利用 Spring MVC 及 RAD 7.0 開發標準基礎型 JSR 168 Portlet

Portlet MVC 架構

除了支援以 Servlet 爲基礎的 Web 開發之外，Spring 還支援 JSR-168 Portlet 開發。Portlet MVC 架構幾乎是 Web MVC 架構的鏡映影像，且利用了相同的基礎視圖抽象概念和整合技術。

附註：
請切記雖然 Spring Portlet MVC 中的 Spring MVC 概念都相同，但 JSR-168 Portlet 的特定工作流程，仍有一些顯著的差異。

Portlet 工作流程與 Servlet 工作流程中主要的差異在於對 Portlet 的要求有兩個明顯的階段：動作階段及呈現階段。動作階段只會在「後端」變更或動作發生時執行一次，如在資料庫中進行變更，然後會在呈現階段產生提供給使用者的最新顯示畫面。這裡的重點在於對單一整體要求而言，動作階段只會執行一次，而呈現階段則可能有多次的執行。如此一來，對修改系統持續狀態的活動之間，以及產生使用者顯示畫面的活動，就可提供（且需要）明確的分隔。

大部份其他的 Portlet MVC 架構都試圖在開發人員面前，將這兩個階段完全隱藏起來，使之看起來儘可能與傳統 Servlet 開發相似，但我們認爲這種方式去除了使用 Portlet 的主要優勢。因此，在整個 Spring Portlet MVC 架構中，保留分成兩個階段。此種方式的主要表現爲 Servlet 版的 MVC 類別將可以一種方法處理要求，而 Portlet 版的 MVC 類別則會有兩種方法來處理要求：一個適用於動作階段，另一個則適用於呈現階段。例如，Servlet 版本的 **AbstractController** 有 **handleRequestInternal ( )** 方法，Portlet 版的 **AbstractController** 則有 **handleActionRequestInternal ( )** 及 **handleRenderRequestInternal ( )** 方法。

DispatcherPortlet 的設計架構是利用可配置的處理器對映及視圖解決方案，將要求分派至處理器，其作業方式恰如 Web 架構中的 **DispatcherServlet**。Portlet MVC 不支援語言環境及佈景主題解決方案 - 這是 portal/portlet-container 的範圍，不適用於 Spring 的層次。然而，Spring 中依語言環境而定的所有機制（如訊息的國際化）仍能正常運作，因爲 DispatcherPortlet 與 DispatcherServlet 都是以相同的方法揭露現有的語言環境。

MVC 中的控制器

預設處理器仍是極簡單的**控制器**介面，僅提供兩種方法：
• void handleActionRequest(request, response)
• ModelAndView handleRenderRequest(request, response)
架構中還包括大部分相同的控制器實作階層，如 AbstractController、SimpleFormController 等等。至於資料連結、Command 物件用法、模型處理及視圖解決方案都與 Servlet 架構相同。

MVC 中的**視圖**

Servlet 架構的所有視圖呈現功能，都透過名爲 ViewRendererServlet 的特殊橋接 Servlet 直接使用。藉由使用這個 Servlet，將 Portlet 要求轉換爲 Servlet 要求，並且可以使用一整個 Servlet 基礎架構來呈現視圖。這表示仍可以在 Portlet 中使用所有現有的展現器 (如 JSP)。

MVC 中的**模型**

Spring Portlet MVC 可支援生命週期在現行 HTTP 要求或 HTTP 階段作業範圍內的 Bean (模型)。這並非 Spring Portlet MVC 本身提供的特定功能，而是 Spring Portlet MVC 使用之 WebApplicationContext 容器中的功能。

DispatcherPortlet：

Portlet MVC 是要求導向的 Web MVC 架構，設計出將要求分派至控制器的 Portlet，並提供有助於 Portlet 應用程式開發的其他功能。然而，Spring 的 **DispatcherPortlet** 功能則不止於此。它與 Spring **ApplicationContext** 完全整合，並讓我們可以使用 Spring 所具有的所有功能。

爲了處理要求及呈現適當的視圖，Spring DispatcherPortlet 使用了一些特殊的 Bean。這些 Bean 都包含在 Spring 架構中，並且可在 WebApplicationContext 中配置，與其他可配置的 Bean 相同。

當已設定要使用 **DispatcherPortlet**，且該特定 **DispatcherPortlet** 收到要求時，即會開始處理要求。DispatcherPortlet 處理要求的完整說明如下 ：

- PortletRequest.getLocale() 傳回的語言環境會與要求連結，讓處理中的元素在處理要求時解析使用的語言環境。
- 搜尋適當的處理器。如果找到處理器，則執行鏈會與該處理器產生關聯。
- 如果傳回模型，則會利用已使用 WebApplicationContext 配置的視圖解析器呈現視圖，若未傳回模型，則不會呈現視圖，因爲該要求已結束執行。

於執行要求期間所傳出的異常，會由 WebApplicationContext 中宣告的處理器異常解析器來處理。使用這些異常解析器，便可定義有異常傳出時的自訂動作。

