簡介 JSF

Added by cheetah, last edited by cheetah on May 05, 2005

資料來源：	PmWiki@caterpillar
作者：	caterpillar

Web 應用程式的開發與傳統的單機程式開發在本質上存在著太多的差異，Web 應用程式開發人員至今不可避免的必須處理 HTTP 的細節，而 HTTP 無狀態的（stateless）本質，與傳統應用程式必須維持程式運行過程中的資訊有明顯的違背，再則 Web 應用程式面對網站上不同的使用者同時的存取，其執行緒安全問題以及資料驗證、轉換處理等問題，又是複雜且難以解決的。

另一方面，本質上是靜態的 HTML 與本質上是動態的應用程式又是一項違背，這造成不可避免的，處理網頁設計的美術人員與程式設計人員，必須被彼此加入至視圖元件中的邏輯互相干擾，即便一些視圖呈現邏輯以標籤的方式呈現，試圖展現對網頁設計美術人員的親切，但它終究必須牽涉到相關的流程邏輯。

有很多方案試著解決種種的困境，而各自的著眼點各不相同，有的從程式設計人員的角度來解決，有的從網頁設計人員的角度來解決，各種的框架被提出，所造成的是各種不統一的標籤與框架，為了促進產能的整合開發環境（IDE）難以整合這些標籤與框架，另一方面，開發人員的學習負擔也不斷的加重，他們必須一人瞭解多個角色的工作。

JavaServer Faces 的提出在試圖解決這個問題，它試圖在不同的角度上提供網頁設計人員、應用程式設計人員、元件開發人員解決方案，讓不同技術的人員可以彼此合作又不互相干擾，它綜合了各家廠商現有的技術特點，由 Java Community Process（JCP）團隊研擬出來的一套標準，並在 2004 年三月發表了 JavaServer Faces 1.0 實作成果。

從網頁設計人員的角度來看，JavaServer Faces 提供了一套像是新版本的 HTML 標籤，但它不是靜態的，而是動態的，可以與後端的動態程式結合，但網頁設計人員不需要理會後端的動態部份，網頁設計人員甚至不太需要接觸 JSTL 這類的標籤，也可以動態的展現資料（像是動態的查詢表格內容），JavaServer Faces 提供標準的標籤，這可以與網頁編輯程式結合在一起，另一方面，JavaServer Faces 也允許您自訂標籤。

從應用程式設計人員的角度來看，JavaServer Faces 提供一個與傳統應用程式開發相類似的模型（當然因某些本質上的差異，模型還是稍有不同），他們可以基於事件驅動來開發程式，不必關切 HTTP 的處理細節，如果必須處理一些視覺元件的屬性的話，他們也可以直接在整合開發環境上拖拉這些元件，點選設定元件的屬性，JavaServer Faces 甚至還為應用程式設計人員處理了物件與字串（HTTP 傳送本質上就是字串）間不匹配的轉換問題。

從 UI 元件開發人員的角度來看，他們可以設計通用的 UI 元件，讓應用程式的開發產能提高，就如同在設計 Swing 元件等，UI 開發人員可以獨立開發，只要定義好相關的屬性選項來調整細節，而不用受到網頁設計人員或應用程式設計人員的干擾。

三個角色的知識領域原則上可以互不干擾，根據您的角色，您只要瞭解其中一個知識領域，就可以運用 JavaServer Faces，其它角色的知識領域您可以不用瞭解太多細節。

當然，就其中一個角色單獨來看，JavaServer Faces 隱藏了許多細節，若要全盤瞭解，其實 JavaServer Faces 是複雜的，每一個處理的環境都值得深入探討，所以學習 JavaServer Faces 時，您要選擇的是通盤瞭解，還是從使用的角度來瞭解，這就決定了您學習時所要花費的心力。

要使用 JSF，首先您要先取得 JavaServer Faces 參考實作（JavaServer Faces Reference Implementation），在將來，JSF 會與 Container 整合在一起，屆時您只要下載支援的 Container，就可以使用 JSF 的功能。

請至 JSF 官方網站的 下載區 下載參考實作，在下載壓縮檔並解壓縮之後，將其 lib 目錄下的 jar 檔案複製至您的 Web 應用程式的/WEB-INF/lib 目錄下，另外您還需要 jstl.jar 與 standard.jar 檔案，這些檔案您可以在 sample 目錄下，解壓縮當中的一個範例，在它的/WEB-INF/lib 目錄下找到，將之一併複製至您的 Web 應用程式的/WEB-INF/lib 目錄下，您總共需要以下的檔案：

```
* jsf-impl.jar
   * jsf-api.jar
   * commons-digester.jar
   * commons-collections.jar
   * commons-beanutils.jar
   * jstl.jar
   * standard.jar
```

接下來配置 Web 應用程式的 web.xml，使用 JSF 時，所有的請求都透過 FacesServlet 來處理，您可以如下定義：

- web.xml

web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
  <web-app xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
   version="2.4">

   <description>
      JSF Demo
   </description>
   <display-name>JSF Demo</display-name>
   <servlet>
      <servlet-name>Faces Servlet</servlet-name>
      <servlet-class>
         javax.faces.webapp.FacesServlet
```

```
        </servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <url-pattern>*.faces</url-pattern>
    </servlet-mapping>

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

在上面的定義中，我們將所有.faces 的請求交由 FaceServlet 來處理，FaceServlet 會喚起相對的.jsp 網頁，例如請求是/index.faces 的話，則實際上會喚起/index.jsp 網頁，完成以上的配置，您就可以開始使用 JSF 了。

第一個 JSF 程式

Added by cheetah, last edited by cheetah on May 05, 2005

資料來源：	PmWiki@caterpillar
作者：	caterpillar

現在可以開發一個簡單的程式了，我們將設計一個簡單的登入程式，使用者送出名稱，之後由程式顯示使用者名稱及歡迎訊息。

程式開發人員

先看看應用程式開發人員要作些什麼事，我們撰寫一個簡單的 JavaBean：

UserBean.java

```
package onlyfun.caterpillar;

public class UserBean {
    private String name;

    public void setName(String name) {
        this.name = name;
    }
```

```
    public String getName() {
        return name;
    }
}
```

這個 Bean 將儲存使用者的名稱，編譯好之後放置在/WEB-INF/classes 下。

接下來設計頁面流程，我們將先顯示一個登入網頁/pages/index.jsp，使用者填入名稱並送出表單，之後在/pages/welcome.jsp 中顯示 Bean 中的使用者名稱與歡迎訊息。

為了讓 JSF 知道我們所設計的 Bean 以及頁面流程，我們定義一個/WEB-INF/faces-config.xml：

faces-config.xml

```
<?xml version="1.0"?>
<!DOCTYPE faces-config PUBLIC
 "-//Sun Microsystems, Inc.//DTD JavaServer Faces Config 1.0//EN"
 "http://java.sun.com/dtd/web-facesconfig_1_0.dtd">

<faces-config>
    <navigation-rule>
        <from-view-id>/pages/index.jsp</from-view-id>
        <navigation-case>
            <from-outcome>login</from-outcome>
            <to-view-id>/pages/welcome.jsp</to-view-id>
        </navigation-case>
    </navigation-rule>

    <managed-bean>
        <managed-bean-name>user</managed-bean-name>
         <managed-bean-class>
             onlyfun.caterpillar.UserBean
         </managed-bean-class>
        <managed-bean-scope>session</managed-bean-scope>
    </managed-bean>
</faces-config>
```

在<navigation-rule>中，我們定義了頁面流程，當請求來自<from-view- id>中指定的頁面，並且指定了<navigation-case>中的<from-outcome>為 login 時，則會將請求導向至<to-view-id>所指定的頁面。

在<managed-bean>中我們可以統一管理我們的 Bean，我們設定 Bean 物件的存活範圍是 session，也就是使用者開啟瀏覽器與程式互動過程中都存活。

接下來要告訴網頁設計人員的資訊是，他們可以使用的 Bean 名稱，即<managed-bean-name>中設定的名稱，以及上面所定義的頁面流程。

網頁設計人員

首先網頁設計人員撰寫 index.jsp 網頁：

index.jsp

```
<%@taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
 <%@taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@page contentType="text/html;charset=Big5"%>
<html>
<head>
<title>第一個 JSF 程式</title>
</head>
<body>
   <f:view>
       <h:form>
           <h3>請輸入您的名稱</h3>
           名稱: <h:inputText value="#{user.name}"/><p>
           <h:commandButton value="送出" action="login"/>
       </h:form>
   </f:view>
</body>
</html>
```

我們使用了 JSF 的 core 與 html 標籤庫，core 是有關於 UI 元件的處理，而 html 則是有關於 HTML 的進階標籤。

<f:view>與<html>有類似的作用，當您要開始使用 JSF 元件時，這些元件一定要在<f: view>與</f:view>之間，就如同使用 HTML 時，所有的標籤一定要在<html>與< /html>之間。

html 標籤庫中幾乎都是與 HTML 標籤相關的進階標籤，<h:form>會產生一個表單，我們使用<h:inputText>來顯示 user 這個 Bean 物件的 name 屬性，而<h:commandButton>會產生一個提交按鈕，我們在 action 屬性中指定將根據之前定義的 login 頁面流程中前往 welcome.jsp 頁面。

網頁設計人員不必理會表單傳送之後要作些什麼，他只要設計好歡迎頁面就好了：

welcome.jsp

```
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
 <%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@page contentType="text/html;charset=Big5"%>
<html>
<head>
<title>第一個 JSF 程式</title>
</head>
<body>
   <f:view>
```

```
        <h:outputText value="#{user.name}"/> 您好!
        <h3>歡迎使用 JavaServer Faces! </h3>
    </f:view>
</body>
</html>
```

這個頁面沒什麼需要解釋的了，如您所看到的，在網頁上沒有程式邏輯，網頁設計人員所作的就是遵照頁面流程，使用相關名稱取出資料，而不用擔心實際上程式是如何運作的。

接下來啟動 Container，連接上您的應用程式網址，例如：http://localhost:8080/jsfDemo/pages/index.faces，填入名稱並送出表單，您的歡迎頁面就會顯示了。

簡單的導航 Navigation

Added by cheetah, last edited by cheetah on May 05, 2005

資料來源：	PmWiki@caterpillar
作者：	caterpillar

在 第一個 JSF 程式 中，我們簡單的定義了頁面的流程由 index.jsp 到 welcome.jsp，接下來我們擴充程式，讓它可以根據使用者輸入的名稱與密碼是否正確，決定要顯示歡迎訊息或是將使用者送回原頁面進行重新登入。

首先我們修改一下 UserBean：

UserBean.java

```
package onlyfun.caterpillar;

public class UserBean {
    private String name;
    private String password;
    private String errMessage;

    public void setName(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public void setPassword(String password) {
        this.password = password;
```

```
    }

    public String getPassword() {
        return password;
    }

    public void setErrMessage(String errMessage) {
        this.errMessage = errMessage;
    }

    public String getErrMessage() {
        return errMessage;
    }

    public String verify() {
        if(!name.equals("justin") ||
           !password.equals("123456")) {
            errMessage = "名稱或密碼錯誤";
            return "failure";
        }
        else {
            return "success";
        }
    }
}
```

在 UserBean 中，我們增加了密碼與錯誤訊息屬性，在 verify()方法中，我們檢查使用者名稱與密碼，它傳回一個字串，"failure"表示登入錯誤，並會設定錯誤訊息，而"success"表示登入正確，這個傳回的字串將決定頁面的流程。

接下來我們修改一下 faces-config.xml 中的頁面流程定義：

faces-config.xml

```
<?xml version="1.0"?>
<!DOCTYPE faces-config PUBLIC
 "-//Sun Microsystems, Inc.//DTD JavaServer Faces Config 1.0//EN"
 "http://java.sun.com/dtd/web-facesconfig_1_0.dtd">

<faces-config>
    <navigation-rule>
        <from-view-id>/pages/index.jsp</from-view-id>
        <navigation-case>
            <from-outcome>success</from-outcome>
            <to-view-id>/pages/welcome.jsp</to-view-id>
        </navigation-case>
        <navigation-case>
```

```
        <from-outcome>failure</from-outcome>
        <to-view-id>/pages/index.jsp</to-view-id>
    </navigation-case>
  </navigation-rule>

  <managed-bean>
    <managed-bean-name>user</managed-bean-name>
    <managed-bean-class>
        onlyfun.caterpillar.UserBean
    </managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
  </managed-bean>
</faces-config>
```

根據上面的定義，當傳回的字串是"success"時，將前往 welcome.jsp，如果是"failure"的話，將送回 index.jsp。

接下來告訴網頁設計人員 Bean 名稱與相關屬性，以及決定頁面流程的 verify 名稱，我們修改 index.jsp 如下：

index.jsp

```
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
 <%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@page contentType="text/html;charset=Big5"%>
<html>
<head>
<title>第一個 JSF 程式</title>
</head>
<body>
  <f:view>
    <h:form>
        <h3>請輸入您的名稱</h3>
        <h:outputText value="#{user.errMessage}"/><p>
       名稱: <h:inputText value="#{user.name}"/><p>
       密碼: <h:inputSecret value="#{user.password}"/><p>
        <h:commandButton value="送出"
                    action="#{user.verify}"/>
    </h:form>
  </f:view>
</body>
</html>
```

當要根據 verify 運行結果來決定頁面流程時，action 屬性中使用 JSF Expression Language "#{user.verify}"，如此 JSF 就知道必須根據 verify 傳回的結果來導航頁面。

<h:outputText>可以取出指定的 Bean 之屬性值，當使用者因驗證錯誤而被送回原頁面時，這個

錯誤訊息就可以顯示在頁面上。

航規則設置

在 JSF 中是根據 faces-config.xml 中<navigation-rule>設定，以決定在符合的條件成立時，該連結至哪一個頁面，一個基本的設定如下：

```
....
   <navigation-rule>
      <from-view-id>/pages/index.jsp</from-view-id>
      <navigation-case>
         <from-outcome>success</from-outcome>
         <to-view-id>/pages/welcome.jsp</to-view-id>
      </navigation-case>
      <navigation-case>
         <from-outcome>failure</from-outcome>
         <to-view-id>/pages/index.jsp</to-view-id>
      </navigation-case>
   </navigation-rule>
....
```

對於 JSF，每一個視圖（View）都有一個獨特的識別（identifier），稱之為 View ID，在 JSF 中的 View ID 是從 Web 應用程式的環境相對路徑開始計算，設定時都是以/作為開頭，如果您請求時的路徑是/pages/index.faces，則 JSF 會將副檔名改為/pages/index.jsp，以此作為 view-id。

在<navigation-rule>中的<from-view-id>是個選擇性的定義，它規定了來源頁面的條件，<navigation-case>中定義各種導覽條件，<from-outcome>定義當表單結果符合的條件時，各自改導向哪一個目的頁面，目的頁面是在<to-view-id>中定義。

您還可以在<navigation-case>中加入<from-action>，進一步規範表單結果必須根據哪一個動作方法（action method），當中是使用 JSF Expression Language 來設定，例如：

```
....
   <navigation-rule>
      <from-view-id>/pages/index.jsp</from-view-id>
      <navigation-case>
         <from-action>#{user.verify}</from-action>
         <from-outcome>success</from-outcome>
         <to-view-id>/pages/welcome.jsp</to-view-id>
      </navigation-case>
```

```
      ....
   </navigation-rule>
....
```

在導航時，預設都是使用 forward 的方式，您可以在<navigation-case>中加入一個<redirect/>，讓 JSF 發出讓瀏覽器重新導向（redirect）的 header，讓瀏覽器主動要求新網頁，例如：

```
....
   <navigation-rule>
      <from-view-id>/pages/index.jsp</from-view-id>
      <navigation-case>
         <from-outcome>success</from-outcome>
         <to-view-id>/pages/welcome.jsp</to-view-id>
         <redirect/>
      </navigation-case>
      ....
   </navigation-rule>
....
```

您的來源網頁可能是某個特定模組，例如在/admin/下的頁面，您可以在<from-view-id>中使用 wildcards，也就是使用 * 字元，例如：

```
....
   <navigation-rule>
      <from-view-id>/admin/*</from-view-id>
      <navigation-case>
         <from-action>#{user.verify}</from-action>
         <from-outcome>success</from-outcome>
         <to-view-id>/pages/welcome.jsp</to-view-id>
      </navigation-case>
      ....
   </navigation-rule>
....
```

在上面的設定中，只要來源網頁是從/admin 來的，都可以開始測試接下來的<navigation-case>。

<from-view-id>如果沒有設定，表示來源網頁不作限制，您也可以使用 * 顯式的在定義檔中表明，例如：

```
....
   <navigation-rule>
```

```
      <from-view-id>/*</from-view-id>

      <navigation-case>
      ....
   </navigation-rule>
....
```

或者是這樣：

```
....
   <navigation-rule>

      <from-view-id>*</from-view-id>

      <navigation-case>
      ....
   </navigation-rule>
....
```

JSF Expression Language

JSF Expression Language 搭配 JSF 標籤來使用，是用來存取資料物件的一個簡易語言。

JSF EL 是以#開始，將變數或運算式放置在

Unknown macro: { 與 }

之間，例如：

```
#{someBeanName}
```

變數名稱可以是 faces-config.xml 中定義的名稱，如果是 Bean 的話，可以透過使用 '.' 運算子來存取它的屬性，例如：

```
...
<f:view>

   <h:outputText value="#{userBean.name}"/>
</f:view>
...
```

