getBundle()時指定的名稱，就是在指定基礎名稱。

ResourceBundle 的 getBundle()若指定"messages"，會嘗試以預設 Locale（由 Locale.getDefault()取得的值）取得.properties 檔案。例如，若預設的 Locale 代表 zh_TW，會嘗試取得 messages_zh_TW.properties，若找不到，再嘗試尋找 messages.properties。

如果希望建立 messages_zh_TW.properties，在當中建立臺灣正體中文的訊息，Java SE 8 以前必須使用 Unicode 碼點表示，這可以透過 JDK 工具程式 native2ascii 來協助轉換。例如，可以在 messages_zh_TW.txt 中撰寫以下內容：

```
cc.openhome.welcome=哈囉
cc.openhome.name=世界
```

如果編輯器使用 MS950 編碼，那麼可以如下執行 native2ascii 程式：

```
> native2ascii -encoding MS950 messages_zh_TW.txt messages_zh_TW.properties
```

4 ISO 639：zh.wikipedia.org/wiki/ISO_639

5 ISO 3166：zh.wikipedia.org/wiki/ISO_3166

如此就會產生 messages_zh_TW.properties 檔案，內容如下：

```
cc.openhome.welcome=\u54c8\u56c9
cc.openhome.name=\u4e16\u754c
```

提示 »» 在 Eclipse 中編輯.properties 時，若撰寫了中文，編輯器會自動轉為碼點表示。

如果想將 Unicode 碼點表示的.properties 轉回中文，可以使用-reverse 引數，例如將上面的程式轉回中文，並使用 UTF-8 編碼檔案儲存：

```
> native2ascii -reverse -encoding UTF-8 messages_zh_TW.properties
messages_zh_TW.txt
```

如果執行先前的 Hello 類別，而系統預設 Locale 為 zh_TW，會顯示"哈囉!世界!"的結果。如果提供 messages_en_US.properties：

```
cc.openhome.welcome=Hello
cc.openhome.name=World
```

ResourceBundle 的 getBundle()可以指定 Locale 物件，若如下撰寫程式：

```
Locale locale = new Locale("en", "US");
ResourceBundle res = ResourceBundle.getBundle("messages", locale);
System.out.print(res.getString("cc.openhome.welcome") + "!");
System.out.println(res.getString("cc.openhome.name") + "!");
```

ResourceBundle 會嘗試取得 messages_en_US.properties 中的訊息，結果就是顯示"Hello!World!"。

提示 »» Java SE 9 以後支援 UTF-8 編碼的.properties 檔案，如果希望建立 messages_zh_TW.properties，並在直接撰寫臺灣正體中文的訊息，只要使用 UTF-8 編碼就可以了，因而 native2ascii 工具程式也就從 JDK9 移除了。

7.3.2　訊息標籤

要使用 JSTL 的 i18n 相容格式標籤庫，慣例上會使用 fmt 作為前置名稱，JSTL 格式標籤庫的 uri 參考為 http://java.sun.com/jsp/jstl/fmt。例如：

```
<%@taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt"%>
```

來看最基本的**<fmt:bundle>**、**<fmt:message>**如何使用，假設準備了一個messages1.properties 檔案如下：

JSTL messages1.properties

```
cc.openhome.title=Welcome
cc.openhome.forGuest=Hello! Guest!
```

這個.properties 檔案必須放在/WEB-INF/classes，在 Eclipse 中，可以在專案的「Java Resources/src」新增檔案。接著建立 JSP 檔案：

JSTL fmt1.jsp

```
<%@page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt"%> ←❶定義前置名稱與 uri
<!DOCTYPE html>
<fmt:bundle basename="messages1"> ←❷使用<fmt:bundle>
<html>
    <head>
        <meta charset="UTF-8">
        <title><fmt:message key="cc.openhome.title" /></title> ←❸使用 <fmt:message>
    </head>
    <body>
            <h1><fmt:message key="cc.openhome.forGuest" /></h1>
    </body>
</html>
</fmt:bundle>
```