我們可以自訂 Spring 的 **DispatcherPortlet**，其方法爲在 portlet.xml 檔或 Portlet 起始設定參數中加入環境定義參數。

ViewRendererServlet：

Portlet MVC 中的呈現處理比 Web MVC 複雜。爲了重複使用 Spring Web MVC 中的所有視圖技術，必須將 PortletRequest / PortletResponse 轉換成 HttpServletRequest / HttpServletResponse，然後呼叫「視圖」的呈現方法。**DispatcherPortlet** 會使用一種稱爲 **ViewRendererServletm** 的特殊 Servlet 來完成此項作業。

若要執行實際的呈現，DispatcherPortlet 將執行下列事項：

1. 連接 WebApplicationContext 與要求，將其當作 DispatcherServlet 所使用之相同 WEB_APPLICATION_CONTEXT_ATTRIBUTE 鍵值項下的屬性。

2. 將 Model 及 View 物件與要求連接，讓 ViewRendererServlet 可以使用這些物件。

3. 建立 PortletRequestDispatcher，並利用對映到 ViewRendererServlet 的 /WEB-INF/servlet/view URL 來執行 include 方法。

然後，ViewRendererServlet 便可使用適當的引數，在「視圖」上呼叫呈現方法。

使用 DispatcherPortlet 的 viewRendererUrlconfiguration 參數，可以變更 ViewRendererServlet 的實際 URL。

控制器 (Controller)：

Portlet MVC 控制器的架構基準爲 **org.springframework.web.portlet.mvc.Controller** 介面。「Portlet 控制器」介面需要兩種方法，用以處理 Portlet 要求的兩個階段：動作要求及呈現要求。動作階段應具有處理動作要求的能力，而呈現階段則需能夠處理呈現要求，並傳回適當的模型及視圖。然而「控制器」介面是非常抽象的，Spring Portlet MVC 提供了多種我們需要的功能控制器。「控制器」介面爲每一個控制器定義出最常用的功能 - 處理動作要求、處理呈現要求，以及傳回模型和視圖。

PortletWrappingController：

在「範例應用程式」中，我們將使用 **PortletWrappingController**。我們可使用 DispatcherPortlet 中現有的 Portlet 來對映要求，而無需開發新的控制器。
使用 PortletWrappingController，可將現有的 Portlet 實例化爲控制器。這是十分有用的，因爲如此便可利用攔截器對這些 Portlet 所收到的要求進行前置或後置處理。

處理器對映：

使用處理器對映，可將收到的 Portlet 要求對映至適當的處理器。DispatcherPortlet 的設計是與其他方法並用來處理要求，而非只使用 Spring Portlet MVC 自有的「控制器」。「處理器」就是任何可以處理 Portlet 要求的「物件」。「控制器」即是「處理器」的其中一例，當然也就是預設值。

基本 HandlerMapping 提供傳遞 HandlerExecutionChain 的功能，其中必須包含符合所收到要求的處理器，並且還要有適用於要求的處理器攔截器清單。當要求進入時，DispatcherPortlet 會將之送交處理器對映來檢視該要求，然後決定適當的 HandlerExecutionChain。DispatcherPortlet 接著會在鏈中執行處理器及攔截器。
配置處理器對映的概念（於執行實際處理器前或後執行，或前後皆執行）可以自行選擇是否內含攔截器，其功能非常實用。

PortletModeHandlerMapping：

這是一個簡單的處理器對映，根據 Portlet 的現行模式（如檢視、編輯、說明）來對映傳入的要求。

PortletNameInterceptor：

這是我們將運用在應用程式中的自訂處理器攔截器。Spring 的處理器對映機制具有處理器攔截器的概念，該機制在將特定功能應用在特定要求時十分有用，例如，檢查原則。Spring Portlet MVC 與 Web MVC 會以相同的方式實作這類概念。

處理器對映中的攔截器必須實作 org.springframework.web.portlet 套件中的 **HandlerInterceptor**。正如 Servlet 版本，此介面定義三種方法：

一種會在執行實際處理器之前被呼叫 (preHandle)，一種會在執行處理器之後被呼叫 (postHandle)，第三種則是在完成要求後被呼叫 (afterCompletion)。這三種方法應

能提供滿足所有前或後置處理的彈性。preHandle 方法傳回布林值，我們可利用此方法中斷或繼續執行鏈的處理。當此方法傳回 true，處理器執行鏈將繼續；傳回 false 時，DispatcherPortlet 會假設攔截器本身會處理要求（例如，呈現適當的視圖），並且將不會繼續執行執行鏈中的其他攔截器和實際處理器。
postHandle 方法只會在 RenderRequest 上被呼叫，而在 ActionRequest 及 RenderRequest 上，則會呼叫 preHandle 及 afterCompletion 方法。如果我們要在這些方法中，僅執行一種要求的邏輯，則請在執行前，檢查該要求並確定其類型。