在 JSF 標籤的屬性上，" 與 " （或'與'）之間如果含有 EL，則會加以運算，您也可以這麼使用它：

```
...
<f:view>

   名稱, 年齡: <h:outputText

               value="#{userBean.name}, #{userBean.age}"/>
```

```
</f:view>
...
```

一個執行的結果可能是這樣顯示的：

```
名稱, 年齡：Justin, 29
```

EL 的變數名也可以程式執行過程中所宣告的名稱，或是 JSF EL 預設的隱含物件，例如下面的程式使用 param 隱含物件來取得使用者輸入的參數：

```
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
 <%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@page contentType="text/html; charset=Big5"%>

<html>
<head>
<title></title>
</head>
<body>
<f:view>
   <b> 您好, <h:outputText value="#{param.name}"/> </b>
</f:view>

</body>
</html>
```

param 是 JSF EL 預設的隱含物件變數，它代表 request 所有參數的集合，實際是一個 java.util.Map 型態物件，JSF 所提供的隱含物件，大致上對應於 JSP 隱含物件，不過 JSF 隱含物件移除了 pageScope 與 pageContext，而增加了 facesContext 與 view，它們分別對應於 javax.faces.context.FacesContext 與 javax.faces.component.UIViewRoot。

對於 Map 型態物件，我們可以使用 '.' 運算子指定 key 值來取出對應的 value，也可以使用 [與] 來指定，例如：

```
...

<f:view>

   <b> 您好, <h:outputText value="#{param['name']}"/> </b>
</f:view>
...
```

在 [與] 之間，也可以放置其它的變數值，例如：

```
...

<f:view>

   <h:outputText value="#{someBean.someMap[user.name]}"/>
```

```
</f:view>
...
```

如果變數是 List 型態或陣列的話，則可以在 [] 中指定索引，例如：

```
....
<f:view>
    <h:outputText value="#{someBean.someList[0]}"/>
    <h:outputText value="#{someBean.someArray[1]}"/>
    <h:outputText
         value="#{someBean.someListOrArray[user.age]}"/>
</f:view>
....
```

您也可以指定字面常數，對於 true、false、字串、數字，JSF EL 會嘗試進行轉換，例如：

```
....
<h:outputText value="#{true}"/>
....

<h:outputText value="#{'This is a test'}"/>
....
```

如果要輸出字串，必須以單引號 ' 或雙引數 " 括住，如此才不會被認為是變數名稱。

在宣告變數名稱時，要留意不可與 JSF 的保留字或關鍵字同名，例如不可取以下這些名稱：

```
true false null div mod and or not eq ne lt gt le ge instanceof empty
```

使用 EL，您可以直接實行一些算術運算、邏輯運算與關係運算，其使用就如同在一般常見的程式語言中之運算。

算術運算子有：加法 (+)，減法 (-)，乘法 (*)，除法 (/ or div) 與餘除 (% or mod) 。下面是算術運算的一些例子：

運算式	結果
#{1}	1
#{1 + 2}	3
#{1.2 + 2.3}	3.5
#{1.2E4 + 1.4}	12001.4
#{-4 - 2}	-6

#{21 * 2}	42
#{3/4}	0.75
#{3 div 4}	0.75，除法
#{3/0}	Infinity
#{10%4}	2
#{10 mod 4}	2，也是餘除
#{(1==2) ? 3 : 4}	4

如同在 Java 語法一樣（ expression ? result1 : result2）是個三元運算，expression 為 true 顯示 result1，false 顯示 result2。

邏輯運算有：and(或&&)、or(或!!)、not(或!)。一些例子為：

運算式	結果
#{true and false}	false
#{true or false}	true
#{not true}	false

關係運算有：小於 Less-than (< or lt)、大於 Greater-than (> or gt)、小於或等於 Less-than-or-equal (<= or le)、大於或等於 Greater-than-or-equal (>= or ge)、等於 Equal (== or eq)、不等於 Not Equal (!= or ne)，由英文名稱可以得到 lt、gt 等運算子之縮寫詞，以下是 Tomcat 的一些例子：

運算式	結果
#{1 < 2}	true
#{1 lt 2}	true
#{1 > (4/2)}	false
#{1 > (4/2)}	false
#{4.0 >= 3}	true
#{4.0 ge 3}	true
#{4 <= 3}	false

#{4 le 3}	false
#{100.0 == 100}	true
#{100.0 eq 100}	true
#{(10*10) != 100}	false
#{(10*10) ne 100}	false

左邊是運算子的使用方式，右邊的是運算結果，關係運算也可以用來比較字元或字串，按字典順序來決定比較結果，例如：

運算式	結果
#{'a' < 'b'}	true
#{'hip' > 'hit'}	false
#{'4' > 3}	true

EL 運算子的執行優先順序與 Java 運算子對應，如果有疑慮的話，也可以使用括號()來自行決定先後順序。

國際化訊息

JSF 的國際化（Internnationalization）訊息處理是基於 Java 對國際化的支援，您可以在一個訊息資源檔中統一管理訊息資源，資源檔的名稱是.properties，而內容是名稱與值的配對，例如：

- messages.properties

```
titleText=JSF Demo

hintText=Please input your name and password

nameText=name

passText=password

commandText=Submit
```

資源檔名稱由 basename 加上語言與地區來組成，例如：

```
* basename.properties

   * basename_en.properties
```

```
   * basename_zh_TW.properties
```

沒有指定語言與地區的 basename 是預設的資源檔名稱，JSF 會根據瀏覽器送來的 Accept-Language header 中的內容來決定該使用哪一個資源檔名稱，例如：

Accept-Language: zh_TW, en-US, en

如果瀏覽器送來這些 header，則預設會使用繁體中文，接著是美式英文，再來是英文語系，如果找不到對應的訊息資源檔，則會使用預設的訊息資源檔。

由於訊息資源檔必須是 ISO-8859-1 編碼，所以對於非西方語系的處理，必須先將之轉換為 Java Unicode Escape 格式，例如您可以先在訊息資源檔中寫下以下的內容：

- messages_zh_TW.txt

```
titleText=JSF 示範

hintText=請輸入名稱與密碼

nameText=名稱

passText=密碼

commandText=送出
```

然後使用 JDK 的工具程式 native2ascii 來轉換，例如：

native2ascii -encoding Big5 messages_zh_TW.txt messages_zh_TW.properties

轉換後的內容會如下：

- messages_zh_TW.properties

```
titleText=JSF\u793a\u7bc4

hintText=\u8acb\u8f38\u5165\u540d\u7a31\u8207\u5bc6\u78bc

nameText=\u540d\u7a31

passText=\u5bc6\u78bc

commandText=\u9001\u51fa
```

接下來您可以使用<f:loadBundle>標籤來指定載入訊息資源，一個例子如下：

- index.jsp

index.jsp

```
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
 <%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@page contentType="text/html;charset=UTF8"%>

<f:view>
<f:loadBundle basename="messages" var="msgs"/>

<html>
<head>
<title><h:outputText value="#{msgs.titleText}"/></title>
</head>
<body>

   <h:form>
       <h3><h:outputText value="#{msgs.hintText}"/></h3>
       <h:outputText value="#{msgs.nameText}"/>:
              <h:inputText value="#{user.name}"/><p>
       <h:outputText value="#{msgs.passText}"/>:
              <h:inputSecret value="#{user.password}"/><p>
       <h:commandButton value="#{msgs.commandText}"
                     actionListener="#{user.verify}"
                     action="#{user.outcome}"/>
  </h:form>

</body>
</html>

</f:view>
```

如此一來，如果您的瀏覽器預設接受 zh_TW 語系的話，則頁面上就可以顯示中文，否則預設將以英文顯示，也就是 messages.properties 的內容，為了能顯示多國語系，我們設定網頁編碼為 UTF8。

<f:view>可以設定 locale 屬性，直接指定所要使用的語系，例如：

```
<f:view locale="zh_TW">
 <f:loadBundle basename="messages" var="msgs"/>
```

直接指定以上的話，則會使用繁體中文來顯示，JSF 會根據<f:loadBundle>的 basename 屬性加上<f:view>的 locale 屬性來決定要使用哪一個訊息資源檔，就上例而言，就是使用 messages_zh_TW.properties，如果設定為以下的話，就會使用 messages_en.properties：

```
<f:view locale="en">
 <f:loadBundle basename="messages" var="msgs"/>
```

您也可以在 faces-config.xml 中設定語系，例如：

```
<faces-config>

   <application>

      <local-config>

         <default-locale>en</default-locale>
         <supported-locale>zh_TW</supported-locale>
      </local-config>
   </application>


.....
</faces-config>
```

在<local-config>一定有一個<default-locale>，而<supported-locale>可以有好幾個，這告訴 JSF 您的應用程式支援哪些語系。

當然，如果您可以提供一個選項讓使用者選擇自己的語系會是更好的方式，例如根據 user 這個 Bean 的 locale 屬性來決定頁面語系：

```
<f:view locale="#{user.locale}">
 <f:loadBundle basename="messages" var="msgs"/>
```

在頁面中設定一個表單，可以讓使用者選擇語系，例如設定單選鈕：

```
<h:selectOneRadio value="#{user.locale}">
    <f:selectItem itemValue="zh_TW"
                itemLabel="#{msgs.zh_TWText}"/>
    <f:selectItem itemValue="en"
                itemLabel="#{msgs.enText}"/>
 </h:selectOneRadio>
```

Backing Beans

JSF 使用 JavaBean 來達到程式邏輯與視圖分離的目的，在 JSF 中的 Bean 其角色是屬於 Backing Bean，又稱之為 Glue Bean，其作用是在真正的業務邏輯 Bean 及 UI 元件之間搭起橋樑，在 Backing Bean 中會呼叫業務邏輯 Bean 處理使用者的請求，或者是將業務處理結果放置其中，等待 UI 元件取出當中的值並顯示結果給使用者。

JSF 將 Bean 的管理集中在 faces-config.xml 中，一個例子如下：

```
....

 <managed-bean>

   <managed-bean-name>user</managed-bean-name>
```

```
      <managed-bean-class>
          onlyfun.caterpillar.UserBean
      </managed-bean-class>
   <managed-bean-scope>session</managed-bean-scope>
</managed-bean>
....
```

這個例子我們在 第一個 JSF 程式 看過，<managed-bean-class>設定所要使用的 Bean 類別，<managed-bean-name>設定之名稱，可供我們在 JSF 頁面上使用 Expression Language 來取得或設定 Bean 的屬性，例如：

```
<h:inputText value="#{user.name}"/>
```

<managed-bean-scope>設定 Bean 的存活範圍，您可以設定為 request、session 與 application，設定為 request 時，Bean 的存活時間為請求階最，設定為 session 則在使用者應用程式交互開始，直到關閉瀏覽器或顯式的結束會話為止（例如登出程式），設定為 application 的話，則 Bean 會一直存活，直到應用程式關閉為止。

您還可以將存活範圍設定為 none，當設定為 none 時會在需要的時候生成一個新的 Bean，例如您在一個 method 中想要生成一個臨時的 Bean，就可以將之設定為 none。

在 JSF 頁面上要取得 Bean 的屬性，是使用 JSF 表示語言 (Expression Language)，要注意到的是，JSF 表示語言是寫成 #{expression}，而 JSP 表示語言 是寫成 ${expression}，因為表示層可能是使用 JSP，所以必須特別區分，另外要注意的是，JSF 的標籤上之屬性設定時，只接受 JSF 表示語言。

Beans 的組態與設定

JSF 預設會讀取 faces-config.xml 中關於 Bean 的定義，如果想要自行設置定義檔的名稱，我們是在 web.xml 中提供 javax.faces.CONFIG_FILES 參數，例如：

```
<web-app>
  <context-param>
    <param-name>javax.faces.CONFIG_FILES</param-name>
    <param-value>/WEB-INF/beans.xml</param-value>
  </context-param>
```

```
  ...
</web-app>
```

定義檔可以有多個，中間以 "," 區隔，例如：

```
/WEB-INF/navigation.xml,/WEB-INF/beans.xml
```

一個 Bean 最基本要定義 Bean 的名稱、類別與存活範圍，例如：

```
....
 <managed-bean>
   <managed-bean-name>user</managed-bean-name>
       <managed-bean-class>
          onlyfun.caterpillar.UserBean
       </managed-bean-class>
   <managed-bean-scope>session</managed-bean-scope>
 </managed-bean>
 ....
```

如果要在其它類別中取得 Bean 物件，則可以先取得 javax.faces.context.FacesContext，它代表了 JSF 目前的執行環境物件，接著嘗試取得 javax.faces.el.ValueBinding 物件，從中取得指定的 Bean 物件，例如：

```
FacesContext context = FacesContext.getCurrentInstance();
ValueBinding binding =
   context.getApplication().createValueBinding("#{user}");
UserBean user = (UserBean) binding.getValue(context);
```

如果只是要嘗試取得 Bean 的某個屬性，則可以如下：

```
FacesContext context = FacesContext.getCurrentInstance();
ValueBinding binding =
   context.getApplication().createValueBinding(
                                "#{user.name}");
String name = (String) binding.getValue(context);
```

如果有必要在啟始 Bean 時，自動設置屬性的初始值，則可以如下設定：

```
....
<managed-bean>
   <managed-bean-name>user</managed-bean-name>
       <managed-bean-class>
          onlyfun.caterpillar.UserBean
       </managed-bean-class>
   <managed-bean-scope>session</managed-bean-scope>
   <managed-property>
       <property-name>name</property-name>
       <value>caterpillar</value>
   </managed-property>
   <managed-property>
       <property-name>password</property-name>
       <value>123456</value>
   </managed-property>
</managed-bean>
....
```

如果要設定屬性為 null 值，則可以使用<null-value/>標籤，例如：

```
....
   <managed-property>
       <property-name>name</property-name>
       <null-value/>
   </managed-property>
   <managed-property>
       <property-name>password</property-name>
       <null-value/>
   </managed-property>
....
```

當然，您的屬性不一定是字串值，也許會是 int、float、boolean 等等型態，您可以設定<value>值時指定這些值的字串名稱，JSF 會嘗試進行轉換，例如設定為 true 時，會嘗試使用 Boolean.valueOf()方法轉換為 boolean 的 true，以下是一些可能進行的轉換：

型態	轉換

short、int、long、float、double、byte，或相應的 Wrapper 類別	嘗試使用 Wrapper 的 valueOf()進行轉換，如果沒有設置，則設為 0
boolean 或 Boolean	嘗試使用 Boolean.valueOf()進行轉換，如果沒有設置，則設為 false
char 或 Character	取設置的第一個字元，如果沒有設置，則設為 0
String 或 Object	即設定的字串值，如果沒有設定，則為空字串 new String("")

您也可以將其它產生的 Bean 設定給另一個 Bean 的屬性，例如：

```
....
 <managed-bean>
    <managed-bean-name>user</managed-bean-name>
        <managed-bean-class>
            onlyfun.caterpillar.UserBean
        </managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
 </managed-bean>


 <managed-bean>
    <managed-bean-name>other</managed-bean-name>
        <managed-bean-class>
            onlyfun.caterpillar.OtherBean
        </managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
    <managed-property>
      <property-name>user</property-name>
      <value>#{user}</value>
    </managed-property>
 </managed-bean>
 ....
```

在上面的設定中，在 OtherBean 中的 user 屬性，接受一個 UserBean 型態的物件，我們設定為

前一個名稱為 user 的 UserBean 物件。

Beans 上的 List, Map

資料來源：	PmWiki@caterpillar
作者：	caterpillar

如果您的 Bean 上有接受 List 或 Map 型態的屬性，則您也可以在組態檔案中直接設定這些屬性的值，一個例子如下：

```
....
 <managed-bean>
    <managed-bean-name>someBean</managed-bean-name>
    <managed-bean-class>
       onlyfun.caterpillar.SomeBean
    </managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>

    <managed-property>
        <property-name>someProperty</property-name>
        <list-entries>
            <value-class>java.lang.Integer</value-class>
            <value>1</value>
            <value>2</value>
            <value>3</value>
        </list-entries>
    </managed-property>
 </managed-bean>
 ....
```

這是一個設定接受 List 型態的屬性，我們使用<list-entries>標籤指定將設定一個 List 物件，其中<value-class>指定將存入 List 的型態，而<value>指定其值，如果是基本型態，則會嘗試使用指定的 <value-class>來作 Wrapper 類別。

設定 Map 的話，則是使用<map-entries>標籤，例如：

```
....
```

```
<managed-bean>

   <managed-bean-name>someBean</managed-bean-name>

   <managed-bean-class>

      onlyfun.caterpillar.SomeBean

   </managed-bean-class>

   <managed-bean-scope>session</managed-bean-scope>



   <managed-property>

       <property-name>someProperty</property-name>

       <map-entries>

           <value-class>java.lang.Integer</value-class>
           <map-entry>
              <key>someKey1</key>
              <value>100</value>
           </map-entry>
           <map-entry>
              <key>someKey2</key>
              <value>200</value>
           </map-entry>
       </map-entries>
   </managed-property>
</managed-bean>
....
```