首先，使用 taglib 指示元素定義前置名稱與 uri❶，然後使用<fmt:bundle>指定 **basename** 屬性為"messages1"❷，這表示預設的訊息檔案為 messages1.properties，使用<fmt:message>的 **key** 屬性則指定訊息檔案中的哪條訊息❸。下圖為執行時的一個參考畫面：

圖 7.3 範例網頁執行結果

如果將<fmt:bundle>的 basename 改設定為"messages2"，並且另外準備一個messages2.properties：

JSTL messages2.properties

```
cc.openhome.title=Aloha
cc.openhome.forGuest=Hi! New Guest!
```

那麼顯示出來的畫面中，訊息內容就是來自 messages2.properties，如下圖：

圖 7.4　範例網頁執行結果

也可以使用**<fmt:setBundle>**標籤設置 **basename** 屬性，設置的效力預設是整個頁面都有作用，若額外有<fmt:bundle>設置，會以<fmt:bundle>的設置為主，例如：

JSTL fmt2.jsp

```
<%@page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt"%>
<!DOCTYPE html>
<fmt:setBundle basename="messages1"/> ←❶ 使用<fmt:setBundle>
<html>
    <head>
        <meta charset="UTF-8">
        <title><fmt:message key="cc.openhome.title" /></title>
    </head>
    <body>
            <h1><fmt:message key="cc.openhome.forGuest" /></h1>
            <fmt:bundle basename="messages2"> ←❷ 使用<fmt:bundle>
                <h1><fmt:message key="cc.openhome.forGuest" /></h1>
            </fmt:bundle>
    </body>
</html>
```

這個 JSP 一開始使用<fmt:setBundle>設置 basename 為"messages1"❶，第一個<fmt:message>取得的訊息來自 messages1.properties，另一個被<fmt:bundle>包括的<fmt:message>，取得的訊息來自 messages2.properties❷。

如果訊息中有些部分必須動態決定，可以使用佔位字符先代替，例如：

JSTL messages3.properties

```
cc.openhome.title=Hello
cc.openhome.forUser=Hi! {0}! It is {1, date, long} and {2, time ,full}.
```

在上面的訊息檔案中，粗體字部分就是佔位字符，號碼從 0 開始，分別代表第幾個佔位字符，在指定時可以指定型態與格式，使用的格式是由 java.text.MessageFormat 定義，可參考 java.text.MessageFormat 的 API 文件說明。

如果想設置佔位字符的真正內容，是使用**<fmt:param>**標籤，例如：

JSTL fmt3.jsp

```
<%@page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt"%>
<jsp:useBean id="now" class="java.util.Date"/> ←❶ 建立 Date 取得目前時間
<!DOCTYPE html>
<fmt:setBundle basename="messages3"/> ←❷ 指定訊息檔案
<html>
    <head>
        <meta charset="UTF-8">
        <title><fmt:message key="cc.openhome.title" /></title>
    </head>
    <body>
            <fmt:message key="cc.openhome.forUser">
                <fmt:param value="${param.username}"/>
                <fmt:param value="${now}"/>
                <fmt:param value="${now}"/>
            </fmt:message>
    </body>
</html>
```

❸ 逐一設置佔位字符

在這個 JSP 中，使用<jsp:useBean>建立 Date 物件以取得目前系統時間，並設置為屬性，訊息檔案的基礎名稱設定為"messages3"，而訊息檔案中每個佔位字元，使用<fmt:param>逐一設置，執行的結果畫面如下所示：

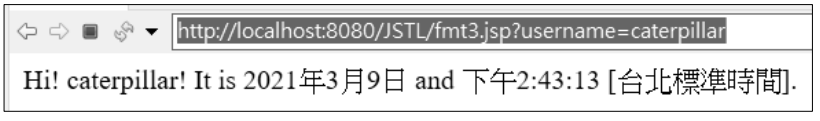

圖 7.5 範例網頁執行結果

7.3.3 地區標籤

在正式介紹地區標籤之前，先看看 Java SE 中，使用 ResourceBundle 時，如何根據基礎名稱取得對應的訊息檔案：

1. 使用指定的 Locale 物件取得訊息檔案。
2. 使用 Locale.getDefault()取得的物件取得訊息檔案。

3. 使用基礎名稱取得訊息檔案。

JSTL 略有不同，簡單地說，JSTL 的 i18n 相容性標籤，會嘗試從屬性範圍取得 **javax.servlet.jsp.jstl.fmt.LocalizationContext** 物件，藉以決定資源包與地區資訊，具體來說，決定訊息檔案的順序如下：