爲服務階層配置建立新的應用程式環境定義：

我們仍需在服務階層本身的應用程式環境定義檔中，配置服務階層。此檔案名稱爲 **"applicationContext.xml"**，我們可以透過 "web.xml" 中定義的 servlet 接聽器載入此檔。您可以從任何 Servlet 環境定義中，參照在此新應用程式環境定義中所配置的所有 Bean。

在 **Web.xml 中**載入 applicationContext.xml 的 Servlet 接聽器

```
<listener>
   <listener-
class>org.springframework.web.context.ContextLoaderListener</listener-class>
 </listener>
```

我們還會將 "messageSource" Bean 項目加入 **applicationContext.xml**， 該 applicationContext.xml 會在訊息資源組 (**"messages.properties"**) 中取回

```
<bean                                                    id="messageSource"
class="org.springframework.context.support.ResourceBundleMessageSource">
      <property name="basename" value="messages"/>
   </bean>
```

現在，控制器指出視圖的完整路徑，並在控制器與視圖之間建立非必要的相依關係。我們會希望使用邏輯名稱來對映視圖，如此可以不必變更控制器即可切換視圖。如果要使用 ResourceBundleViewResolver 及 SimpleUrlHandlerMapping 類別，也可以在內容檔中設定這種對映。如需視圖與位置的基本對映，則僅需在 InternalResourceViewResolver 上簡單地設定字首及字尾即可。我們目前實作的即是第二種方法，因此，我們會修改 **"applicationContext.xml"** 並宣告 "viewResolver" 項目。藉由選擇 JstlView 的方式，使我們能在大量的訊息資源組合中使用 JSTL，並提供國際化的支援。

applicationContext.xml 提供應用程式的廣域配置。

第一個 Bean 參考 messages.properties 檔。如果我們希望能夠以不同的語言顯示，或是預設語言並非英文時，則切記要在檔案中加入語言環境。

定義於 applicationContext.xml 中的下一個 Bean 是 viewResolver，負責顯示「視圖」。字首及字尾內容定義顯示檔案的路徑及檔案的類型，此例中爲 JSP。 將內容 viewClass 設爲「Spring 架構」的 JstlView Class。此類別會啓用明確的 JSTL 支援，並且是使用 Spring 語言環境及訊息來源實作時的要件。

defaultExceptionHandlerTemplate Bean 會參照 SimpleMappingExceptionResolver，將異常指派給有錯誤訊息的 JSP。我們依照 exceptionMappings 內容的 prop 標記來查看與錯誤頁面對應的異常類型。

```
 <bean id="viewResolver"
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
     <property name="viewClass"
value="org.springframework.web.servlet.view.JstlView"></property>
     <property name="prefix" value="/WEB-INF/jsp/"></property>
     <property name="suffix" value=".jsp"></property>
  </bean>
```

請記住，applicationContext.xml 檔案中已配置「控制器」。在瀏覽器中呈現 Portlet 時，應該重新導向至 **"/WEB-INF/servlet/view"**，並由 DispatcherServlet 來處理，然後依序將要求授權給頁面控制器，把日期和時間放入模型中，並且標記可用來檢視 "HelloWorldPortlet" 的模型。

我們會使用可自訂的 PortletNameInterceptor.java 類別來設定 PortletContext。這是十分有用的，因爲如此便可利用攔截器對這些 Portlet 所收到的要求進行前置或後置處理。

PortletContext 介面會定義 Portlet container 的 Portlet 視圖，並讓 Portlet 可使用資源。使用環境定義，Portlet 可以存取 Portlet 日誌，並取得資源的 URL 參照。每個 Java 虛擬機器都有一個「Portlet 應用程式」和環境定義。

Web.xml

Log4jConfigListener 會載入 log4j.properties 並負責 Log4j 的起始設定。log4j.properties 檔案的位置是由環境定義參數 log4jConfigLocation 提供。設定 log4j.logger.org. springframework=DEBUG，以進行 Spring 的除錯。

ContextLoaderListener 的宣告非常重要，因爲這個「接聽器」將從 ContextConfigLocation 環境定義參數所傳遞的位置，載入應用程式環境定義。

ViewRendererServlet 是一種橋接 Servlet，主要用於支援 Portlet MVC，它負責視圖的呈現，以 MVC 中的 V 來代表。

所參照的 Tag 程式庫對 JSP 的呈現十分重要。

我們將定義一個 DispatcherServlet（也稱之爲 "Front Controller"），它將根據我們在後續重點上所輸入的資訊，控制所遞送的所有要求。這個 Servlet 定義還伴隨著 <servlet-mapping/> 項目，對映到將使用的 URL 型樣。我們已決定將具有 "/WEB-INF/servlet/view" 延伸的所有 URL 都遞送到 "ViewRendererServlet" Servlet (DispatcherServlet)。

```
<servlet>
   <servlet-name>ViewRendererServlet</servlet-name>
   <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-
class>
   <load-on-startup>1</load-on-startup>
 </servlet>

 <servlet-mapping>
   <servlet-name>ViewRendererServlet</servlet-name>
   <url-pattern>/WEB-INF/servlet/view</url-pattern>
 </servlet-mapping>
```