由於 Map 物件是以 key-value 對的方式來存入，所以我們在每一個<map-entry>中使用<key>與<value>標籤來分別指定。

您也可以直接像設定 Bean 一樣，設定一個 List 或 Map 物件，例如在 JSF 附的範例中，有這樣的設定：

```
....
  <managed-bean>

   <description>

     Special expense item types

   </description>

   <managed-bean-name>specialTypes</managed-bean-name>

   <managed-bean-class>
```

```
      java.util.TreeMap

  </managed-bean-class>

  <managed-bean-scope>application</managed-bean-scope>

  <map-entries>

    <value-class>java.lang.Integer</value-class>
    <map-entry>
      <key>Presentation Material</key>
      <value>100</value>
    </map-entry>
    <map-entry>
      <key>Software</key>
      <value>101</value>
    </map-entry>
    <map-entry>
      <key>Balloons</key>
      <value>102</value>
    </map-entry>
  </map-entries>
 </managed-bean>
....
```

而範例中另一個設定 List 的例子如下：

```
....
 <managed-bean>

  <managed-bean-name>statusStrings</managed-bean-name>

  <managed-bean-class>

      java.util.ArrayList

  </managed-bean-class>

  <managed-bean-scope>request</managed-bean-scope>

  <list-entries>

    <null-value/>
    <value>Open</value>
    <value>Submitted</value>
    <value>Accepted</value>
    <value>Rejected</value>
  </list-entries>
 </managed-bean>
....
```

標準轉換器

Web 應用程式與瀏覽器之間是使用 HTTP 進行溝通，所有傳送的資料基本上都是字串文字，而 Java 應用程式本身基本上則是物件，所以物件資料必須經由轉換傳送給瀏覽器，而瀏覽器送來的資料也必須轉換為物件才能使用。

JSF 定義了一系列標準的轉換器（Converter），對於基本資料型態（primitive type）或是其 Wrapper 類別，JSF 會使用 javax.faces.Boolean、javax.faces.Byte、 javax.faces.Character、javax.faces.Double、javax.faces.Float、 javax.faces.Integer、javax.faces.Long、javax.faces.Short 等自動進行轉換，對於 BigDecimal、BigInteger，則會使用 javax.faces.BigDecimal、javax.faces.BigInteger 自動進行轉換。

至於 DateTime、Number，我們可以使用<f:convertDateTime>、<f:convertNumber>標籤進行轉換，它們各自提供有一些簡單的屬性，可以讓我們在轉換時指定一些轉換的格式細節。

來看個簡單的例子，首先我們定義一個簡單的 Bean：

- UserBean.java

UserBean.java

```
package onlyfun.caterpillar;

import java.util.Date;

public class UserBean {
   private Date date = new Date();

   public Date getDate() {
       return date;
   }

   public void setDate(Date date) {
       this.date = date;
   }
}
```

這個 Bean 的屬性接受 Date 型態的參數，按理來說，接收到 HTTP 傳來的資料中若有相關的日期資訊，我們必須剖析這個資訊，再轉換為 Date 物件，然而我們可以使用 JSF 的標準轉換器來協助這項工作，例如：

- index.jsp

index.jsp

```
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
 <%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
```

```
<%@page contentType="text/html;charset=Big5"%>

<f:view>

<html>
<head>
<title>轉換器示範</title>
</head>
<body>

設定的日期是:
        <b>
        <h:outputText value="#{user.date}">
           <f:convertDateTime pattern="dd/MM/yyyy"/>
        </h:outputText>
        </b>

   <h:form>
      <h:inputText id="dateField" value="#{user.date}">
         <f:convertDateTime pattern="dd/MM/yyyy"/>
      </h:inputText>
      <h:message for="dateField" style="color:red"/>
      <br>
      <h:commandButton value="送出" action="show"/>
   </h:form>
</body>
</html>

</f:view>
```

在<f:convertDateTime>中，我們使用 pattern 指定日期的樣式為 dd/MM/yyyy，即「日/月/西元」格式，如果轉換錯誤，則<h:message>可以顯示錯誤訊息，for 屬性參考至<h:inputText> 的 id 屬性，表示將有關 dateField 的錯誤訊息顯示出來。

假設 faces-config.xml 是這樣定義的：

faces-config.xml

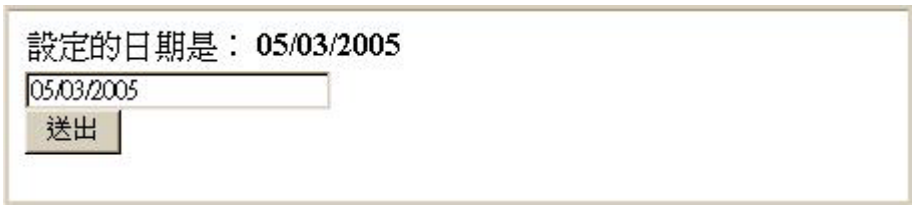
```
<?xml version="1.0"?>
<!DOCTYPE faces-config PUBLIC
 "-//Sun Microsystems, Inc.//DTD JavaServer Faces Config 1.0//EN"
 "http://java.sun.com/dtd/web-facesconfig_1_0.dtd">

<faces-config>
   <navigation-rule>
      <from-view-id>/*</from-view-id>
      <navigation-case>
```

```
            <from-outcome>show</from-outcome>
            <to-view-id>/pages/index.jsp</to-view-id>
        </navigation-case>
    </navigation-rule>

    <managed-bean>
        <managed-bean-name>user</managed-bean-name>
        <managed-bean-class>
            onlyfun.caterpillar.UserBean
        </managed-bean-class>
        <managed-bean-scope>session</managed-bean-scope>
    </managed-bean>
</faces-config>
```

首次連上頁面時顯示的畫面如下：



如您所看到的，轉換器自動依 pattern 設定的樣式將 Date 物件格式化了，當您依格式輸入資料並送出後，轉換器也會自動將您輸入的資料轉換為 Date 物件，如果轉換時發生錯誤，則會出現以下的訊息：

<f:convertDateTime>標籤還有幾個可用的屬性，您可以參考 Tag Library Documentation 的說明，而依照類似的方式，您也可以使用<f:convertNumber>來轉換數值。

您還可以參考 Using the Standard Converters 這篇文章中有關於標準轉換器的說明。

自訂轉換器

資料來源：	PmWiki@caterpillar
作者：	caterpillar

除了使用標準的轉換器之外，您還可以自行定製您的轉換器，您可以實作

javax.faces.convert.Converter 介面，這個介面有兩個要實作的方法：

```
public Object getAsObject(FacesContext context,
                          UIComponent component,
                          String str);
public String getAsString(FacesContext context,
                          UIComponent component,
                          Object obj);
```

簡單的說，第一個方法會接收從客戶端經由 HTTP 傳來的字串資料，您在第一個方法中將之轉換為您的自訂物件，這個自訂物件將會自動設定給您指定的 Bean 物件；第二個方法就是將從您的 Bean 物件得到的物件轉換為字串，如此才能藉由 HTTP 傳回給客戶端。

直接以一個簡單的例子來作說明，假設您有一個 User 類別：

- User.java

User.java

```
package onlyfun.caterpillar;

public class User {
   private String firstName;
   private String lastName;

   public String getFirstName() {
       return firstName;
   }

   public void setFirstName(String firstName) {
       this.firstName = firstName;
   }

   public String getLastName() {
       return lastName;
   }

   public void setLastName(String lastName) {
       this.lastName = lastName;
   }
}
```

這個 User 類別是我們轉換器的目標物件，而您有一個 GuestBean 類別：

- GuestBean.java

GuestBean.java

```
package onlyfun.caterpillar;

public class GuestBean {
   private User user;

   public void setUser(User user) {
      this.user = user;
   }

   public User getUser() {
      return user;
   }
}
```

這個 Bean 上的屬性直接傳回或接受 User 型態的參數，我們來實作一個簡單的轉換器，為 HTTP 字串與 User 物件進行轉換：

- UserConverter.java

UserConverter.java

```
package onlyfun.caterpillar;

import javax.faces.component.UIComponent;
import javax.faces.context.FacesContext;
import javax.faces.convert.Converter;
import javax.faces.convert.ConverterException;

public class UserConverter implements Converter {
   public Object getAsObject(FacesContext context,
                             UIComponent component,
                             String str)
                               throws ConverterException {
      String[] strs = str.split(",");

      User user = new User();

      try {
         user.setFirstName(strs[0]);
         user.setLastName(strs[1]);
      }
      catch(Exception e) {
         // 轉換錯誤，簡單的丟出例外
         throw new ConverterException();
      }
```

```
        return user;
    }

    public String getAsString(FacesContext context,
                              UIComponent component,
                              Object obj)
                                 throws ConverterException {
        String firstName = ((User) obj).getFirstName();
        String lastName = ((User) obj).getLastName();

        return firstName + "," + lastName;
    }
}
```

實作完成這個轉換器，我們要告訴 JSF 這件事，這是在 faces-config.xml 中完成註冊：

- faces-config.xml

faces-config.xml

```
<?xml version="1.0"?>
<!DOCTYPE faces-config PUBLIC
 "-//Sun Microsystems, Inc.//DTD JavaServer Faces Config 1.0//EN"
 "http://java.sun.com/dtd/web-facesconfig_1_0.dtd">

<faces-config>
    <navigation-rule>
        <from-view-id>/*</from-view-id>
        <navigation-case>
            <from-outcome>show</from-outcome>
            <to-view-id>/pages/index.jsp</to-view-id>
        </navigation-case>
    </navigation-rule>

    <converter>
        <converter-id>onlyfun.caterpillar.User</converter-id>
        <converter-class>
            onlyfun.caterpillar.UserConverter
        </converter-class>
    </converter>

    <managed-bean>
        <managed-bean-name>guest</managed-bean-name>
        <managed-bean-class>
            onlyfun.caterpillar.GuestBean
```

```
      </managed-bean-class>
      <managed-bean-scope>session</managed-bean-scope>
   </managed-bean>
</faces-config>
```

註冊轉換器時，需提供轉換器識別（Converter ID）與轉換器類別，接下來要在 JSF 頁面中使用轉換器的話，就是指定所要使用的轉換器識別，例如：

- index.jsp

index.jsp

```
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
 <%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@page contentType="text/html;charset=Big5"%>

<f:view>

<html>
<head>
<title>自訂轉換器</title>
</head>
<body>

Guest 名稱是：<b>
        <h:outputText value="#{guest.user}"
               converter="onlyfun.caterpillar.User"/>
        </b>

   <h:form>
      <h:inputText id="userField"
                 value="#{guest.user}"
                 converter="onlyfun.caterpillar.User"/>
      <h:message for="userField" style="color:red"/>
      <br>
      <h:commandButton value="送出" action="show"/>
   </h:form>
</body>
</html>

</f:view>
```

您也可以<f:converter>標籤並使用 converterId 屬性來指定轉換器，例如：

```
<h:inputText id="userField" value="#{guest.user}">
    <f:converter converterId="onlyfun.caterpillar.User"/>
</h:inputText>
```

除了向 JSF 註冊轉換器之外，還有一個方式可以不用註冊，就是直接在 Bean 上提供一個取得轉換器的方法，例如：

- GuestBean.java

GuestBean.java

```
package onlyfun.caterpillar;

import javax.faces.convert.Converter;

public class GuestBean {
   private User user;
   private Converter converter = new UserConverter();

   public void setUser(User user) {
       this.user = user;
   }

   public User getUser() {
       return user;
   }

   public Converter getConverter() {
       return converter;
   }
}
```

之後可以直接結合 JSF Expression Language 來指定轉換器：

```
<h:inputText id="userField"
    value="#{guest.user}"
    converter="#{guest.converter}"/>
```

標準驗證器

當應用程式要求使用者輸入資料時，必然考慮到使用者輸入資料之正確性，對於使用者的輸入必須進行檢驗，檢驗必要的兩種驗證是語法檢驗（Synatic Validation）與語意檢驗（Semantic Validation）。

語法檢驗是要檢查使用者輸入的資料是否合乎我們所要求的格式，最基本的就是檢查使用者是否填入了欄位值，或是欄位值的長度、大小值等等是否符合要求。語意檢驗是在語法檢驗之後，在格式符合需求之後，我們進一步驗證使用者輸入的資料語意上是否正確，例如檢查使用者的名稱與密碼是否匹配。

在 簡單的導航 (Navigation) 中，我們對使用者名稱與密碼檢查是否匹配，這是語意檢驗，我

們可以使用 JSF 所提供的標準驗證器，為其加入語法檢驗，例如：

- index.jsp

index.jsp

```
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
 <%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@page contentType="text/html;charset=Big5"%>
<html>
<head>
<title>驗證器示範</title>
</head>
<body>
   <f:view>
       <h:messages layout="table" style="color:red"/>
       <h:form>
           <h3>請輸入您的名稱</h3>
           <h:outputText value="#{user.errMessage}"/><p>
          名稱: <h:inputText value="#{user.name}"
                          required="true"/><p>
          密碼: <h:inputSecret value="#{user.password}"
                          required="true">
            <f:validateLength minimum="6"/>
        </h:inputSecret><p>
           <h:commandButton value="送出"
                         action="#{user.verify}"/>
       </h:form>
   </f:view>
</body>
</html>
```

在<h:inputText>、</h:inputSecret>中，我們設定了 required 屬性為 true，這表示這個欄位一定要輸入值，我們也在</h:inputSecret>設定了<f: validateLength>，並設定其 minimum 屬性為 6，這表示這個欄位最少需要 6 個字元。

這一次在錯誤訊息的顯示上，我們使用<h:messages>標籤，當有驗證錯誤發生時，相關的錯誤訊息會收集起來，使用<h:messages>標籤可以一次將所有的錯誤訊息顯示出來。

下面是一個驗證錯誤的訊息顯示：

JSF 提供了三種標準驗證器：<f:validateDoubleRange>、<f:validateLongRange>、<f:validateLength>，您可以分別查詢它們的 Tag Library Documentation，瞭解他們有哪些屬性可以使用，或者是參考 Using the Standard Validators 這篇文章中有關於標準驗證器的說明。

自訂驗證器

您可以自訂自己的驗證器，所需要的是實作 javax.faces.validator.Validator 介面，例如我們實作一個簡單的密碼驗證器，檢查字元長度，以及密碼中是否包括字元與數字：

- PasswordValidator.java

PasswordValidator.java

```
package onlyfun.caterpillar;

import javax.faces.application.FacesMessage;
import javax.faces.component.UIComponent;
import javax.faces.context.FacesContext;
import javax.faces.validator.Validator;
import javax.faces.validator.ValidatorException;

public class PasswordValidator implements Validator {
   public void validate(FacesContext context,
                        UIComponent component,
                        Object obj)
          throws ValidatorException {
       String password = (String) obj;

       if(password.length() < 6) {
           FacesMessage message = new FacesMessage(
               FacesMessage.SEVERITY_ERROR,
               "字元長度小於 6",
               "字元長度不得小於 6");
```

```
            throw new ValidatorException(message);
        }

        if(!password.matches(".+[0-9]+")) {
            FacesMessage message = new FacesMessage(
                  FacesMessage.SEVERITY_ERROR,
                  "密碼必須包括字元與數字",
                  "密碼必須是字元加數字所組成");
            throw new ValidatorException(message);
        }
    }
}
```

您要實作 javax.faces.validator.Validator 介面中的 validate()方法，如果驗證錯誤，則丟出一個 ValidatorException，它接受一個 FacesMessage 物件，這個物件接受三個參數，分別表示訊息的嚴重程度（INFO、 WARN、ERROR、FATAL）、訊息概述與詳細訊息內容，這些訊息將可以使用<h:messages>或<h: message>標籤顯示在頁面上。

接下來要在 faces-config.xml 中註冊驗證器的識別（Validater ID），要加入以下的內容：

- faces-config.xml

faces-config.xml

```
<?xml version="1.0"?>
<!DOCTYPE faces-config PUBLIC
  "-//Sun Microsystems, Inc.//DTD JavaServer Faces Config 1.0//EN"
  "http://java.sun.com/dtd/web-facesconfig_1_0.dtd">

<faces-config>
....
   <validator>
       <validator-id>
           onlyfun.caterpillar.Password
       </validator-id>
       <validator-class>
           onlyfun.caterpillar.PasswordValidator
       </validator-class>
   </validator>
....
</faces-config>
```

要使用自訂的驗證器，我們可以使用<f:validator>標籤並設定 validatorId 屬性，例如：

```
....
<h:inputSecret value="#{user.password}" required="true">
   <f:validator validatorId="onlyfun.caterpillar.Password"/>
```

```
</h:inputSecret><p>
....
```

您也可以讓 Bean 自行負責驗證的工作，可以在 Bean 上提供一個驗證方法，這個方法沒有傳回值，並可以接收 FacesContext、UIComponent、Object 三個參數，例如：

- UserBean.java

UserBean.java

```
package onlyfun.caterpillar;

import javax.faces.application.FacesMessage;
import javax.faces.component.UIComponent;
import javax.faces.context.FacesContext;
import javax.faces.validator.ValidatorException;

public class UserBean {
   ....

   public void validate(FacesContext context,
                        UIComponent component,
                        Object obj)
           throws ValidatorException {
       String password = (String) obj;

       if(password.length() < 6) {
           FacesMessage message = new FacesMessage(
               FacesMessage.SEVERITY_ERROR,
               "字元長度小於 6",
               "字元長度不得小於 6");
           throw new ValidatorException(message);
       }

       if(!password.matches(".+[0-9]+")) {
           FacesMessage message = new FacesMessage(
                 FacesMessage.SEVERITY_ERROR,
                 "密碼必須包括字元與數字",
                 "密碼必須是字元加數字所組成");
           throw new ValidatorException(message);
       }
   }
}
```