1. 使用指定的 Locale 物件取得訊息檔案。
2. 根據瀏覽器 Accept-Language 標頭指定的偏好地區（Prefered locale）順序，這可以使用 HttpServletRequest 的 getLocales()取得。
3. 根據後備地區（fallback locale）資訊取得訊息檔案。
4. 使用基礎名稱取得訊息檔案

例如，先前的範例並沒有指定 Locale，而瀏覽器指定的偏好地區為"zh_TW"，因而嘗試尋找 messages3_zh_TW.properties 檔案，結果沒有找到，而範例並沒有設置偏好地區，最後尋找 messages.properties 檔案。

<fmt:message>標籤有個 **bundle** 屬性，可用以指定 LocalizationContext 物件，可以在建立 LocalizationContext 物件時，指定 ResourceBundle 與 Locale 物件，例如以下程式會嘗試從四個訊息檔案取得訊息：

JSTL fmt4.jsp

```
<%@page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@page import="java.util.*, javax.servlet.jsp.jstl.fmt.*"%>
<%@taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt"%>
<!DOCTYPE html>
<%
    // 假設這邊的 Java 程式碼是在另一個控制器中完成的
    ResourceBundle zh_TW =
        ResourceBundle.getBundle("hello", new Locale("zh", "TW"));
    ResourceBundle zh_CN =
        ResourceBundle.getBundle("hello", new Locale("zh", "CN"));
    ResourceBundle ja_JP =
        ResourceBundle.getBundle("hello", new Locale("ja", "JP"));
    ResourceBundle en_US =
        ResourceBundle.getBundle("hello", new Locale("en", "US"));
    pageContext.setAttribute("zh_TW", new LocalizationContext(zh_TW));
    pageContext.setAttribute("zh_CN", new LocalizationContext(zh_CN));
    pageContext.setAttribute("ja_JP", new LocalizationContext(ja_JP));
    pageContext.setAttribute("en_US", new LocalizationContext(en_US));
%>                                                    ❶建立 LocalizationContext
<html>
```

```
    <head>
        <meta charset="UTF-8">
    </head>
    <body>
                                ❷指定 LocalizationContext
            <fmt:message bundle="${zh_TW}" key="cc.openhome.hello"/><br>
            <fmt:message bundle="${zh_CN}" key="cc.openhome.hello"/><br>
            <fmt:message bundle="${ja_JP}" key="cc.openhome.hello"/><br>
            <fmt:message bundle="${en_US}"  key="cc.openhome.hello"/>
    </body>
</html>
```

範例中使用四個 ResourceBundle 建立四個 LocalizationContext，並指定為 page 屬性範圍❶，在使用<fmt:message>時，指定 bundle 屬性為不同的 LocalizationContext❷，範例還準備了四個不同的.properties，分別代表繁體中文的 hello_zh_TW.properties、簡體中文的 hello_zh_CN.properties、日文的 hello_ja_JP.properties 與美式英文的 hello_en_US.properties，內容已透過工具轉換為 Unicode 碼點表示。結果如下所示：

圖 7.6 顯示不同訊息檔案的訊息

如果要共用 Locale 資訊，可以使用<fmt:setLocale>標籤，在 **value** 屬性指定地區資訊。例如：

JSTL fmt5.jsp

```
<%@page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt"%>
<fmt:setLocale value="zh_TW"/>
<fmt:setBundle basename="hello"/>
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
    </head>
    <body>
            <fmt:message key="cc.openhome.hello"/>
    </body>
</html>
```

這個 JSP 會使用 hello_zh_TW.properties 網頁，結果就是顯示「哈囉」的文字。

<fmt:setLocale>會呼叫 HttpServletResponse 的 setLocale()設定回應編碼，事實上，<fmt:bundle>、<fmt:setBundle>或<fmt:message>也會呼叫 HttpServletResponse 的 setLocale()設定回應編碼，不過要注意，正如 3.3.2 提到，如果使用 setCharacterEncoding()或 setContentType()時指定 charset，就會忽略 setLocale()。