Portlet.xml

除了 Portlet 類別的對映及起始設定參數 contextConfigLocation，這只是一般 Portlet 宣告。

如在 web.xml 章節中的說明，Portlet 類別會對映 DispatcherPortlet。

此外，contextConfigLocation 起始設定參數會再次提供，它會提供 Portlet 所擁有之 Spring 配置檔 (helloworld.xml) 的位置，可能因爲我們在應用程式中加入多個 Portlet 而造成此狀況。因此，web.xml 參照的環境定義檔可提供應用程式中所有 Portlet 的廣域配置，而定義於 Portlet 宣告中的環境定義檔則提供此 Portlet 的特定配置。

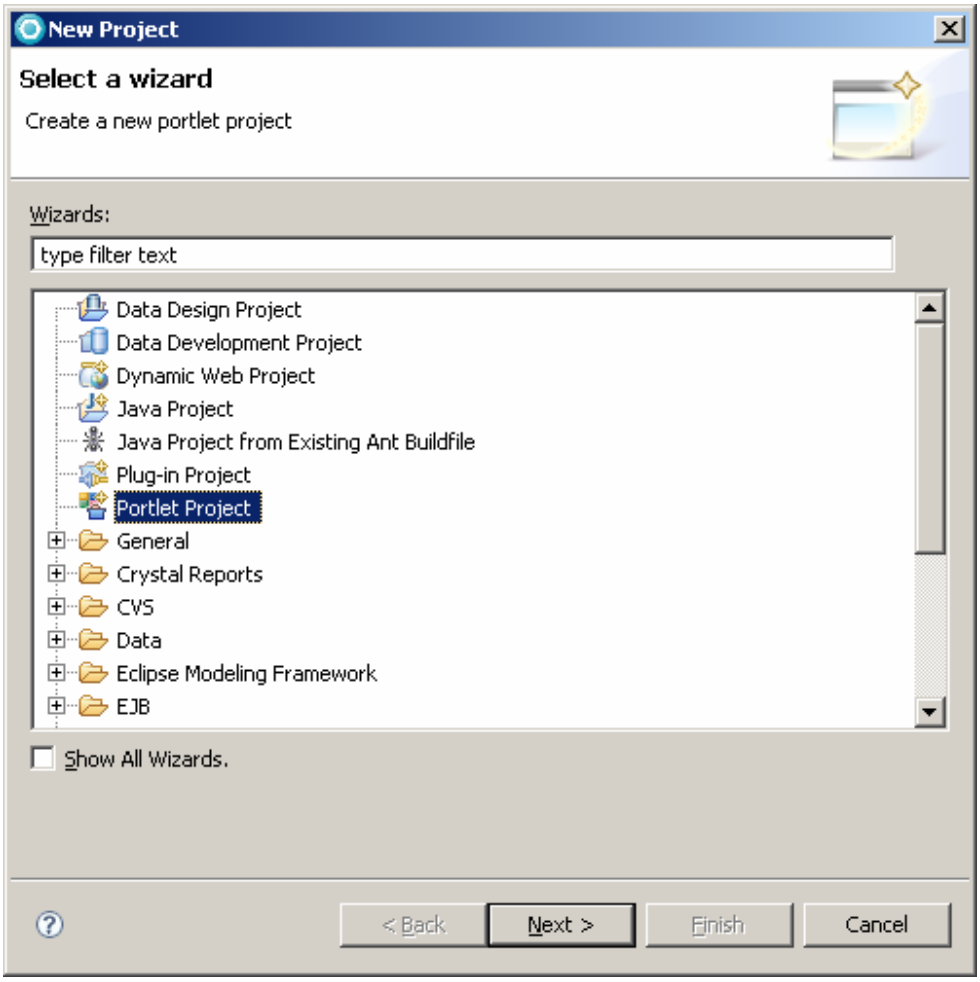
Helloworld.xml

portletModeHandlerMapping 的主要用途爲將 Portlet 模式對映到環境定義中定義的 Bean，Spring MVC 概念對映是執行 Spring Portlet 的要件，「Spring 架構」可支援 Portlet 模式的檢視、編輯、配置及說明，但它也可實作自訂的模式。此例中，Portlet 僅支援檢視模式，因爲如此即足以顯示 Spring MVC Portlet 的基本功能。