接著可以在頁面下如下使用驗證器：

```
.....
<h:inputSecret value="#{user.password}"
               required="true"
               validator="#{user.validate}"/>
....
```

錯誤訊息處理

在使用標準轉換器或驗證器時，當發生錯誤時，會有一些預設的錯誤訊息顯示，這些訊息可以使用<h:messages>或<h:message>標籤來顯示出來，而這些預設的錯誤訊息也是可以修改的，您所要作的是提供一個訊息資源檔案，例如：

messages.properties

```
javax.faces.component.UIInput.CONVERSION=Format Error.
javax.faces.component.UIInput.REQUIRED=Please input your data.
....
```

javax.faces.component.UIInput.CONVERSION 是用來設定當轉換器發現錯誤時顯示的訊息，而 javax.faces.component.UIInput.REQUIRED 是在標籤設定了 required 為 true，而使用者沒有在欄位輸入時顯示的錯誤訊息。

您要在 faces-config.xml 中告訴 JSF 您使用的訊息檔案名稱，例如：

faces-config.xml

```
<?xml version="1.0"?>
<!DOCTYPE faces-config PUBLIC
 "-//Sun Microsystems, Inc.//DTD JavaServer Faces Config 1.0//EN"
 "http://java.sun.com/dtd/web-facesconfig_1_0.dtd">

<faces-config>
   <application>
      <local-config>
         <default-locale>en</default-locale>
         <supported-locale>zh_TW</supported-locale>
      </local-config>
      <message-bundle>messages</message-bundle>
   </application>
   .....

</faces-config>
```

在這邊我們設定了訊息檔案的名稱為 messages_xx_YY.properties，其中 xx_YY 是根據您的 Locale

來決定，轉換器或驗證器的錯誤訊息如果有設定的話，就使用設定值，如果沒有設定的話，就使用預設值。

驗證器錯誤訊息，除了上面的 javax.faces.component.UIInput.REQUIRED 之外，還有以下的幾個：

訊息識別	預設訊息	用於
javax.faces.validator.NOT_IN_RANGE	Validation Error: Specified attribute is not between the expected values of {0} and {1}.	DoubleRangeValidator 與 LongRangeValidator，{0}與{1}分別代表 minimum 與 maximum 所設定的屬性
javax.faces.validator.DoubleRangeValidator.MAXIMUM、javax.faces.validator.LongRangeValidator.MAXIMUM	Validation Error: Value is greater than allowable maximum of '{0}'.	DoubleRangeValidator 或 LongRangeValidator，{0}表示 maximum 屬性
javax.faces.validator.DoubleRangeValidator.MINIMUM、javax.faces.validator.LongRangeValidator.MINIMUM	Validation Error: Value is less than allowable minimum of '{0}'.	DoubleRangeValidator 或 LongRangeValidator，{0}代表 minimum 屬性
javax.faces.validator.DoubleRangeValidator.TYPE、javax.faces.validator.LongRangeValidator.TYPE	Validation Error: Value is not of the correct type.	DoubleRangeValidator 或 LongRangeValidator
javax.faces.validator.LengthValidator.MAXIMUM	Validation Error: Value is greater	LengthValidator，{0}代表 maximum

	than allowable maximum of "{0}".	
javax.faces.validator.LengthValidator.MINIMUM	Validation Error: Value is less than allowable minimum of "{0}".	LengthValidator，{0} 代表 minimum 屬性

在您提供自訂訊息的時候，也可以提供{0}或{1}來設定顯示相對的屬性值，以提供詳細正確的錯誤提示訊息。

訊息的顯示有概述訊息與詳述訊息，如果是詳述訊息，則在識別上加上 "_detail"，例如：

```
javax.faces.component.UIInput.CONVERSION=Error.

 javax.faces.component.UIInput.CONVERSION_detail= Detail Error.

 ....
```

除了在訊息資源檔中提供訊息，您也可以在程式中使用 FacesMessage 來提供訊息，例如在 自訂驗證器 中我們就這麼用過：

```
....
       if(password.length() < 6) {
          FacesMessage message = new FacesMessage(
              FacesMessage.SEVERITY_ERROR,
              "字元長度小於 6",
              "字元長度不得小於 6");
          throw new ValidatorException(message);
       }
 ....
```

最好的方法是在訊息資源檔中提供訊息，這麼一來如果我們要修改訊息，就只要修改訊息資源檔的內容，而不用修改程式，來看一個簡單的例子，假設我們的訊息資源檔中有以下的內容：

```
onlyfun.caterpillar.message1=This is message1.

 onlyfun.caterpillar.message2=This is message2 with \{0} and \{1}.
```

則我們可以在程式中取得訊息資源檔的內容，例如：

```
package onlyfun.caterpillar;
```

```
import java.util.Locale;
import java.util.ResourceBundle;
import javax.faces.context.FacesContext;
improt javax.faces.component.UIComponent;
import javax.faces.application.Application;
import javax.faces.application.FacesMessage;

    ....
    public void xxxMethod(FacesContext context,
                          UIComponent component,
                          Object obj) {
        // 取得應用程式代表物件
        Application application = context.getApplication();
        // 取得訊息檔案主名稱
        String messageFileName =
                           application.getMessageBundle();
        // 取得當前 Locale 物件
        Locale locale = context.getViewRoot().getLocale();
        // 取得訊息綁定 ResourceBundle 物件
        ResourceBundle rsBundle =
          ResourceBundle.getBundle(messageFileName, locale);

        String message = rsBundle.getString(
                          "onlyfun.caterpillar.message1");
        FacesMessage facesMessage = new FacesMessage(
            FacesMessage.SEVERITY_FATAL, message, message);
        ....
    }
    ....
....
```

接下來您可以將 FacesMessage 物件填入 ValidatorException 或 ConverterException 後再丟出，FacesMessage 建構時所使用的三個參數是嚴重程度、概述訊息與詳述訊息，嚴重程度有 SEVERITY_FATAL、SEVERITY_ERROR、SEVERITY_WARN 與 SEVERITY_INFO 四種。

如果需要在訊息資源檔中設定{0}、{1}等參數，則可以如下：

```
....
String message = rsBundle.getString(
                    "onlyfun.caterpillar.message2");
Object[] params = {"param1", "param2"};
```

```
message = java.text.MessageFormat.format(message, params);

FacesMessage facesMessage = new FacesMessage(
          FacesMessage.SEVERITY_FATAL, message, message);
      ....
```

如此一來，在顯示訊息時，onlyfun.caterpillar.message2 的{0}與{1}的位置就會被"param1"與"param2"所取代。

自訂轉換，驗證標籤

在 自訂驗證器 中，我們的驗證器只能驗證一種 pattern（.+[0-9]+），我們希望可以在 JSF 頁面上自訂匹配的 pattern，然而由於我們使用<f: validator>這個通用的驗證器標籤，為了要能提供 pattern 屬性，我們可以使用<f:attribute>標籤來設置，例如：

```
....
 <h:inputSecret value="#{user.password}" required="true">
   <f:validator validatorId="onlyfun.caterpillar.Password"/>
   <f:attribute name="pattern" value=".+[0-9]+"/>
</h:inputSecret><p>
....
```

使用<f:attribute>標籤來設定屬性，接著我們可以如下取得所設定的屬性：

```
....
public void validate(FacesContext context,
                     UIComponent component,
                     Object obj)
          throws ValidatorException {
    ....
    String pattern = (String)
               component.getAttributes().get("pattern");
    ....
}
....
```

您也可以開發自己的一組驗證標籤，並提供相關屬性設定，這需要瞭解 JSP Tag Library 的撰寫，所以請您先參考 JSP/Servlet 中有關於 JSP Tag Library 的介紹。

要開發驗證器轉用標籤，您可以直接繼承 javax.faces.webapp.ValidatorTag，這個類別可以幫您處理大部份的細節，您所需要的，就是重新定義它的 createValidator()方法，我們以改寫 自訂驗證器 中的 PasswordValidator 為例：

- PasswordValidator.java

PasswordValidator.java

```
package onlyfun.caterpillar;

import javax.faces.application.FacesMessage;
import javax.faces.component.UIComponent;
import javax.faces.context.FacesContext;
import javax.faces.validator.Validator;
import javax.faces.validator.ValidatorException;

public class PasswordValidator implements Validator {
   private String pattern;

   public void setPattern(String pattern) {
       this.pattern = pattern;
   }

   public void validate(FacesContext context,
                        UIComponent component,
                        Object obj)
           throws ValidatorException {
       String password = (String) obj;

       if(password.length() < 6) {
           FacesMessage message = new FacesMessage(
             FacesMessage.SEVERITY_ERROR,
             "字元長度小於 6", "字元長度不得小於 6");
           throw new ValidatorException(message);
       }

       if(pattern != null && !password.matches(pattern)) {
           FacesMessage message = new FacesMessage(
                FacesMessage.SEVERITY_ERROR,
                "密碼必須包括字元與數字",
                "密碼必須是字元加數字所組成");
           throw new ValidatorException(message);
       }
   }
}
```

主要的差別是我們提供了 pattern 屬性，在 validate()方法中進行驗證時，是根據我們所設定的 pattern 屬性，接著我們繼承 javax.faces.webapp.ValidatorTag 來撰寫自己的驗證標籤：

PasswordValidatorTag.java

```
package onlyfun.caterpillar;
```

```
import javax.faces.application.Application;
import javax.faces.context.FacesContext;
import javax.faces.validator.Validator;
import javax.faces.webapp.ValidatorTag;

public class PasswordValidatorTag extends ValidatorTag {
   private String pattern;

   public void setPattern(String pattern) {
       this.pattern = pattern;
   }

   protected Validator createValidator() {
       Application application =
           FacesContext.getCurrentInstance().
                      getApplication();
       PasswordValidator validator =
           (PasswordValidator) application.createValidator(
                  "onlyfun.caterpillar.Password");
       validator.setPattern(pattern);
       return validator;
   }
}
```

application.createValidator()方法建立驗證器物件時，是根據在 faces-config.xml 中註冊驗證器的識別（Validater ID）：

- faces-config.xml

faces-config.xml

```
<?xml version="1.0"?>
<!DOCTYPE faces-config PUBLIC
 "-//Sun Microsystems, Inc.//DTD JavaServer Faces Config 1.0//EN"
 "http://java.sun.com/dtd/web-facesconfig_1_0.dtd">

<faces-config>
....
   <validator>
       <validator-id>
           onlyfun.caterpillar.Password
       </validator-id>
       <validator-class>
           onlyfun.caterpillar.PasswordValidator
       </validator-class>
```

```
    </validator>
....
</faces-config>
```

剩下來的工作，就是佈署 tld 描述檔了，我們簡單的定義一下：

- taglib.tld

taglib.tld

```
<?xml version="1.0" encoding="UTF-8" ?>

<taglib xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
                        web-jsptaglibrary_2_0.xsd"
    version="2.0">

    <description>PasswordValidator Tag</description>
    <tlib-version>1.0</tlib-version>
    <jsp-version>2.0</jsp-version>
    <short-name>co</short-name>

    <uri>http://caterpillar.onlyfun.net</uri>

    <tag>
        <description>PasswordValidator</description>
        <name>passwordValidator</name>
        <tag-class>
            onlyfun.caterpillar.PasswordValidatorTag
        </tag-class>
        <body-content>empty</body-content>
        <attribute>
            <name>pattern</name>
            <required>true</required>
            <rtexprvalue>false</rtexprvalue>
        </attribute>
    </tag>

</taglib>
```

而我們的 index.jsp 改寫如下：

- index.jsp

index.jsp

```
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
 <%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
```

```
<%@ taglib uri="/WEB-INF/taglib.tld" prefix="co" %>
<%@page contentType="text/html;charset=Big5"%>
<html>
<head>
<title>驗證器示範</title>
</head>
<body>
   <f:view>
      <h:messages layout="table" style="color:red"/>
      <h:form>
          <h3>請輸入您的名稱</h3>
          <h:outputText value="#{user.errMessage}"/><p>
         名稱: <h:inputText value="#{user.name}"
                         required="true"/><p>
         密碼: <h:inputSecret value="#{user.password}"
                           required="true">
                  <co:passwordValidator pattern=".+[0-9]+"/>
              </h:inputSecret> <p>
          <h:commandButton value="送出"
                         action="#{user.verify}"/>
      </h:form>
   </f:view>
</body>
</html>
```

主要的差別是，我們使用了自己的驗證器標籤：

```
<co:passwordValidator pattern=".+[0-9]+"/>
```

如果要自訂轉換器標籤，方法也是類似，您要作的是繼承 javax.faces.webapp.ConverterTag，並重新定義其 createConverter()方法。

動作事件

JSF 支援事件處理模型，雖然由於 HTTP 本身無狀態（stateless）的特性，使得這個模型多少有些地方仍不太相同，但 JSF 所提供的事件處理模型已足以讓一些傳統 GUI 程式的設計人員，可以用類似的模型來開發程式。

在 簡單的導航 中，我們根據動作方法（action method）的結果來決定要導向的網頁，一個按鈕繫結至一個方法，這樣的作法實際上即使 JSF 所提供的簡化的事件處理程序，在按鈕上使用 action 繫結至一個動作方法（action method），實際上 JSF 會為其自動產生一個「預設的 ActionListener」

來處理事件，並根據其傳回值來決定導向的頁面。

如果您需要使用同一個方法來應付多種事件來源，並想要取得事件來源的相關訊息，您可以讓處理事件的方法接收一個 javax.faces.event.ActionEvent 事件參數，例如：

- UserBean.java

UserBean.java

```
package onlyfun.caterpillar;

import javax.faces.event.ActionEvent;

public class UserBean {
   private String name;
   private String password;
   private String errMessage;
   private String outcome;

   public void setName(String name) {
       this.name = name;
   }

   public String getName() {
       return name;
   }

   public void setPassword(String password) {
       this.password = password;
   }

   public String getPassword() {
       return password;
   }

   public void setErrMessage(String errMessage) {
       this.errMessage = errMessage;
   }

   public String getErrMessage() {
       return errMessage;
   }

   public void verify(ActionEvent e) {
       if(!name.equals("justin") ||
          !password.equals("123456")) {
           errMessage = "名稱或密碼錯誤" + e.getSource();
```

```
            outcome = "failure";
        }
        else {
            outcome = "success";
        }
    }

    public String outcome() {
        return outcome;
    }
}
```

在上例中，我們讓 verify 方法接收一個 ActionEvent 物件，當使用者按下按鈕，會自動產生 ActionEvent 物件代表事件來源，我們故意在錯誤訊息之後如上事件來源的字串描述，這樣就可以在顯示錯誤訊息時一併顯示事件來源描述。

為了提供 ActionEvent 的存取能力，您的 index.jsp 可以改寫如下：

- index.jsp

index.jsp

```
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
 <%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@page contentType="text/html;charset=Big5"%>
<html>
<head>
<title>第一個 JSF 程式</title>
</head>
<body>
   <f:view>
       <h:form>
           <h3>請輸入您的名稱</h3>
           <h:outputText value="#{user.errMessage}"/><p>
          名稱: <h:inputText value="#{user.name}"/><p>
          密碼: <h:inputSecret value="#{user.password}"/><p>
           <h:commandButton value="送出"
                        actionListener="#{user.verify}"
                        action="#{user.outcome}"/>
       </h:form>
   </f:view>
</body>
</html>
```

主要改變的是按鈕上使用了 actionListener 屬性，這種方法可以使用一個 ActionListener，JSF 會先檢查是否有指定的 actionListener，然後再檢查是否指定了動作方法並產生預設的

ActionListener，並根據其傳回值導航頁面。

如果您要註冊多個 ActionListener，例如當使用者按下按鈕時，順便在記錄檔中增加一些記錄訊息，您可以實作 javax.faces.event.ActionListener，例如：

LogHandler.java

```
package onlyfun.caterpillar;

import javax.faces.event.ActionListener;
....

public class LogHandler implements ActionListener {
    public void processAction(ActionEvent e) {
        // 處理 Log
    }
}
```

VerifyHandler.java

```
package onlyfun.caterpillar;

import javax.faces.event.ActionListener;
....

public class VerifyHandler implements ActionListener {
    public void processAction(ActionEvent e) {
        // 處理驗證
    }
}
```

這麼一來，您就可以使用<f:actionListener>標籤向元件註冊事件，例如：

```
<h:commandButton value="送出" action="#{user.outcome}">
   <f:actionListener type="onlyfun.caterpillar.LogHandler"/>
   <f:actionListener type="onlyfun.caterpillar.VerifyHandler"/>
</h:commandButton>
```

<f:actionListener>會自動產生 type 所指定的物件，並呼叫元件的 addActionListener()方法註冊 Listener。

即時事件

所謂的即時事件（Immediate Events），是指 JSF 視圖元件在取得請求中該取得的值之後，即

立即處理指定的事件，而不再進行後續的轉換器處理、驗證器處理、更新模型值等流程。

在 JSF 的事件模型中會有所謂即時事件，導因於 Web 應用程式的先天特性不同於 GUI 程式，所以 JSF 的事件模式與 GUI 程式的事件模式仍有相當程度的不同，一個最基本的問題正因為 HTTP 無狀態的特性，使得 Web 應用程式天生就無法直接喚起伺服端的特定物件。