<fmt:requestEncoding> 用來設定請求物件的編碼，它會呼叫 HttpServletRequest 的 setCharacterEncoding()，必須在取得請求參數前使用。

提示 »» 對於初學者，使用<fmt:setLocale>與<fmt:setBundle>來設定地區與訊息檔案基礎名稱就足夠了，不過 JSTL i18n 的功能與彈性蠻大的，接下來要說明的內容比較進階，初學可以暫時忽略。

<fmt::message>等標籤會使用 LocalizationContext 取得地區與資源包資訊，<fmt:setLocale>會在屬性範圍設定 LocalizationContext，如果想使用程式碼設定 LocalizationContext 物件，可以透過 javax.servlet.jsp.jstl.core.Config 的 set()方法設定。例如：

JSTL fmt6.jsp

```
<%@page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@page import="java.util.*,javax.servlet.jsp.jstl.core.*"%>
<%@page import="javax.servlet.jsp.jstl.fmt.*"%>
<%@taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt"%>
<%
    Locale locale = new Locale("ja", "JP");
    ResourceBundle res = ResourceBundle.getBundle("hello", locale);
    Config.set(pageContext, Config.FMT_LOCALIZATION_CONTEXT,
         new LocalizationContext(res), PageContext.PAGE_SCOPE);
%>
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
    </head>
    <body>
            <fmt:message key="cc.openhome.hello"/>
    </body>
</html>
```

這個 JSP 沒有使用<fmt:setLocale>也沒有指定<fmt:message>的 bundle 屬性，因此使用預設的 LocalizationContext，如粗體字的程式所示，在設定 LocalizationContext 時可以指定屬性範圍，<fmt:message>會自動在四個屬性範圍依次搜尋 LocalizationContext，找到就使用，如果後續有使用<fmt:setLocale>或指定<fmt:message>的 bundle 屬性，以後續指定為主。

另一個指定預設 LocalizationContext 的方式，就是直接指定屬性名稱，例如在 ServletContextListener 如下指定：

```
...
    public void contextInitialized(ServletContextEvent sce) {
        Locale locale = new Locale("ja", "JP");
        ResourceBundle res = ResourceBundle.getBundle("hello", locale);
        ServletContext context = sce.getServletContext();
        context.setAttribute(
         "javax.servlet.jsp.jstl.fmt.LocalizationContext.application",
         new LocalizationContext(res));
    }
...
```

屬性名稱開頭是"javax.servlet.jsp.jstl.fmt.localizationContext"並加上一個範圍後綴字，四個範圍的後綴字是".page"、".request"、".session"與".application"。事實上，若使用<fmt:setBundle>時，就會設置這個屬性，範圍可由 scope 屬性來決定，預設值是"page"。

<fmt:setLocale>可以設置地區資訊，如果想使用程式碼來設置地區資訊，可以使用 Config 的 set()如下設定：

```
<%
    ...
    Config.set(pageContext, Config.FMT_LOCALE,
         new Locale("ja", "JP"), PageContext.PAGE_SCOPE);
%>
```

或者是直接指定屬性名稱，例如在 ServletContextListener 如下指定：

```
...
    public void contextInitialized(ServletContextEvent sce) {
        ServletContext context = sce.getServletContext();
        context.setAttribute(
         " javax.servlet.jsp.jstl.fmt.locale.application",
         new Locale("ja", "JP"));
    }
...
```

屬性名稱開頭是"javax.servlet.jsp.jstl.fmt.locale"加上一個範圍後綴字，四個範圍的後綴字是".page"、".request"、".session"與".application"。若使用<fmt:setLocale>時，就會設置這個屬性，範圍可由 scope 屬性來決定，預設值是"page"。

如果想設置後備地區資訊，可以使用 Config 的 set()設定：

```
<%
    ...
    Config.set(pageContext, Config.FMT_FALLBACK_LOCALE,
         new Locale("ja", "JP"), PageContext.PAGE_SCOPE);
%>
```