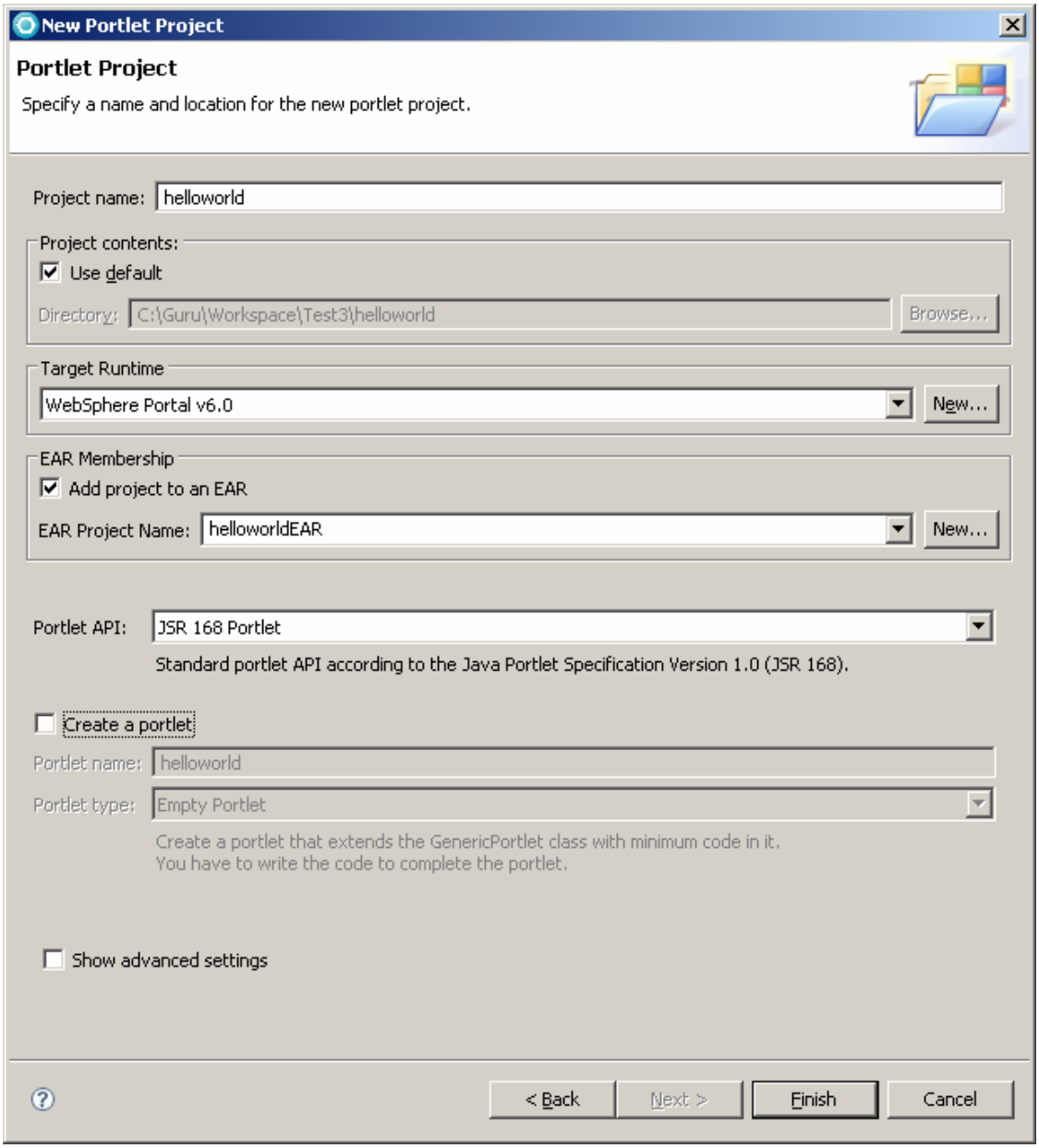
最後一個 Bean 可透過 applicationContext.xml 中的母項屬性將 defaultExceptonHandler 對映到 defaultExceptionHandlerTemplate。