所有的物件喚起都是在伺服端執行的，至於該喚起什麼物件，則是依一個基本的流程：

- 回復畫面（Restore View）

依客戶端傳來的 session 資料或伺服端上的 session 資料，回復 JSF 畫面元件。

- 套用請求值（Apply Request Values）

JSF 畫面元件各自獲得請求中的值屬於自己的值，包括舊的值與新的值。

- 執行驗證（Process Validations）

轉換為物件並進行驗證。

- 更新模型值（Update Model Values）

更新 Bean 或相關的模型值。

- 喚起應用程式（Invoke Application）

執行應用程式相關邏輯。

- 繪製回應畫面（Render Response）

對先前的請求處理完之後，產生畫面以回應客戶端執行結果。

對於動作事件（Action Event）來說，元件的動作事件是在套用請求值階段就生成 ActionEvent 物件了，但相關的事件處理並不是馬上進行，ActionEvent 會先被排入佇列，然後必須再通過驗證、更新模式值階段，之後才處理佇列中的事件。

這樣的流程對於按下按鈕然後執行後端的應用程式來說不成問題，但有些事件並不需要這樣的流程，例如只影響畫面的事件。

舉個例子來說，在表單中可能有使用者名稱、密碼等欄位，並提供有一個地區選項按鈕，使用者可以在不填下按鈕的情況下，就按下地區選項按鈕，如果依照正常的流程，則會進行驗證、更新模型值、喚起應用程式等流程，但顯然的，使用者名稱與密碼是空白的，這會引起不必要的錯誤。

您可以設定元件的事件在套用請求值之後立即被處理，並跳過後續的階段，直接進行畫面繪製以回應請求，對於 JSF 的 input 與 command 元件，都有一個 immediate 屬性可以設定，只要將其設定為 true，則指定的事件就成為立即事件。

一個例子如下：

- index.jsp

index.jsp

```
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
 <%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@page contentType="text/html;charset=UTF8"%>

<f:view locale="#{user.locale}">
<f:loadBundle basename="messages" var="msgs"/>

<html>
<head>
<title><h:outputText value="#{msgs.titleText}"/></title>
</head>
<body>

   <h:form>
       <h3><h:outputText value="#{msgs.hintText}"/></h3>
       <h:outputText value="#{msgs.nameText}"/>:
              <h:inputText value="#{user.name}"/><p>
       <h:outputText value="#{msgs.passText}"/>:
              <h:inputSecret value="#{user.password}"/><p>
       <h:commandButton value="#{msgs.commandText}"
                     action="#{user.verify}"/>
       <h:commandButton value="#{msgs.Text}"
                   immediate="true"
                   actionListener="#{user.changeLocale}"/>
  </h:form>

</body>
</html>

</f:view>
```

這是一個可以讓使用者決定使用語系的示範，最後一個 commandButton 元件被設定了 immediate 屬性，當按下這個按鈕後，JSF 套用請求值之後會立即處理指定的 actionListener，而不再進行驗證、更新模型值，簡單的說，就這個程式來說，您在輸入欄位與密碼欄位中填入的值，不會影響您的 user.name 與 user.password。

基於範例的完整起見，我們列出這個程式 Bean 物件及 faces-config.xml：

- UserBean.java

UserBean.java

```
package onlyfun.caterpillar;

import javax.faces.event.ActionEvent;

public class UserBean {
   private String locale = "en";
   private String name;
   private String password;
   private String errMessage;

   public void changeLocale(ActionEvent e) {
       if(locale.equals("en"))
           locale = "zh_TW";
       else
           locale = "en";
   }

   public String getLocale() {
       if (locale == null) {
           locale = "en";
       }
       return locale;
   }

   public void setName(String name) {
       this.name = name;
   }

   public String getName() {
       return name;
   }

   public void setPassword(String password) {
       this.password = password;
   }

   public String getPassword() {
       return password;
   }

   public void setErrMessage(String errMessage) {
       this.errMessage = errMessage;
   }

   public String getErrMessage() {
```

```
        return errMessage;
    }

    public String verify() {
        if(!name.equals("justin") ||
           !password.equals("123456")) {
            errMessage = "名稱或密碼錯誤";
            return "failure";
        }
        else {
            return "success";
        }
    }
}
```

- faces-config.xml

faces-config.xml

```
<?xml version="1.0"?>
<!DOCTYPE faces-config PUBLIC
  "-//Sun Microsystems, Inc.//DTD JavaServer Faces Config 1.0//EN"
  "http://java.sun.com/dtd/web-facesconfig_1_0.dtd">

<faces-config>
    <navigation-rule>
        <from-view-id>/pages/index.jsp</from-view-id>
        <navigation-case>
            <from-outcome>success</from-outcome>
            <to-view-id>/pages/welcome.jsp</to-view-id>
        </navigation-case>
        <navigation-case>
            <from-outcome>failure</from-outcome>
            <to-view-id>/pages/index.jsp</to-view-id>
        </navigation-case>
    </navigation-rule>

    <managed-bean>
        <managed-bean-name>user</managed-bean-name>
        <managed-bean-class>
            onlyfun.caterpillar.UserBean
        </managed-bean-class>
        <managed-bean-scope>session</managed-bean-scope>
    </managed-bean>
</faces-config>
```

訊息資源檔的內容則是如下：

- messages_en.properties

messages_en.properties

```
titleText=JSF Demo

hintText=Please input your name and password

nameText=name

passText=password

commandText=Submit

Text=\u4e2d\u6587
```

Text 中設定的是「中文」轉換為 Java Unicode Escape 格式的結果，另一個訊息資源檔的內容則是英文訊息的翻譯而已，其轉換為 Java Unicode Escape 格式結果如下：

- messages_zh_TW.properties

messages_zh_TW.properties

```
titleText=JSF\u793a\u7bc4

hintText=\u8acb\u8f38\u5165\u540d\u7a31\u8207\u5bc6\u78bc

nameText=\u540d\u7a31

passText=\u5bc6\u78bc

commandText=\u9001\u51fa

Text=English
```

welcome.jsp 就請自行設計了，程式的畫面如下：

值變事件

如果使用者改變了 JSF 輸入元件的值後送出表單，就會發生值變事件（Value Change Event），這會丟出一個 javax.faces.event.ValueChangeEvent 物件，如果您想要處理這個事件，有兩種方式，一是直接設定 JSF 輸入元件的 valueChangeListener 屬性，例如：

```
<h:selectOneMenu value="#{user.locale}"
                 onchange="this.form.submit();"
                 valueChangeListener="#{user.changeLocale}">

    <f:selectItem itemValue="zh_TW" itemLabel="Chinese"/>
    <f:selectItem itemValue="en" itemLabel="English"/>
</h:selectOneMenu>
```

為了模擬 GUI 中選擇了選單項目之後就立即發生反應，我們在 onchange 屬性中使用了 JavaScript，其作用是在選項項目發生改變之後，立即送出表單，而不用按下提交按鈕；而 valueChangeListener 屬性所綁定的 user.changeLocale 方法必須接受 ValueChangeEvent 物件，例如：

UserBean.java

```
package onlyfun.caterpillar;

import javax.faces.event.ValueChangeEvent;

public class UserBean {
   private String locale = "en";
   private String name;
   private String password;
   private String errMessage;

   public void changeLocale(ValueChangeEvent event) {
      if(locale.equals("en"))
         locale = "zh_TW";
      else
```

```
            locale = "en";
        }

        public void setLocale(String locale) {
            this.locale = locale;
        }

        public String getLocale() {
            if (locale == null) {
                locale = "en";
            }
            return locale;
        }

        public void setName(String name) {
            this.name = name;
        }

        public String getName() {
            return name;
        }

        public void setPassword(String password) {
            this.password = password;
        }

        public String getPassword() {
            return password;
        }

        public void setErrMessage(String errMessage) {
            this.errMessage = errMessage;
        }

        public String getErrMessage() {
            return errMessage;
        }

        public String verify() {
            if(!name.equals("justin") ||
               !password.equals("123456")) {
                errMessage = "名稱或密碼錯誤";
                return "failure";
            }
            else {
```

```
            return "success";
        }
    }
}
```

另一個方法是實作 javax.faces.event.ValueChangeListener 介面，並定義其 processValueChange()方法，例如：

SomeListener.java

```
package onlyfun.caterpillar;
 ....
public class SomeListener implements ValueChangeListener {
    public void processValueChange(ValueChangeEvent event) {
        ....
    }
    ....
}
```

然後在 JSF 頁面上使用<f:valueChangeListener>標籤，並設定其 type 屬性，例如：

```
{code:borderStyle=solid}
<h:selectOneMenu value="#{user.locale}"
                 onchange="this.form.submit();">
    <f:valueChangeListener
            type="onlyfun.caterpillar.SomeListener"/>
    <f:selectItem itemValue="zh_TW" itemLabel="Chinese"/>
    <f:selectItem itemValue="en" itemLabel="English"/>
</h:selectOneMenu>
```

下面這個頁面是對 立即事件 中的範例程式作一個修改，將語言選項改以下拉式選單的選擇方式呈現，這必須配合上面提供的 UserBean 類別來使用：

index.jsp

```
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
 <%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@page contentType="text/html;charset=UTF8"%>

<f:view locale="#{user.locale}">
<f:loadBundle basename="messages" var="msgs"/>

<html>
<head>
<title><h:outputText value="#{msgs.titleText}"/></title>
</head>
<body>
```

```
   <h:form>
      <h:selectOneMenu value="#{user.locale}"
              immediate="true"
              onchange="this.form.submit();"
              valueChangeListener="#{user.changeLocale}">

         <f:selectItem itemValue="zh_TW"
                     itemLabel="Chinese"/>
         <f:selectItem itemValue="en"
                     itemLabel="English"/>
      </h:selectOneMenu>

      <h3><h:outputText value="#{msgs.hintText}"/></h3>
      <h:outputText value="#{msgs.nameText}"/>:
            <h:inputText value="#{user.name}"/><p>
      <h:outputText value="#{msgs.passText}"/>:
            <h:inputSecret value="#{user.password}"/><p>
      <h:commandButton value="#{msgs.commandText}"
                    action="#{user.verify}"/>
   </h:form>

</body>
</html>

</f:view>
```

Phase 事件

在 即時事件 中我們提到，JSF 的請求執行到回應，完整的過程會經過六個階段：

- 回復畫面（Restore View）

依客戶端傳來的 session 資料或伺服端上的 session 資料，回復 JSF 畫面元件。

- 套用請求值（Apply Request Values）

JSF 畫面元件各自獲得請求中的值屬於自己的值，包括舊的值與新的值。

- 執行驗證（Process Validations）

轉換為物件並進行驗證。

- 更新模型值（Update Model Values）

更新 Bean 或相關的模型值。

- 喚起應用程式（Invoke Application）

執行應用程式相關邏輯。

- 繪製回應畫面（Render Response）

對先前的請求處理完之後，產生畫面以回應客戶端執行結果。

在每個階段的前後會引發 javax.faces.event.PhaseEvent，如果您想嘗試在每個階段的前後捕捉這個事件，以進行一些處理，則可以實作 javax.faces.event.PhaseListener，並向 javax.faces.lifecycle.Lifecycle 登記這個 Listener，以有適當的時候通知事件的發生。

PhaseListener 有三個必須實作的方法 getPhaseId()、beforePhase()與 afterPhase()，其中 getPhaseId()傳回一個 PhaseId 物件，代表 Listener 想要被通知的時機，可以設定的時機有：

- PhaseId.RESTORE_VIEW
- PhaseId.APPLY_REQUEST_VALUES
- PhaseId.PROCESS_VALIDATIONS
- PhaseId.UPDATE_MODEL_VALUES
- PhaseId.INVOKE_APPLICATION
- PhaseId.RENDER_RESPONSE
- PhaseId.ANY_PHASE

其中 PhaseId.ANY_PHASE 指的是任何的階段轉換時，就進行通知；您可以在 beforePhase()與 afterPhase()中撰寫階段前後撰寫分別想要處理的動作，例如下面這個簡單的類別會列出每個階段的名稱：

ShowPhaseListener.java

```
package onlyfun.caterpillar;

import javax.faces.event.PhaseEvent;
import javax.faces.event.PhaseId;
import javax.faces.event.PhaseListener;

public class ShowPhaseListener implements PhaseListener {

   public void beforePhase(PhaseEvent event) {
       String phaseName = event.getPhaseId().toString();
       System.out.println("Before " + phaseName);
   }

   public void afterPhase(PhaseEvent event) {
       String phaseName = event.getPhaseId().toString();
       System.out.println("After " + phaseName);
```

```
    }

    public PhaseId getPhaseId() {
        return PhaseId.ANY_PHASE;
    }
}
```

撰寫好 PhaseListener 後，我們可以在 faces-config.xml 中向 Lifecycle 進行註冊：

faces-config.xml

```
<?xml version="1.0"?>
<!DOCTYPE faces-config PUBLIC
  "-//Sun Microsystems, Inc.//DTD JavaServer Faces Config 1.0//EN"
  "http://java.sun.com/dtd/web-facesconfig_1_0.dtd">

<faces-config>
    <lifecycle>
        <phase-listener>
            onlyfun.caterpillar.ShowPhaseListener
        </phase-listener>
    </lifecycle>
    ......
</faces-config>
```

您可以使用這個簡單的類別，看看在請求任一個 JSF 畫面時所顯示的內容，藉此瞭解 JSF 每個階段的流程變化。

簡介 JSF 標準標籤

JSF 提供了標準的 HTML Renderer Kit，可以讓您搭配 JSF 元件輸出 HTML 文件，標準的 HTML Renderer Kit 主要包括了幾個類別：

- 輸出（Outputs）

其名稱以 output 作為開頭，作用為輸出指定的訊息或綁定值。

- 輸入（Inputs）

其名稱以 input 作為開頭，其作用為提供使用者輸入欄位。

- 命令（Commands）

其名稱以 command 作為開頭，其作用為提供命令或連結按鈕。

- 選擇（Selections）

其名稱以 select 作為開頭，其作用為提供使用者選項的選取。

- 其它

包括了 form、message、messages、graphicImage 等等未分類的標籤。

JSF 標準 HTML 標籤包括了幾個共通的屬性，整理如下：

屬性名稱	適用	說明
id	所有元件	可指定 id 名稱，以讓其它標籤或元件參考
binding	所有元件	綁定至 UIComponent
rendered	所有元件	是否顯示元件
styleClass	所有元件	設定 Cascading stylesheet (CSS)
value	輸入、輸出、命令元件	設定值或綁定至指定的值
valueChangeListener	輸入元件	設定值變事件處理者
converter	輸入、輸出元件	設定轉換器
validator	輸入元件	設定驗證器
required	輸入元件	是否驗證必填欄位
immediate	輸入、命令元件	是否為立即事件

除了共通的屬性之外，您還可以在某些元件上設定標籤 HTML 4.01 的屬性，像是 size、alt、width 等屬性，或者是設定 DHTML 事件屬性，例如 onchange、onclick 等等。

除了 JSF 的標準 HTML 標籤之外，您還需要一些標準核心標籤，這些標籤是獨立於 Renderer Kit 的，JSF 並不限制在 HTML 輸出表示層，核心標籤可以搭配其它的 Renderer Kit 來使用。

詳細的 HTML 標籤或核心標籤的使用與屬性說明可以查詢 Tag Library Documentation 文件

輸出類標籤

輸出類的標籤包括了 outputLabel、outputLink、outputFormat 與 outputText，分別舉例說明如下：

outputLabel

產生<label> HTML 標籤，使用 for 屬性指定元件的 client ID，例如：

```
<h:inputText id="user" value="#{user.name}"/>
 <h:outputLabel for="user" value="#{user.name}"/>
```

這會產生像是以下的標籤：

```
<input id="user" type="text" name="user" value="guest" />
 <label for="user">
```

outputLink

產生<a> HTML 標籤，例如：

```
<h:outputLink value="../index.jsp"/>
```

value 所指定的內容也可以是 JSF EL 綁定。

outputFormat

產生指定的文字訊息，可以搭配<f:param>來設定訊息的參數以格式化文字訊息，例如：

```
<f:loadBundle basename="messages" var="msgs"/>
 <h:outputFormat value="#{msgs.welcomeText}">
    <f:param value="Hello"/>
    <f:param value="Guest"/>
 </h:outputFormat>
```

如果您的 messages.properties 包括以下的內容：

```
welcomeText={0}, Your name is {1}.
```

則{0}與{1}會被取代為<f:param>設定的文字，最後顯示的文字會是：

```
Hello, Your name is Guest.
```

另一個使用的方法則是：

```
<h:outputFormat value="{0}, Your name is {1}.">
    <f:param value="Hello"/>
    <f:param value="Guest"/>
 </h:outputFormat>
```

outputText

簡單的顯示指定的值或綁定的訊息，例如：

```
<h:outputText value="#{user.name}"/>
```

輸入類標籤

輸入類標籤包括了 inputText、inputTextarea、inputSecret、 inputHidden，分別舉例說明如下：

inputText

顯示單行輸入欄位，即輸出<input> HTML 標籤，其 type 屬性設定為 text，例如：

```
<h:inputText value="#{user.name}"/>
```

inputTextarea

顯示多行輸入文字區域，即輸出<textarea> HTML 標籤，例如：

```
<h:inputTextarea value="#{user.command}"/>
```

inputSecret

顯示密碼輸入欄位，即輸出<input> HTML 標籤，其 type 屬性設定為 password，例如：

```
<h:inputSecret value="#{user.password}"/>
```