或者是直接指定屬性名稱，例如在 ServletContextListener 如下指定：

```
...
    public void contextInitialized(ServletContextEvent sce) {
        ServletContext context = sce.getServletContext();
        context.setAttribute(
         " javax.servlet.jsp.jstl.fmt.fallbackLocale.application",
         new LocalizationContext(new Locale("ja", "JP")));
    }
...
```

屬性名稱開頭是"javax.servlet.jsp.jstl.fmt.fallbackLocale"加上一個範圍後綴字，四個範圍的後綴字是".page"、".request"、".session"與".application"。

Locale、LocalizationContext或後備地區資訊會分別被哪個標籤使用或設置，在 JSTL 的規格書 JSR52 的表格 8.11 做了不錯的整理，以下摘錄表格內容：

表 7.1　Locale 的設定與使用

隱含物件	說明
屬性名稱前置	javax.servlet.jsp.jstl.fmt.locale
Java 常數	Config.FMT_LOCALE
設置型態	Locale 或 String
由哪個標籤設置	<fmt:setLocale>
被哪些標籤使用	<fmt:bundle>、<fmt:setBundle>、<fmt:message>、<fmt:formatNumber>、<fmt:parseNumber>、<fmt:formatDate>、<fmt:parseDate>

表 7.2 後備地區的設定與使用

隱含物件	說明
屬性名稱前置	javax.servlet.jsp.jstl.fmt.fallbackLocale
Java 常數	Config.FMT_FALLBACK_LOCALE
設置型態	Locale 或 String
由哪個標籤設置	無
被哪些標籤使用	<fmt:bundle>、<fmt:setBundle>、<fmt:message>、<fmt:formatNumber>、<fmt:parseNumber>、<fmt:formatDate>、<fmt:parseDate

表 7.3 LocalizationContext 的設定與使用

隱含物件	說明
屬性名稱前置	javax.servlet.jsp.jstl.fmt.localizationContext
Java 常數	Config.FMT_LOCALIZATION_CONTEXT
設置型態	LocalizationContext 或 String
由哪個標籤設置	<fmt:setBundle>
被哪些標籤使用	<fmt:message>、<fmt:formatNumber>、<fmt:parseNumber>、<fmt:formatDate>、 <fmt:parseDate>

提示 »» i18n 是個複雜的議題，JSR 52 第 8 單元是不錯的參考文件，建議閱讀，其中對於各標籤的屬性使用也有相關說明。

7.3.4 格式標籤

JSTL 的格式標籤可以針對數字、日期與時間，搭配地區設定或指定的格式進行格式化，也可以進行數字、日期與時間的剖析，以日期、時間格式化為例：

JSTL fmt7.jsp

```
<%@page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt"%>
<jsp:useBean id="now" class="java.util.Date"/>
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
    </head>
    <body>
```

```
        <fmt:formatDate value="${now}"/><br>
        <fmt:formatDate value="${now}" dateStyle="full"/><br>
        <fmt:formatDate value="${now}"
                        type="time" timeStyle="full"/><br>
        <fmt:formatDate value="${now}" pattern="dd.MM.yy"/><br>
        <fmt:timeZone value="GMT+1:00">
            <fmt:formatDate value="${now}" type="both"
                            dateStyle="full" timeStyle="full"/><br>
        </fmt:timeZone>
    </body>
</html>
```

<fmt:formatDate>用來格式化日期，可根據不同地區設定呈現不同格式，這個範例沒有指定地區設定，會根據瀏覽器的 Accept-Language 標頭來決定地區。

dateStyle 屬性指定日期的詳細程度，可設定的值有"default"、"short"、"medium"、"long"、"full"，如果想顯示時間，要在 type 屬性指定"time"或"both"，預設是"date"，**timeStyle** 屬性指定時間的詳細程度，可設定的值有"default"、"short"、"medium"、"long"。

pattern 屬性可自訂格式，格式的指定方式與 java.text.SimpleDateFormat 的指定方式相同，可參考 SimpleDateFormat 的 API 文件說明。

<fmt:timeZone>可指定時區，可使用字串或 java.util.TimeZone 物件指定，字串指定的方式，可參考 TimeZone 的 API 文件說明，如果需要全域的時區指定，可以使用**<fmt:setTimeZone>**標籤，<fmt:formateDate>本身亦有個 **timeZone** 屬性可以設定時區，也可以透過屬性範圍或 Config 物件設定，屬性名稱、常數名稱與會套用時區設定的標籤如下表所示：