建立 Spring MVC Portlet 的 RAD 7 環境：

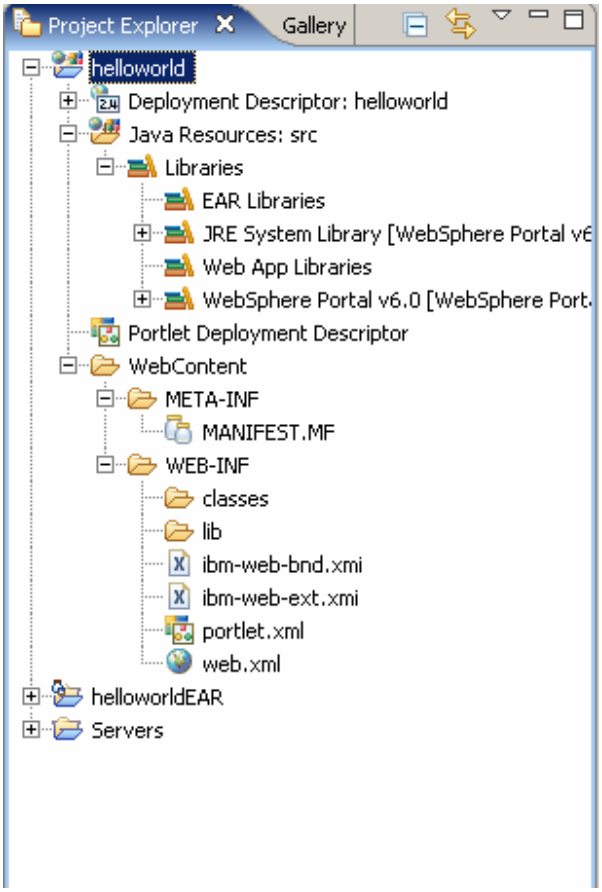
1. 開啓 RAD，並建立新的「Portlet 專案」，如下列畫面所示。



2. 建立全新的 JSR 168 Portlet 專案。

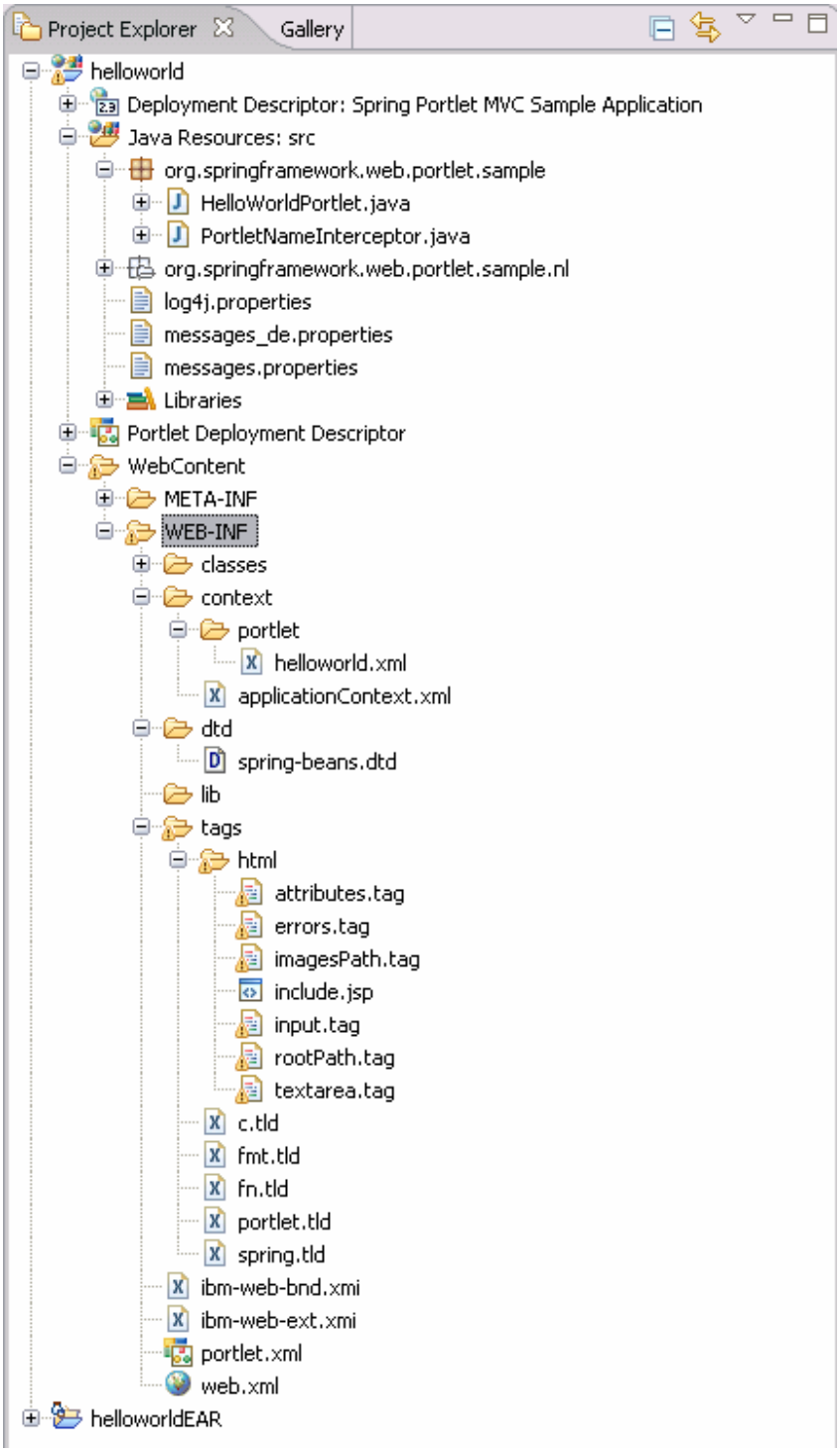

New Portlet Project
Portlet Project
Specify a name and location for the new portlet project.
Project name: helloworld
Project contents:
Use default
Directory: C:\Guru\Workspace\Test3\helloworld
Browse...
Target Runtime
WebSphere Portal v6.0
New...
EAR Membership
Add project to an EAR
EAR Project Name: helloworldEAR
New...
Portlet API: JSR 168 Portlet
Standard portlet API according to the Java Portlet Specification Version 1.0 (JSR 168).
Create a portlet
Portlet name: helloworld
Portlet type: Empty Portlet
Create a portlet that extends the GenericPortlet class with minimum code in it.
You have to write the code to complete the portlet.
Show advanced settings
< Back
Next >
Finish
Cancel