您可以設定 redisplay 屬性以決定是否要顯示密碼欄位的值，預設是 false。

inputHidden

隱藏欄位，即輸出<input> HTML 標籤，其 type 屬性設定為 hidden，隱藏欄位的值用於保留一些訊息於客戶端，以在下一次發送表單時一併送出，例如：

```
<h:inputHidden value="#{user.hiddenInfo}"/>
```

命令類標籤

命令類標籤包括 commandButton 與 commandLink，其主要作用在於提供一個命令按鈕或連結，以下舉例說明：

commandButton

顯示一個命令按鈕，即輸出<input> HTML 標籤，其 type 屬性可以設定為 button、submit 或 reset，預設是 submit，按下按鈕會觸發 javax.faces.event.ActionEvent，使用例子如下：

```
<h:commandButton value="送出" action="#{user.verify}"/>
```

您可以設定 image 屬性，指定圖片的 URL，設定了 image 屬性的話，<input>標籤的 type 屬性會被設定為 image，例如：

```
<h:commandButton value="#{msgs.commandText}"
                 image="images/logowiki.jpg"
                 action="#{user.verify}"/>
```

commandLink

產生超連結，會輸出<a> HTML 標籤，而 href 屬性會有'#'，而 onclick 屬性會含有一段 JavaScript 程式，這個 JavaScript 的目的是按下連結後自動提交表單，具體來說其作用就像按鈕，但外觀卻是超連結，包括在本體部份的內容都會成為超連結的一部份，一個使用的例子如下：

```
<h:commandLink value="#{msgs.commandText}"
               action="#{user.verify}"/>
```

產生的 HTML 輸出範例如下：

```
<a href="#" onclick="document.forms['_id3']['_id3:_idcl'].value='_id3:_id13';
document.forms['_id3'].submit(); return false;">Submit</a>
```

如果搭配<f:param>來使用，則所設定的參數會被當作請求參數一併送出，例如：

```
<h:commandLink>

  <h:outputText value="welcome"/>
  <f:param name="locale" value="zh_TW"/>
 </h:commandLink>
```

選擇類標籤 一

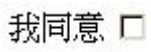
選擇類的標籤可略分為單選標籤與多選標籤，依外型的不同可以分為單選鈕（Radio）、核取方塊（CheckBox）、列示方塊（ListBox）與選單（Menu），以下分別先作簡單的說明。

<h:selectBooleanCheckbox>

在視圖上呈現一個核取方塊，例如：

```
我同意 <h:selectBooleanCheckbox value="#\{user.aggree\}"/>
```

value 所綁定的屬性必須接受與傳回 boolean 型態。這個元件在網頁上呈現的外觀如下：



<h:selectOneRadio> 、 <h:selectOneListbox> 、 <h:selectOneMenu>

這三個標籤的作用，是讓使用者從其所提供的選項中選擇一個項目，所不同的就是其外觀上的差別，例如：

```
<h:selectOneRadio value="#{user.education}">
   <f:selectItem itemLabel="高中" itemValue="高中"/>
   <f:selectItem itemLabel="大學" itemValue="大學"/>
   <f:selectItem itemLabel="研究所以上" itemValue="研究所以上"/>
 </h:selectOneRadio><p>
```

value 所綁定的屬性可以接受字串以外的型態或是自訂型態，但記得如果是必須轉換的型態或自訂型態，必須搭配 標準轉換器 或 自訂轉換器 來轉換為物件，<h:selectOneRadio>的外觀如下：

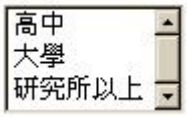
您也可以設定 layout 屬性，可設定的屬性是 lineDirection、pageDirection，預設是 lineDirection，也就是由左到右來排列選項，如果設定為 pageDirection，則是由上至下排列選項，例如設定為：

```
<h:selectOneRadio layout="pageDirection"
                value="#{user.education}">
   <f:selectItem itemLabel="高中" itemValue="高中"/>
   <f:selectItem itemLabel="大學" itemValue="大學"/>
   <f:selectItem itemLabel="研究所以上" itemValue="研究所以上"/>
 </h:selectOneRadio><p>
```

則外觀如下：

○ 高中
○ 大學
○ 研究所以上

<h:selectOneListbox>、<h:selectOneMenu>的設定方法類似於<h: selectOneRadio>，以下分別列出<h:selectOneListbox>、<h: selectOneMenu>的外觀：



<h:selectManyCheckbox> 、<h:selectManyListbox>、<h: selectManyMenu>

這三個標籤提供使用者複選項目的功能，一個<h:selectManyCheckbox>例子如下：

```
<h:selectManyCheckbox layout="pageDirection"
                      value="#{user.preferColors}">
    <f:selectItem itemLabel="紅" itemValue="false"/>
    <f:selectItem itemLabel="黃" itemValue="false"/>
    <f:selectItem itemLabel="藍" itemValue="false"/>
 </h:selectManyCheckbox><p>
```

value 所綁定的屬性必須是陣列或集合（Collection）物件，在這個例子中所使用的是 boolean 陣列，例如：

UserBean.java

```
package onlyfun.caterpillar;

public class UserBean {
   private boolean[] preferColors;

   public boolean[] getPreferColors() {
      return preferColors;
   }

   public void setPreferColors(boolean[] preferColors) {
      this.preferColors = preferColors;
   }

   ......
```

```
}
```

如果是其它型態的物件，必要時必須搭配轉換器（Converter）進行字串與物件之間的轉換。

下圖是<h:selectManyCheckbox>的外觀，這是將 layout 設定為 pageDirection 的外觀：

<h:selectManyListbox>的設定方法類似，其外觀如下：

<h:selectManyMenu>在不同的瀏覽器中會有不同的外觀，在 Mozilla Firefox 中是這樣的：

在 Internet Explorer 則是這樣的：

Comments (Hide)

選擇類標籤 二

選擇類標籤可以搭配<f:selectItem>或<f:selectItems>標籤來設定選項，例如：

```
<f:selectItem itemLabel="高中"
              itemValue="高中"
              itemDescription="學歷"
              itemDisabled="true"/>
```

itemLabel 屬性設定顯示在網頁上的文字，itemValue 設定發送至伺服端時的值，itemDescription 設定文字描述，它只作用於一些工具程式，對 HTML 沒有什麼影響，itemDisabled 設定是否選項是否作用，這些屬性也都可以使用 JSF Expression Language 來綁定至一個值。

<f:selectItem>也可以使用 value 來綁定一個傳回 javax.faces.model.SelectItem 的方法，例如：

```
<f:selectItem value="#{user.sex}"/>
```

則綁定的 Bean 上必須提供下面這個方法：

```
....
    public SelectItem getSex()  {
       return new SelectItem("男");
    }
 ....
```

如果要一次提供多個選項，則可以使用<f:selectItems>，它的 value 綁定至一個提供傳回 SelectItem?的陣列、集合，或者是 Map 物件的方法，例如：

```
<h:selectOneRadio value="#{user.education}">
    <f:selectItems value="#{user.educationItems}"/>
 </h:selectOneRadio><p>
```

這個例子中<f:selectItems>的 value 綁定至 user.educationItems，其內容如下：

```
....
    private SelectItem[] educationItems;

   public SelectItem[] getEducationItems() {
      if(educationItems == null) {
         educationItems = new SelectItem[3];
         educationItems[0] =
              new SelectItem("高中", "高中");
         educationItems[1] =
              new SelectItem("大學", "大學");
         educationItems[2] =
              new SelectItem("研究所以上", "研究所以上");
      }

      return educationItems;
   }
 ....
```

在這個例子中，SelectItem 的第一個建構參數用以設定 value，而第二個參數用以設定 label，SelectItem 還提供有數個建構函式，記得可以參考一下線上 API 文件。

您也可以提供一個傳回 Map 物件的方法，Map 的 key-value 會分別作為選項的 label-value，例如：

```
<h:selectManyCheckbox layout="pageDirection"
                    value="#{user.preferColors}">
   <f:selectItems value="#{user.preferColorItems}"/>
```

```
</h:selectManyCheckbox><p>
```

您要提供下面的程式來搭配上面這個例子：

```
....
   private Map preferColorItems;

   public Map getPreferColorItems() {
      if(preferColorItems == null) {
         preferColorItems = new HashMap();
         preferColorItems.put("紅", "Red");
         preferColorItems.put("黃", "Yellow");
         preferColorItems.put("藍", "Blue");
      }

      return preferColorItems;
   }
....
```

其它標籤

<h:messages>或<h:message>標籤的介紹，在 錯誤訊息處理 中已經有介紹了。

<h:graphicImage>

這個標籤會繪製一個 HTML <img>標籤，value 可以指定路徑或圖片 URL，路徑可以指定相對路徑或絕對路徑，例如：

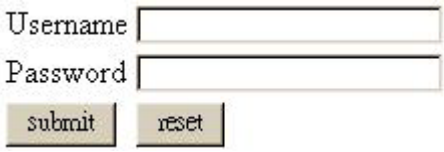
```
<h:graphicImage value="/images/logowiki.jpg"/>
```

<h:panelGrid>

這個標籤可以用來作簡單的元件排版，它會使用 HTML 表格標籤來繪製表格，並將元件置於其中，主要指定 columns 屬性，例如設定為 2：

```
<h:panelGrid columns="2">
   <h:outputText value="Username"/>
   <h:inputText id="name" value="#{userBean.name}"/>
   <h:outputText value="Password"/>
   <h:inputText id="password" value="#{userBean.password}"/>
   <h:commandButton value="submit" action="login"/>
   <h:commandButton value="reset" type="reset"/>
</h:panelGrid>
```

則自動將元件分作 2 個 column 來排列，排列出來的樣子如下：



<h:panelGrid>的本體間只能包括 JSF 元件，如果想要放入非 JSF 元件，例如簡單的樣版（template）文字，則要使用 <f:verbatim>包括住，例如：

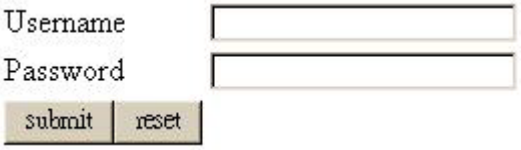
```
<h:panelGrid columns="2">
   <f:verbatim>Username</f:verbatim>
   <h:inputText id="name" value="#{userBean.name}"/>
   <f:verbatim>Password</f:verbatim>
   <h:inputText id="password" value="#{userBean.password}"/>
   <h:commandButton value="submit" action="login"/>
   <h:commandButton value="reset" type="reset"/>
</h:panelGrid>
```

<h:panelGroup>

這個元件用來將數個 JSF 元件包裝起來，使其看來像是一個元件，例如：

```
<h:panelGrid columns="2">
   <h:outputText value="Username"/>
   <h:inputText id="name" value="#{userBean.name}"/>
   <h:outputText value="Password"/>
   <h:inputText id="password" value="#{userBean.password}"/>
   <h:panelGroup>
      <h:commandButton value="submit" action="login"/>
      <h:commandButton value="reset" type="reset"/>
   </h:panelGroup>
</h:panelGrid>
```

在<h:panelGroup>中包括了兩個<h:commandButton>，這使得< h:panelGrid>在處理時，將那兩個<h:commandButton>看作是一個元件來看待，其完成的版面配置如下所示：



Comments (Hide)

簡單的表格

很多資料經常使用表格來表現，JSF 提供<h:dataTable>標籤讓您得以列舉資料並使用表格方式來呈現，舉個實際的例子來看，假設您撰寫了以下的兩個類別：

UserBean.java

```
package onlyfun.caterpillar;

public class UserBean {
   private String name;
   private String password;

   public UserBean() {
   }

   public UserBean(String name, String password) {
       this.name = name;
       this.password = password;
   }

   public String getName() {
       return name;
   }

   public void setName(String name) {
       this.name = name;
   }

   public String getPassword() {
       return password;
   }

   public void setPassword(String password) {
       this.password = password;
   }
}
```

TableBean.java

```
package onlyfun.caterpillar;

import java.util.*;

public class TableBean {
   private List userList;

   public List getUserList() {
       if(userList == null) {
          userList = new ArrayList();
```

```
            userList.add(
               new UserBean("caterpillar", "123456"));
            userList.add(
               new UserBean("momor", "654321"));
            userList.add(
               new UserBean("becky", "7890"));
         }

         return userList;
      }
   }
```

在 TableBean 中，我們假設 getUserList()方法實際上是從資料庫中查詢出 UserBean 的內容，之後傳回 List 物件，若我們的 faces-config.xml 如下：

faces-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE faces-config PUBLIC "-//Sun Microsystems,
  Inc.//DTD JavaServer Faces Config 1.0//EN"
  "http://java.sun.com/dtd/web-facesconfig_1_0.dtd">

<faces-config>
   <managed-bean>
      <managed-bean-name>tableBean</managed-bean-name>
      <managed-bean-class>
         onlyfun.caterpillar.TableBean
      </managed-bean-class>
      <managed-bean-scope>request</managed-bean-scope>
   </managed-bean>
   <managed-bean>
      <managed-bean-name>userBean</managed-bean-name>
      <managed-bean-class>
         onlyfun.caterpillar.UserBean
      </managed-bean-class>
      <managed-bean-scope>request</managed-bean-scope>
   </managed-bean>
</faces-config>
```

我們可以如下使用<h:dataTable>來產生表格資料：

index.jsp

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<html>
```

```
<body>
<f:view>
   <h:dataTable value="#{tableBean.userList}" var="user">
       <h:column>
           <h:outputText value="#{user.name}"/>
       </h:column>
       <h:column>
           <h:outputText value="#{user.password}"/>
       </h:column>
   </h:dataTable>
</f:view>
</body>
</html>
```

<h:dataTable>的 value 值綁定 tableBean 的 userList 方法，它會一個一個取出 List 中的資料並設定給 var 設定的 user，之後在每一個 column 中我們可以顯示所列舉出的 user.name 與 user.password，程式的結果會如下所示：

所產生的 HTML 表格標籤如下：

```
<table>
   <tbody>
       <tr>
           <td>caterpillar</td>
           <td>123456</td>
       </tr>
       <tr>
           <td>momor</td>
           <td>654321</td>
       </tr>
       <tr>
           <td>becky</td>
           <td>7890</td>
       </tr>
   </tbody>
</table>
```

<h:dataTable>的 value 值綁定的對象可以是以下的型態：

- 陣列
- java.util.List 的實例
- java.sql.ResultSet 的實例
- javax.servlet.jsp.jstl.sql.Result 的實例
- javax.faces.model.DataModel 的實例

表頭，表尾

<h:dataTable>配合<h:column>來以表格的方式顯示資料，< h:column>中只能包括 JSF 元件或者是<f:facet>，JSF 支援兩種 facet：header 與 footer。分別用以設定表格的表頭與表尾文字，一個設定的例子如下：

```
<h:dataTable value="#{tableBean.userList}" var="user">
    <h:column>
        <f:facet name="header">
            <h:outputText value="Name"/>
        </f:facet>
        <h:outputText value="#{user.name}"/>
        <f:facet name="footer">
            <h:outputText value="****"/>
        </f:facet>
    </h:column>
    <h:column>
        <f:facet name="header">
            <h:outputText value="Password"/>
        </f:facet>
        <h:outputText value="#{user.password}"/>
        <f:facet name="footer">
            <h:outputText value="****"/>
        </f:facet>
    </h:column>
</h:dataTable>
```

所產生的表格如下所示：

Name	Password
caterpillar	123456
momor	654321
becky	7890
*****	*****

另外，對於表頭、表尾仍至於每一行列，都可以分別設定 CSS 風格，例如下面這個 styles.css 摘錄自 Core JSF 一書：

styles.css

```
.orders {
  border: thin solid black;
}

.ordersHeader {
  text-align: center;
  font-style: italic;
  color: Snow;
  background: Teal;
}

.evenColumn {
  height: 25px;
  text-align: center;
  background: MediumTurquoise;
}

.oddColumn {
  text-align: center;
  background: PowderBlue;
}
```

可以在我們的頁面中如下加入：

```
....
```

```
<link href="styles.css" rel="stylesheet"  type="text/css"/>
....
<h:dataTable value="#{tableBean.userList}" var="user"
                    styleClass="orders"
                    headerClass="ordersHeader"
                    rowClasses="evenColumn,oddColumn">
   <h:column>
      <f:facet name="header">
         <h:outputText value="Name"/>
      </f:facet>
      <h:outputText value="#{user.name}"/>
      <f:facet name="footer">
         <h:outputText value="*****"/>
      </f:facet>
   </h:column>
   <h:column>
      <f:facet name="header">
         <h:outputText value="Password"/>
      </f:facet>
      <h:outputText value="#{user.password}"/>
      <f:facet name="footer">
         <h:outputText value="*****"/>
      </f:facet>
   </h:column>
</h:dataTable>
```