表 7.4　時區設定與使用

隱含物件	說明
屬性名稱前置	javax.servlet.jsp.jstl.fmt.timeZone
Java 常數	Config.FMT_TIMEZONE
設置型態	java.util.TimeZone 或 String
由哪個標籤設置	<fmt:setTimeZone>
被哪些標籤使用	<fmt:formatDate>、<fmt:parseDate>

下圖為範例的執行結果：

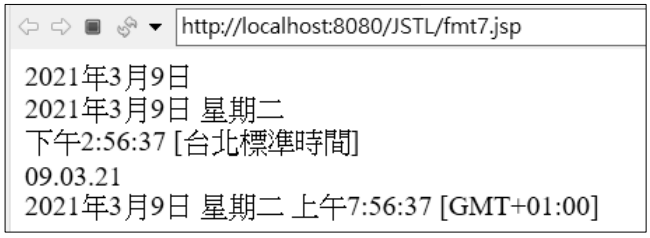

圖 7.7 不同的日期、時間格式設定範例

接著來看一些數字格式化的例子：

JSTL fmt8.jsp

```
<%@page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt"%>
<jsp:useBean id="now" class="java.util.Date"/>
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
    </head>
    <body>
        <fmt:formatNumber value="12345.678"/><br>
        <fmt:formatNumber value="12345.678" type="currency"/><br>
        <fmt:formatNumber value="12345.678"
                          type="currency" currencySymbol="新台幣"/><br>
        <fmt:formatNumber value="12345.678" type="percent"/><br>
        <fmt:formatNumber value="12345.678" pattern="#,#00.0#"/>
    </body>
</html>
```

<fmt:formatNumber>用來格式化數定，可根據不同地區設定呈現不同格式，這個範例沒有指定地區設定，會根據瀏覽器的 Accept-Language 標頭決定地區。

type 屬性可設定的值有"number"（預設）、"currency"、"percent"，指定"currency"時，會將數字依貨幣格式進行格式化，**currencySymbol** 屬性可指定貨幣符號，type 指定為"percent"時，會以百分比格式進行格式化，也可以指定 pattern 屬性，指定格式的方式與 java.text.DecimalFormat 的說明相同，可參考 DecimalFormat 的 API 文件說明。

- 切割字串為字串陣列：`split`
- 連接字串陣列為字串：`join`
- 替換 XML 字元：`escapeXML`

課後練習

實作題

1. 請建立一個首頁，預設用英文顯示訊息，但可以讓使用者選擇使用英文、正體中文或簡體中文。

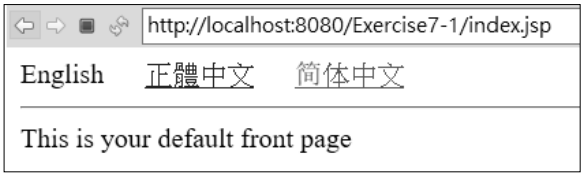

圖 7.14 預設是英文首頁

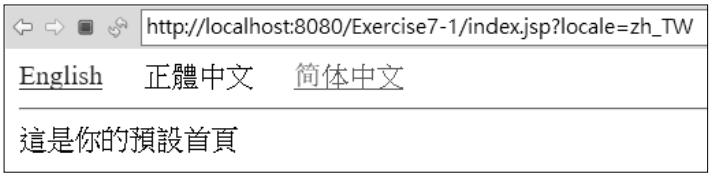

圖 7.15 切換至正體中文首頁

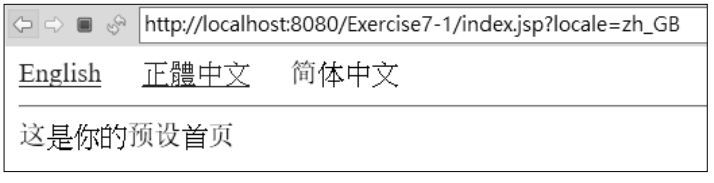

圖 7.16 切換至簡體中文首頁