預設目錄結構：



3. 依下列畫面所示，建立「目錄結構」。

修改的目錄結構：



4. 以下列程式碼取代現有的 web.xml 檔案：

```
<?xml version="1.0"?>
```

```
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web
Application 2.3//EN"
                         "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>

      <display-name>Spring Portlet MVC Sample Application</display-
name>

      <context-param>
            <param-name>webAppRootKey</param-name>
            <param-value>org.springframework.web.portlet.sample</param-
value>
      </context-param>

    <context-param>
        <param-name>log4jConfigLocation</param-name>
        <param-value>/WEB-INF/classes/log4j.properties</param-value>
    </context-param>

      <context-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>/WEB-
INF/context/applicationContext.xml</param-value>
      </context-param>

      <listener>
            <listener-
class>org.springframework.web.util.WebAppRootListener</listener-class>
      </listener>

    <listener>
        <listener-
class>org.springframework.web.util.Log4jConfigListener</listener-class>
    </listener>

      <listener>
            <listener-
class>org.springframework.web.context.ContextLoaderListener</listener-
class>
      </listener>

      <servlet>
            <servlet-name>ViewRendererServlet</servlet-name>
            <servlet-
class>org.springframework.web.servlet.ViewRendererServlet</servlet-
class>
            <load-on-startup>1</load-on-startup>
      </servlet>

      <servlet-mapping>
            <servlet-name>ViewRendererServlet</servlet-name>
            <url-pattern>/WEB-INF/servlet/view</url-pattern>
      </servlet-mapping>

    <taglib>
```

```
        <taglib-uri>http://java.sun.com/jsp/jstl/core</taglib-uri>
        <taglib-location>/WEB-INF/tags/c.tld</taglib-location>
    </taglib>

    <taglib>
        <taglib-uri>http://java.sun.com/jsp/jstl/fmt</taglib-uri>
        <taglib-location>/WEB-INF/tags/fmt.tld</taglib-location>
    </taglib>

    <taglib>
        <taglib-uri>http://java.sun.com/jsp/jstl/functions</taglib-uri>
        <taglib-location>/WEB-INF/tags/fn.tld</taglib-location>
    </taglib>

    <taglib>
        <taglib-uri>http://www.springframework.org/tags</taglib-uri>
        <taglib-location>/WEB-INF/tags/spring.tld</taglib-location>
    </taglib>

</web-app>
```

5. 以下列程式碼取代現有的 portlet.xml 檔案：

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

<portlet-app xmlns="http://java.sun.com/xml/ns/portlet/portlet-
app_1_0.xsd"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/portlet-
app_1_0.xsd
                          http://java.sun.com/xml/ns/portlet/portlet-
app_1_0.xsd"
      version="1.0">

    <portlet>
        <portlet-name>helloworld</portlet-name>
        <portlet-
class>org.springframework.web.portlet.DispatcherPortlet</portlet-class>
            <init-param>
                  <name>contextConfigLocation</name>
                  <value>/WEB-
INF/context/portlet/helloworld.xml</value>
            </init-param>
        <supports>
            <mime-type>text/html</mime-type>
            <portlet-mode>view</portlet-mode>
        </supports>
        <portlet-info>
            <title>Hello World</title>
        </portlet-info>
    </portlet>

</portlet-app>
```

6. 使用以下的程式碼，在 WEB-INF/context 目錄中建立 **applicationContext.xml** 檔案。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN" "../dtd/spring-
beans.dtd">

<beans>

	<!-- Message source for this context, loaded from localized
"messages_xx" files -->
	<bean id="messageSource"
class="org.springframework.context.support.ResourceBundleMessageSource"
>
		<property name="basenames">
			<list>
				<value>messages</value>
			</list>
		</property>

	</bean>

	<!-- Default View Resolver -->
	<bean id="viewResolver"
class="org.springframework.web.servlet.view.InternalResourceViewResolve
r">
		<property name="cache" value="false"/>
		<property name="viewClass"
value="org.springframework.web.servlet.view.JstlView"/>
		<property name="prefix" value="/WEB-INF/jsp/"/>
		<property name="suffix" value=".jsp"/>
	</bean>

	<!-- Abstract Default Exception Handler Bean -->
    <bean id="defaultExceptionHandlerTemplate"
class="org.springframework.web.portlet.handler.SimpleMappingExceptionRe
solver" abstract="true">
		<property name="defaultErrorView" value="defError"/>
		<property name="exceptionMappings">
			<props>
				<prop
key="javax.portlet.PortletSecurityException">notAuthorized</prop>
				<prop
key="javax.portlet.UnavailableException">notAvailable</prop>
			</props>
		</property>
	</bean>

	<bean id="portletNameInterceptor"
class="org.springframework.web.portlet.sample.PortletNameInterceptor"/>

</beans>
```

7. 使用以下的程式碼，在 WEB-INF/context/portlet 目錄中建立 **helloworld.xml** 檔案。

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN" "../../dtd/spring-
beans.dtd">
<beans>

	<!-- Reused Portlet -->

	<bean id="helloWorldPortlet"
class="org.springframework.web.portlet.mvc.PortletWrappingController">
		<property name="portletClass">