則顯示的表格結果如下：

Name	Password
caterpillar	123456
momor	654321
becky	7890
*****	*****

Comments (Hide)

TableModel 類別

在 簡單的表格 中曾經提過，<h:dataTable>可以列舉以下幾種型態的資料：

- 陣列
- java.util.List 的實例
- java.sql.ResultSet 的實例

- javax.servlet.jsp.jstl.sql.Result 的實例
- javax.faces.model.DataModel 的實例

對於前四種型態，JSF 實際上是以 javax.faces.model.DataModel 加以包裝，DataModel 是個抽象類別，其子類別都是位於 javax.faces.model 這個 package 下：

- ArrayDataModel
- ListDataModel
- ResultDataModel
- ResultSetDataModel
- ScalarDataModel

如果您想要對表格資料有更多的控制，您可以直接使用 DataModel 來設定表格資料，呼叫 DataModel 的 setWrappedObject()方法可以讓您設定對應型態的資料，呼叫 getWrappedObject() 則可以取回資料，例如：

TableBean.java

```
package onlyfun.caterpillar;

import java.util.*;
import javax.faces.model.DataModel;
import javax.faces.model.ListDataModel;

public class TableBean {
   private DataModel model;
   private int rowIndex = -1;

   public DataModel getUsers() {
       if(model == null) {
           model = new ListDataModel();
           model.setWrappedData(getUserList());
       }

       return model;
   }

   private List getUserList() {
       List userList = new ArrayList();
       userList.add(new UserBean("caterpillar", "123456"));
       userList.add(new UserBean("momor", "654321"));
       userList.add(new UserBean("becky", "7890"));

       return userList;
   }

   public int getSelectedRowIndex() {
```

```
        return rowIndex;
    }

    public String select() {
        rowIndex = model.getRowIndex();
        return "success";
    }
}
```

在這個 Bean 中，我們直接設定 DataModel?，將 userList 設定給它，如您所看到的，我們還可以取得 DataModel?的各個變項，在這個例子中，select()將作為點選表格之後的事件處理方法，我們可以藉由 DataModel?的 getRowIndex ()來取得所點選的是哪一 row 的資料，例如：

index.jsp

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
 <%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<html>
<link href="styles.css" rel="stylesheet"  type="text/css"/>
<body>
<f:view>
   <h:form>
       <h:dataTable value="#{tableBean.users}" var="user"
                    styleClass="orders"
                    headerClass="ordersHeader"
                    rowClasses="evenColumn,oddColumn">
          <h:column>
             <f:facet name="header">
                <h:outputText value="Name"/>
             </f:facet>
             <h:commandLink action="#{tableBean.select}">
                <h:outputText value="#{user.name}"/>
             </h:commandLink>

             <f:facet name="footer">
                <h:outputText value="****"/>
             </f:facet>
          </h:column>
          <h:column>
             <f:facet name="header">
                <h:outputText value="Password"/>
             </f:facet>
             <h:outputText value="#{user.password}"/>
             <f:facet name="footer">
                <h:outputText value="****"/>
             </f:facet>
          </h:column>
```

```
      </h:dataTable>
   </h:form>
   Selected Row: <h:outputText
                 value="#{tableBean.selectedRowIndex}"/>
</f:view>
</body>
</html>
```

DataModel 的 rowIndex 是從 0 開始計算，當處理 ActionEvent 時，JSF 會逐次遞增 rowIndex 的值，這讓您可以得知目前正在處理的是哪一個 row 的資料，一個執行的圖示如下：

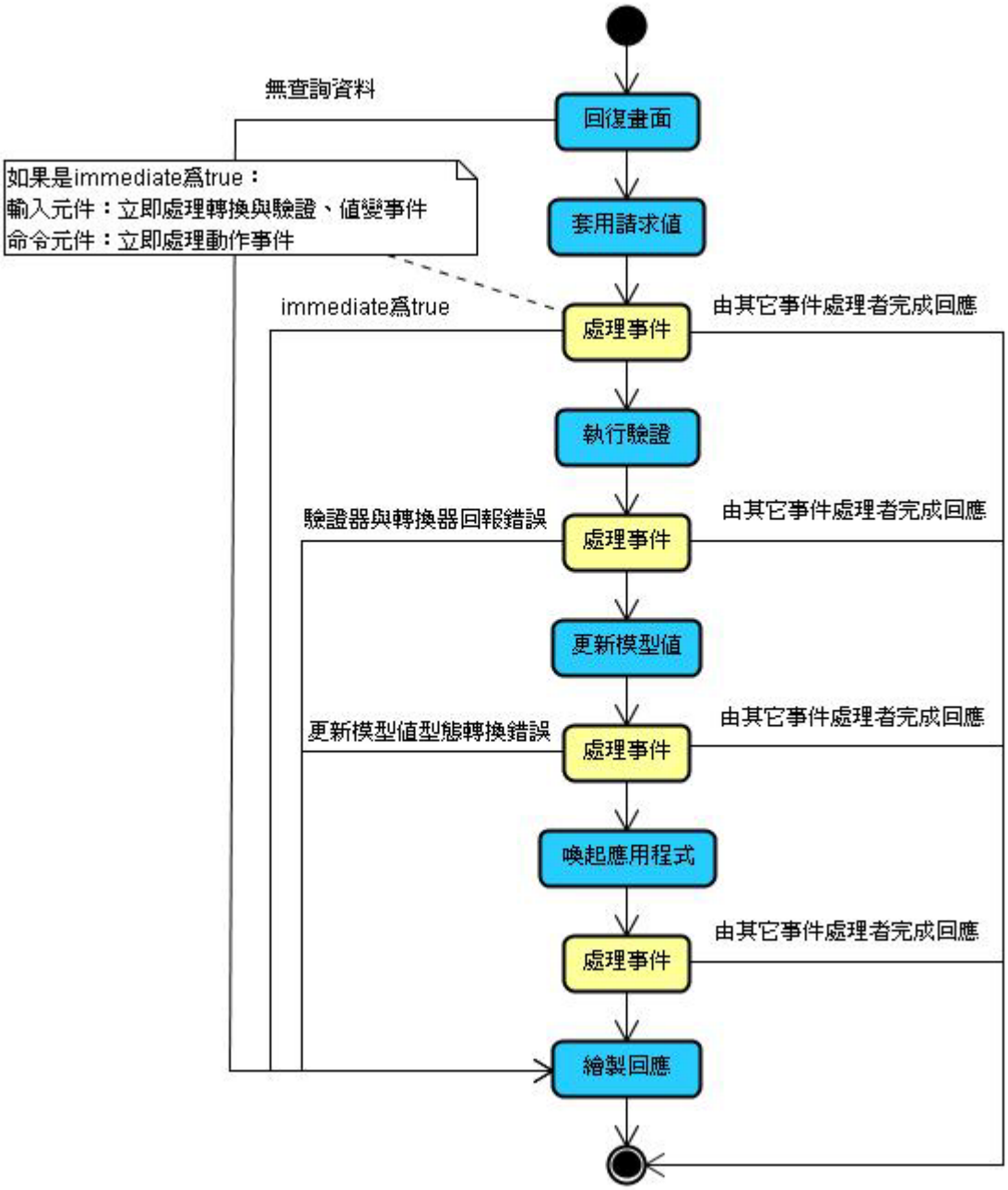
Name	Password
caterpillar	123456
momor	654321
becky	7890
*****	*****

Selected Row: 2

JSF 生命週期

JSF 的每個元件基本上都是可替換的，像是轉換器（Converter）、驗證器（Validator）、元件（Component）、繪製器（Renderer）等等，每個元件都可以替換讓 JSF 在使用時更有彈性，但相對的所付出的就是元件組合時的複雜性，為此，最基本的，如果您打算自訂一些 JSF 元件，那麼您對於 JSF 處理請求的每個階段必須要有所瞭解。

下圖是 JSF 處理請求時的每個階段與簡單說明，起始狀態即使用者端發出請求時，終止狀態則相當於繪製器發出回應時：



扣除事件處理，JSF 總共必須經過六個階段：

- 回復畫面（Restore View）

對於選擇的頁面如果是初次瀏覽則建立新的元件樹。如果是會話階段，會從使用者端或伺服器端的資料找尋資料以回復每個元件的狀態並重建元件樹，如果不包括請求參數，則直接跳過接下來的階段直接繪製回應。

- 套用申請值（Apply Request Values）

每個元件嘗試從到來的請求中找尋自己的參數並更新元件值，在這邊會觸發 ActionEvent，這個事件會被排入佇列中，然後在喚起應用程式階段之後才會真正由事件處理者進行處理。
然而對於設定 immeduate 為 true 的命令（Commamnd）元件來說，會立即處理事件並跳過之後的

階段直接繪製回應，而對於設定 immediate 為 true 的輸入（Input）元件，會馬上進行轉換驗證並處理值變事件，之後跳過接下來的階段，直接繪製回應。

- 執行驗證（Process Validations）

進行轉換與驗證處理，如果驗證錯誤，則會跳過之後的階段，直接繪製回應，結果是重新呼叫同一頁繪製結果。

- 更新模型值（Update Model Values）

更新每一個與元件綁定的 backing bean 或模型物件。

- 喚起應用程式（Invoke Application）

處理動作事件，並進行後端應用程式邏輯。

- 繪製回應（Render Response）

使用繪製器繪製頁面。

如果您只是要「使用」JSF，則您最基本的只需要知道「執行驗證」、「更新模型值」、與「喚起應用程式」這三個階段及中間的事件觸發，JSF 參考實作將這三個階段之外的其它階段之複雜性隱藏起來了，您不需要知道這幾個階段的處理細節。

然而如果您要自訂元件，則您還必須知道「回復畫面」、「套用請求值」與「繪製回應」這些階段是如何處理的，這幾個階段相當複雜，所幸的是您可以使用 JSF 所提供的框架來進行元件自訂，JSF 提供的框架已經很大程度上降低了元件製作的複雜性。

當然，即使 JSF 框架降低了複雜性，但實際上要處理 JSF 自訂元件還是很複雜的一件事，在嘗試開發自訂元件之前，您可以先搜尋一些網站，像是 Apache MyFaces http://myfaces.apache.org/，看看是不是已經有相關類似的元件已經開發完成，省去您重新自訂元件的氣力。

概述自訂元件

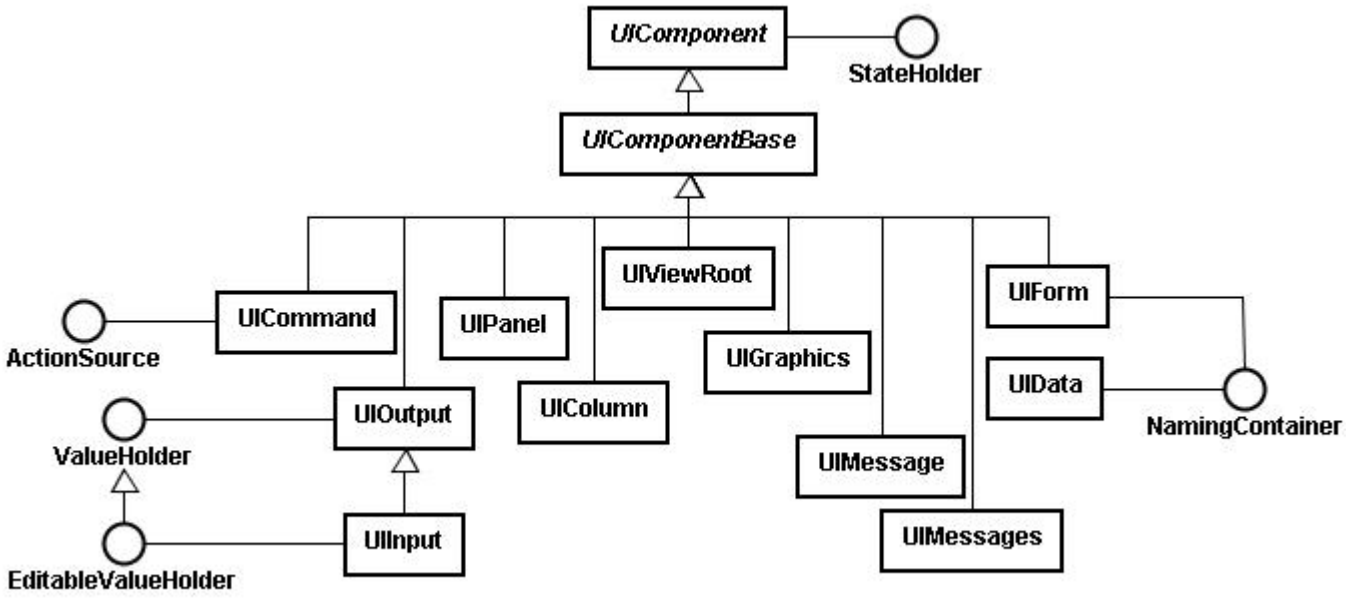
所謂的「自訂 JSF 元件」是一個概略的稱呼，事實上，一個 JSF 元件包括了三個部份：Tag、Component 與 Renderer。

Tag 即之前一直在使用的 JSF 標籤，類似於 HTML 標籤，JSF 標籤主要是方便網頁設計人員進行版面配置與資料呈現的一種方式，實際的處理中，JSF 標籤的目的在於設定 Component 屬性、設定驗證器、設定資料綁定、設定方法綁定等等。

Component 的目的在於處理請求，當請求來到伺服端應用程式時，每一個 Component 都有機會根據自己的 client id，從請求中取得屬於自己的值，接著 Component 可以將這個值作處理，然後設定給綁定的 bean。

當請求來到 Web 應用程式時，HTTP 中的字串內容可以轉換為 JSF 元件所需的值，這個動作稱之為解碼（decode），相對的，將 JSF 元件的值轉換為 HTTP 字串資料並送至客戶端，這個動作稱之為編碼（encode），Component 可自己處理編碼、解碼的任務，也可以將之委託給 Renderer 來處理。

當您要自訂 Component 時，您可以繼承 UIComonent 或其相關的子類別，這要根據您實際要自訂的元件而定，如果您要自訂一個輸出元件，可以繼承 UIOutput，如果要自訂一個輸入元件，則可以繼承 UIInput，每一個標準的 JSF 元件實際上都對應了一個 UIComponent 的子類別，下圖為一個大致的類別繼承架構圖：



實際上要自訂一個元件是複雜的一件工作，您首先要學會的是一個完整的自訂元件流程，實際上要自訂一個元件時，您可以參考一下網路上的一些成品，例如 Apache MyFaces http://myfaces.apache.org/，接下來後面的幾個主題所要介紹的，將只是一個自訂元件的簡單流程。

Renderer 是一個可替換的元件，您的 Component 可以搭配不同的 Renderer，而不用自行擔任繪製回應或解碼的動作，這會讓您的 Component 可以重用，當您需要將回應從 HTML 轉換為其它的媒介時（例如行動電話網路），則只要替換 Renderer 就可以了，這是一個好處，或者您可以簡單的替換掉一個 Renderer，就可以將原先簡單的 HTML 回應，替換為有 JavaScript 功能的 Renderer。

當您開始接觸自訂元件時，您會開始接觸到 JSF 的框架（Framework），也許有幾個類別會是您經常接觸的：

- javax.faces.component.UIComponent

自訂 Component 所要繼承的父類別，但通常，您是繼承其子類別，例如 UIInput、UIOutput 等等。

- javax.faces.webapp.UIComponentTag

自訂 JSF 標籤所要繼承的父類別，繼承它可以幫您省去許多 JSF 標籤處理的細節。

- javax.faces.context.FacesContext

包括了 JSF 相關的請求資訊，您可以透過它取得請求物件或請求參數，或者是

javax.faces.application.Application 物件。

- javax.faces.application.Application

包括了一個應用程式所共享的資訊，像是 locale、驗證器、轉換器等等，您可以透過一些 工廠方法 取得相關的資訊。

編碼，解碼

資料來源：	From PmWiki@caterpillar
作者：	caterpillar

Component 可以自己負責將物件資料編碼為 HTML 文件或其它的輸出文件，也可以將這個任務委託給 Renderer，這邊先介紹的是讓 Component 自己負責編碼的動作。

這邊著重的是介紹完成自訂元件所必須的流程，所以我們不設計太複雜的元件，這邊將完成以下的元件，這個元件會有一個輸入文字欄位以及一個送出按鈕：

您要繼承 UIComponent 或其子類別來自訂 Component，由於文字欄位是一個輸入欄位，為了方便，您可以繼承 UIInput 類別，這可以讓您省去一些處理細節的功夫，在繼承 UIComponent 或其子類別後，與編碼相關的主要有三個方法：

- encodeBegin()
- encodeChildren()
- encodeEnd()

其中 encodeChildren()是在包括子元件時必須定義，Component 如果它的 getRendersChildren()方法傳回 true 時會呼叫 encodeChildren()方法，預設上，getRendersChildren()方法傳回 false。

由於我們的自訂元件相當簡單，所以將編碼的動作寫在 encodeBegin()或是 encodeEnd()都可以，我們這邊是定義 encodeBegin ()方法：

UITextWithCmd.java

```
package onlyfun.caterpillar;

import java.io.IOException;
import java.util.Map;
import javax.faces.component.UIInput;
import javax.faces.context.FacesContext;
import javax.faces.context.ResponseWriter;
```

```
public class UITextWithCmd extends UIInput {
   private static final String TEXT = ".text";
   private static final String CMD = ".cmd";

   public UITextWithCmd() {
      setRendererType(null);
   }

   public void encodeBegin(FacesContext context)
                              throws IOException {
      ResponseWriter writer = context.getResponseWriter();
      String clientId = getClientId(context);

      encodeTextField(writer, clientId);
      encodeCommand(writer, clientId);
   }

   public void decode(FacesContext context) {
      // .....
   }

   private void encodeTextField(ResponseWriter writer,
                  String clientId) throws IOException {
      writer.startElement("input", this);
      writer.writeAttribute("name", clientId + TEXT, null);

      Object value = getValue();
      if(value != null) {
         writer.writeAttribute("value",
                        value.toString(), null);
      }

      String size = (String) getAttributes().get("size");
      if(size != null) {
         writer.writeAttribute("size", size, null);
      }

      writer.endElement("input");
   }

   private void encodeCommand(ResponseWriter writer,
                  String clientId) throws IOException {
      writer.startElement("input", this);
```

```
        writer.writeAttribute("type", "submit", null);
        writer.writeAttribute("name", clientId + CMD, null);
        writer.writeAttribute("value", "submit", null);
        writer.endElement("input");
    }
}
```

在 encodeBegin()方法中，我們取得 ResponseWriter 物件，這個物件可以協助您輸出 HTML 標籤、屬性等，我們使用 getClientId()取得元件的 id，這個 id 是每個元件的唯一識別，預設上如果您沒有指定，則 JSF 會自動為您產生 id 值。

接著我們分別對輸入文字欄位及送出鈕作 HTML 標籤輸出，在輸出時，我們將 name 屬性設成 clientId 與一個字串值的結合（即 TEXT 或 CMD），這是為了方便在解碼時，取得對應 name 屬性的請求值。

在 encodeTextField 中我們有呼叫 getValue()方法，這個方法是從 UIOutput 繼承下來的，getValue() 方法可以取得 Component 的設定值，這個值可能是靜態的屬性設定值，也可能是 JSF Expression 的綁定值，預設會先從元件的屬性設定值開始找尋，如果找不到，再從綁定值（ValueBinding 物件）中找尋，元件的屬性值或綁定值的設定，是在定義 Tag 時要作的事。

編碼的部份總結來說，是取得 Component 的值並作適當的 HTML 標籤輸出，再來我們看看解碼的部份，這是定義在 decode()方法中，將下面的內容加入至上面的類別定義中：

```
....
    public void decode(FacesContext context) {
        Map reqParaMap = context.getExternalContext().
                             getRequestParameterMap();
        String clientId = getClientId(context);

        String submittedValue =
                 (String) reqParaMap.get(clientId + TEXT);
        setSubmittedValue(submittedValue);
        setValid(true);
    }
....
```

我們必須先取得 RequestParameterMap，這個 Map 物件中填入了所有客戶端傳來的請求參數，Component 在這個方法中有機會查詢這些請求參數中，是否有自己所想要取得的資料，記得我們之前解碼時，是將輸入欄位的 name 屬性解碼為 client id 加上一個字串值（即 TEXT 設定的值），所以這時，我們嘗試從 RequestParameterMap 中取得這個請求值。

取得請求值之後，您可以將資料藉由 setSumittedValue()設定給綁定的 bean，最後呼叫 setValid()方法，這個方法設定為 true 時，表示元件正確的獲得自己的值，沒有任何的錯誤發生。

由於我們先不使用 Renderer，所以在建構函式中，我們設定 RendererType 為 null，表示我們不使用 Renderer 進行解碼輸出：

```
public UITextWithCmd() {
      setRendererType(null);
   }
```

在我們的例子中，我們都是處理字串物件，所以這邊不需要轉換器，如果您需要使用轉換器，可以呼叫 setConverter()方法加以設定，在不使用 Renderer 的時候，Component 要設定轉換器來自行進行字串與物件的轉換。

Comments (Hide)

元件標籤

完成 Component 的自訂，接下來要設定一個自訂 Tag 與之對應，自訂 Tag 的目的，在於設定 Component 屬性，取得 Componenty 型態，取得 Renderer 型態值等；屬性的設定包括了設定靜態值、設定綁定值、設定驗證器等等。

要自訂與 Component 對應的 Tag，您可以繼承 UIComponentTag，例如：

TextWithCmdTag.java

```
package onlyfun.caterpillar;

import javax.faces.application.Application;
import javax.faces.component.UIComponent;
import javax.faces.context.FacesContext;
import javax.faces.el.ValueBinding;
import javax.faces.webapp.UIComponentTag;

public class TextWithCmdTag extends UIComponentTag {
   private String size;
   private String value;

   public String getComponentType() {
       return "onlyfun.caterpillar.TextWithCmd";
   }

   public String getRendererType() {
       return null;
   }

   public void setProperties(UIComponent component) {
       super.setProperties(component);

       setStringProperty(component, "size", size);
       setStringProperty(component, "value", value);
   }
```

```
    private void setStringProperty(UIComponent component,
                    String attrName, String attrValue) {
        if(attrValue == null)
            return;

        if(isValueReference(attrValue)) {
            FacesContext context =
                        FacesContext.getCurrentInstance();
            Application application =
                        context.getApplication();
            ValueBinding binding =
                  application.createValueBinding(attrValue);
            component.setValueBinding(attrName, binding);
        }
        else {
            component.getAttributes().
                         put(attrName, attrValue);
        }
    }

    public void release() {
        super.release();
        size = null;
        value = null;
    }

    public String getSize() {
        return size;
    }

    public void setSize(String size) {
        this.size = size;
    }

    public String getValue() {
        return value;
    }

    public void setValue(String value) {
        this.value = value;
    }
}
```

首先看到這兩個方法：

```
public String getComponentType() {
       return "onlyfun.caterpillar.TextWithCmd";
   }

   public String getRendererType() {
       return null;
   }
```

由於我們的 Component 目前不使用 Renderer，所以 getRendererType()傳回 null 值，而 getComponentType()在於讓 JSF 取得這個 Tag 所對應的 Component，所傳回的值在 faces-config.xml 中要有定義，例如：

```
....
 <component>
   <component-type>
       onlyfun.caterpillar.TextWithCmd
   </component-type>
   <component-class>
       onlyfun.caterpillar.UITextWithCmd
   </component-class>
 </component>
 ....
```

藉由 faces-config.xml 中的定義，JSF 可以得知 onlyfun.caterpillar.TextWithCmd 的真正類別，而這樣的定義方式很顯然的，您可以隨時換掉<component- class>所對應的類別，也就是說，Tag 所對應的 Component 是可以隨時替換的。

在設定 Component 屬性值時，可以由 component.getAttributes()取得 Map 物件，並將標籤屬性值存入 Map 中，這個 Map 物件可以在對應的 Component 中使用 getAttributes()取得，例如在上一個主題中的 UITextWithCmd 中可以如下取得存入 Map 的 size 屬性：

UITextWithCmd.java

```
package onlyfun.caterpillar;

import java.io.IOException;
import java.util.Map;

import javax.faces.component.UIInput;
import javax.faces.context.FacesContext;
import javax.faces.context.ResponseWriter;

public class UITextWithCmd extends UIInput {
```

```
    ....
    private void encodeTextField(ResponseWriter writer,
                    String clientId) throws IOException {
        ....

        String size = (String) getAttributes().get("size");
        if(size != null) {
            writer.writeAttribute("size", size, null);
        }
        .....
    }
    ....
}
```

可以使用 isValueReference()來測試是否為 JSF Expression Language 的綁定語法，如果是的話，則我們必須建立 ValueBinding 物件，並設定值綁定：

```
....
private void setStringProperty(UIComponent component,
                    String attrName, String attrValue) {
        if(attrValue == null)
            return;

        if(isValueReference(attrValue)) {
            FacesContext context =
                        FacesContext.getCurrentInstance();
            Application application =
                        context.getApplication();
            ValueBinding binding =
                  application.createValueBinding(attrValue);
            component.setValueBinding(attrName, binding);
        }
        else {
            component.getAttributes().
                        put(attrName, attrValue);
        }
    }
....
```

如果是 value 屬性，記得在上一個主題中我們提過，從 UIOutput 繼承下來的 getValue()方法可以取得 Component 的 value 設定值，這個值可能是靜態的屬性設定值，也可能是 JSF Expression 的綁定值，預設會先從元件的屬性設定值開始找尋，如果找不到，再從綁定值（ValueBinding 物件）中找尋。

最後，我們必須提供自訂 Tag 的 tld 檔：

textcmd.tld

```
<?xml version="1.0" encoding="UTF-8"?>
 <taglib version="2.0"
    xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation=
 "http://java.sun.com/xml/ns/j2ee/web-jsptaglibrary_2_0.xsd">
<tlib-version>1.0</tlib-version>
<jsp-version>2.0</jsp-version>
<short-name>textcmd</short-name>
 <uri>http://caterpillar.onlyfun.net/textcmd</uri>


<tag>
    <name>textcmd</name>
    <tag-class>onlyfun.caterpillar.TextWithCmdTag</tag -class>
    <body-content>empty</body-content>
    <attribute>
       <name>size</name>
    </attribute>
    <attribute>
       <name>value</name>
       <required>true</required>
    </attribute>
</tag>


</taglib>
```

使用自訂元件

在 Component 與 Tag 自訂完成後，這邊來看看如何使用它們，首先定義 faces-config.xml：

faces-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
 <!DOCTYPE faces-config PUBLIC "-//Sun Microsystems,

  Inc.//DTD JavaServer Faces Config 1.0//EN"

 "http://java.sun.com/dtd/web-facesconfig_1_0.dtd">
<faces-config>
   <component>
      <component-type>
         onlyfun.caterpillar.TextWithCmd
      </component-type>
```

```
        <component-class>
            onlyfun.caterpillar.UITextWithCmd
        </component-class>
    </component>
    <managed-bean>
        <managed-bean-name>someBean</managed-bean-name>
        <managed-bean-class>
            onlyfun.caterpillar.SomeBean
        </managed-bean-class>
        <managed-bean-scope>session</managed-bean-scope>
    </managed-bean>
</faces-config>
```

<component>中定義 Component 的型態與實際的類別對應，在您於自訂 Tag 中呼叫 getComponentType()方法所返回的值，就是尋找<component-type>設定的值對應，並由此得知真正對應的 Component 類別。

我們所撰寫的 SomeBean 測試類別如下：

SomeBean

```
package onlyfun.caterpillar;

public class SomeBean {
    private String data;

    public String getData() {
        return data;
    }

    public void setData(String data) {
        this.data = data;
    }
}
```

這邊寫一個簡單的網頁來測試一下我們撰寫的自訂元件：

index.jsp

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
 <%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="/WEB-INF/textcmd.tld" prefix="oc" %>
<html>
<link href="styles.css" rel="stylesheet"  type="text/css"/>
<head>
<title></title>
</head>
<body>
```

```
<f:view>
    <h:form>
    Input data: <oc:textcmd size="10"
                        value="#{someBean.data}"/>
    </h:form>
    <h:outputText value="#{someBean.data}"/>
</f:view>
</body>
</html>
```

Comments (Hide)

自訂 Renderer

Component 可以將解碼、編碼的動作交給 Renderer，這讓您的表現層技術可以輕易的抽換，我們可以將之前的自訂元件的解碼、編碼動作移出至 Renderer，不過由於我們之前設計的 Component 是個很簡單的元件，事實上，如果只是要新增一個 Command 在輸入欄位旁邊，我們並不需要大費周章的自訂一個新的元件，我們可以直接為輸入欄位更換一個自訂的 Renderer。

要自訂一個 Renderer，您要繼承 javax.faces.render.Renderer，我們的自訂 Renderer 如下：

TextCmdRenderer.java

```
package onlyfun.caterpillar;

import java.io.IOException;
import java.util.Map;
import javax.faces.component.EditableValueHolder;
import javax.faces.component.UIComponent;
importjavax.faces.component.UIInput;
import javax.faces.context.FacesContext;
import javax.faces.context.ResponseWriter;
import javax.faces.render.Renderer;

public class TextCmdRenderer extends Renderer {
   private static final String TEXT = ".text";
   private static final String CMD = ".cmd";

   public void encodeBegin(FacesContext context,
               UIComponent component) throws IOException {
      ResponseWriter writer = context.getResponseWriter();
      String clientId = component.getClientId(context);

      encodeTextField(component, writer, clientId);
      encodeCommand(component, writer, clientId);
   }
```

```
    public void decode(FacesContext context,
                           UIComponent component) {
        Map reqParaMap = context.getExternalContext().
                               getRequestParameterMap();
        String clientId = component.getClientId(context);

        String submittedValue =
                (String) reqParaMap.get(clientId + TEXT);
        ((EditableValueHolder) component).setSubmittedValue(
                                        submittedValue);
        ((EditableValueHolder) component).setValid(true);
    }

    private void encodeTextField(UIComponent component,
                  ResponseWriter writer, String clientId)
                                throws IOException {
        writer.startElement("input", component);
        writer.writeAttribute("name", clientId + TEXT, null);

        Object value = ((UIInput) component).getValue();
        if(value != null) {
            writer.writeAttribute("value",
                       alue.toString(), null);
        }

        String size =
             (String) component.getAttributes().get("size");
        if(size != null) {
            writer.writeAttribute("size", size, null);
        }

        writer.endElement("input");
    }

    private void encodeCommand(UIComponent component,
                    ResponseWriter writer,
                    String clientId) throws IOException {
        writer.startElement("input", component);
        writer.writeAttribute("type", "submit", null);
        writer.writeAttribute("name", clientId + CMD, null);
        writer.writeAttribute("value", "submit", null);
        writer.endElement("input");
    }
}
```

這個自訂的 Renderer 其解碼、編碼過程，與之前直接在 Component 中進行解碼或編碼過程是類似的，所不同的是在解碼與編碼的方法上，多了 UIComponent 參數，代表所代理繪製的 Component。

接下來在自訂 Tag 上，我們的 TextWithCmdTag 與之前主題所介紹的沒什麼差別，只不過在 getComponentType()與 getRendererType()方法上要修改一下：

TextWithCmdTag.java

```
package onlyfun.caterpillar;

import javax.faces.application.Application;
import javax.faces.component.UIComponent;
import javax.faces.context.FacesContext;
import javax.faces.el.ValueBinding;
importjavax.faces.webapp.UIComponentTag;

public class TextWithCmdTag extends UIComponentTag {
   private String size;
   private String value;

   public String getComponentType() {
      return "javax.faces.Input";
   }

   public String getRendererType() {
      return "onlyfun.caterpillar.TextCmd";
   }
   .....
}
```

getComponentType()取得的是"javax.faces.Input"，它實際上對應至 UIInput 類別，而 getRendererType()取回的是"onlyfun.caterpillar.TextCmd"，這會在 faces-config.xml 中定義，以對應至實際的 Renderer 類別：

faces-config.xml

```
....
<faces-config>
   <render-kit>
      <renderer>
         <component-family>
            javax.faces.Input
         </component-family>
```

```
            <renderer-type>
                onlyfun.caterpillar.TextCmd
            </renderer-type>
            <renderer-class>
                onlyfun.caterpillar.TextCmdRenderer
            </renderer-class>
        </renderer>
    </render-kit>
....
</faces-config>
```

為 Component 定義一個 Renderer，必須由 component family 與 renderer type 共同定義，這並不難理解，因為一個 Component 可以搭配不同的 Renderer，但它是屬於同一個 component family，例如 UIInput 就是屬於 javax.faces.Input 這個元件家族，而我們為它定義一個新的 Renderer。

接下未完成的範例可以取之前主題介紹過的，我們雖然沒有自訂元件，但我們為 UIInput 置換了一個新的 Renderer，這個 Renderer 會在輸入欄位上加入一個按鈕。

如果您堅持使用之前自訂的 UITextWithCmd，則可以如下修改：

UITextWithCmd.java

```
package onlyfun.caterpillar;

import javax.faces.component.UIInput;

public class UITextWithCmd extends UIInput {
   public UITextWithCmd() {
       setRendererType("onlyfun.caterpillar.TextCmd");
   }
}
```

我們只是單純的繼承 UIInput，然後使用 setRendererType()設定 "onlyfun.caterpillar.TextCmd"，但並沒有為元件加入什麼行為，看來什麼事都沒有作，但事實上這是因為繼承了 UIInput，它為我們處理了大多數的細節。

接下來同樣的，設定自訂 Tag：

TextWithCmdTag.java

```
package onlyfun.caterpillar;

import javax.faces.application.Application;
```

```
import javax.faces.component.UIComponent;
import javax.faces.context.FacesContext;
import javax.faces.el.ValueBinding;
import javax.faces.webapp.UIComponentTag;

public class TextWithCmdTag extends UIComponentTag {
   private String size;
   private String value;

   public String getComponentType() {
       return "onlyfun.caterpillar.TextWithCmd";
   }

   public String getRendererType() {
       return "onlyfun.caterpillar.TextCmd";
   }
   .....
}
```

要使用自訂的 Component，記得要在 faces-config.xml 中再加入：

```
....
   <component>
      <component-type>
         onlyfun.caterpillar.TextWithCmd
      </component-type>
      <component-class>
         onlyfun.caterpillar.UITextWithCmd
      </component-class>
   </component>
...
```