	<value>org.springframework.web.portlet.sample.HelloWorldPortlet</
value>
		</property>
		<property name="useSharedPortletConfig">
			<value>false</value>
		</property>
		<property name="portletName">
			<value>Spring MVC Portlet Demo</value>
		</property>
		<property name="initParameters">
			<props>
				<prop key="Rajaguru">Name</prop>
				<prop key="IBM">Organization</prop>
			</props>
		</property>
	</bean>

	<!-- Alternate method to reuse portlet - simpler, but no ability
to rename or set init parameters
	<bean id="helloWorldPortlet"
class="org.springframework.web.portlet.sample.HelloWorldPortlet"/>
	-->

	<!-- Handler Mapping -->

	<bean id="portletModeHandlerMapping"
class="org.springframework.web.portlet.handler.PortletModeHandlerMappin
g">
		<property name="portletModeMap">
			<map>
				<entry key="view"><ref
bean="helloWorldPortlet"/></entry>
			</map>
		</property>
	</bean>

	<!-- Exceptions Handler -->
```

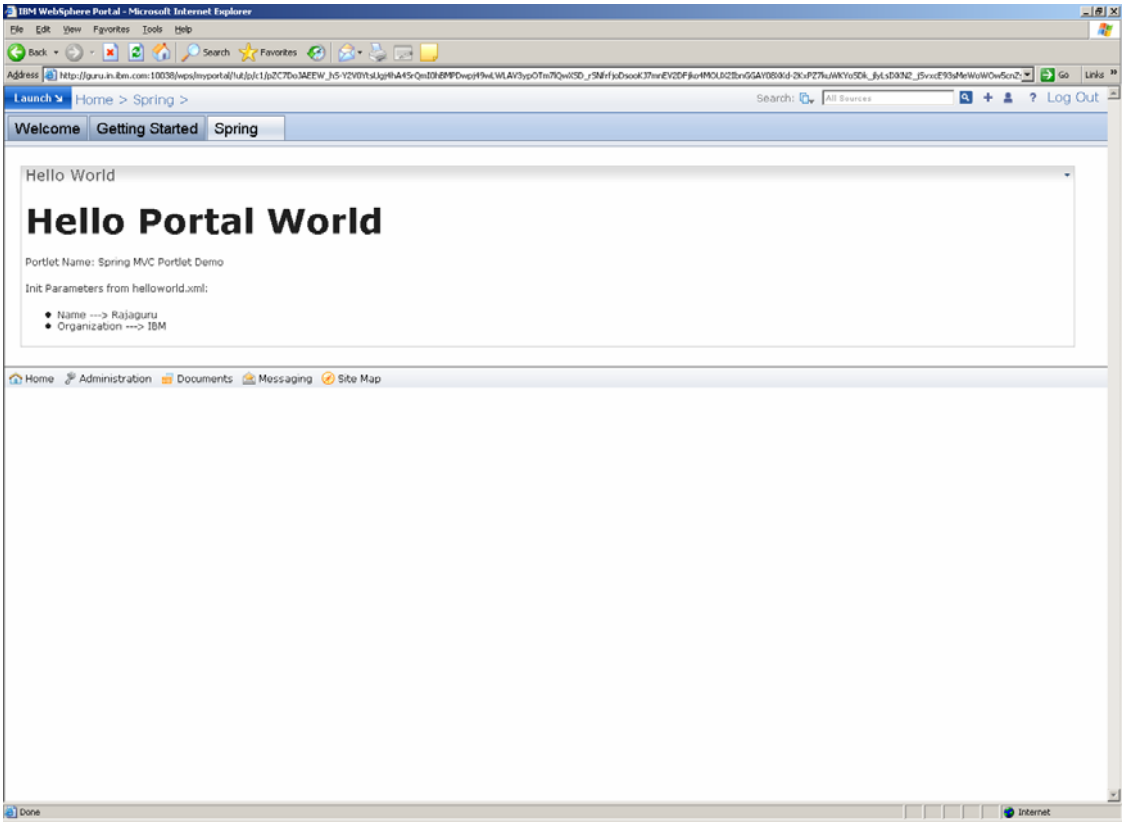
```
      <bean id="defaultExceptionHandler"
parent="defaultExceptionHandlerTemplate"/>

</beans>
```

附註：

將 SPRING、LOGGING 及 JSTL 架構所需的所有 JAR 檔案複製到 **WEB-INF/lib** 目錄。

WAR 檔（含程式碼）可使用範例應用程式。「Spring 架構」的所有 JAR 檔也可內含在 WAR 檔的 LIB 資料夾中，如此將有助於 Spring 架構的開發，原始檔也置於此。

HelloWorldPorlet 畫面：


IBM WebSphere Portal - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address
Launch
Home > Spring >
Search:
Log Out
Welcome
Getting Started
Spring
Hello World
Hello Portal World
Portlet Name: Spring MVC Portlet Demo
Init Parameters from helloworld.xml:
Name ---> Rajaguru
Organization ---> IBM
Home
Administration
Documents
Messaging
Site Map
Done
Internet