

Development of Spring 2.0 Framework

Activity Guide

SL-750 Rev A

D66979TC10

Edition 1.0

July 2010

D68082

ORACLE®

Copyright © 2007, 2010, Oracle and/or its affiliates. All rights reserved.

Disclaimer

This document contains proprietary information, is provided under a license agreement containing restrictions on use and disclosure, and is protected by copyright and other intellectual property laws. You may copy and print this document solely for your own use in an Oracle training course. The document may not be modified or altered in any way. Except as expressly permitted in your license agreement or allowed by law, you may not use, share, download, upload, copy, print, display, perform, reproduce, publish, license, post, transmit, or distribute this document in whole or in part without the express authorization of Oracle.

The information contained in this document is subject to change without notice. If you find any problems in the document, please report them in writing to: Oracle University, 500 Oracle Parkway, Redwood Shores, California 94065 USA. This document is not warranted to be error-free.

Sun Microsystems, Inc. Disclaimer

This training manual may include references to materials, offerings, or products that were previously offered by Sun Microsystems, Inc. Certain materials, offerings, services, or products may no longer be offered or provided. Oracle and its affiliates cannot be held responsible for any such references should they appear in the text provided.

Restricted Rights Notice

If this documentation is delivered to the U.S. Government or anyone using the documentation on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS

The U.S. Government's rights to use, modify, reproduce, release, perform, display, or disclose these training materials are restricted by the terms of the applicable Oracle license agreement and/or the applicable U.S. Government contract.

Trademark Notice

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. UNIX is a registered trademark licensed through X/Open Company, Ltd.



SL-750
Spring 2.0 技術開發
Student Workbook



Tian Feng (tim9958@gmail.com) has a non-transferable license to
use this Student Guide.

目錄

實驗一	1-1
Spring Framework 入門	1-1
實驗目標	1-1
準備工作	1-2
練習 1：設定 Spring Framework 的開發環境	1-3
工作項目 1：認識 Spring Framework 的安裝目錄	1-3
工作項目 2：安裝 Eclipse JST 附加元件	1-3
工作項目 3：安裝 Eclipse Spring IDE 附加元件	1-5
練習 2：寫作一個簡單的 Spring 應用程式	1-5
工作項目 1：設定 Spring 為 User Library	1-6
工作項目 2：建立 Spring 專案	1-6
工作項目 3：寫作一個簡單的 Spring 服務	1-7
實驗二	2-1
Spring 元件與容器管理.....	2-1
實驗目標	2-1
準備工作	2-2
練習 1：Bean 容器基本操作	2-3
工作項目 1：匯入練習專案	2-3
工作項目 2：使用 BeanFactory	2-3
工作項目 3：使用 ApplicationContext	2-4
工作項目 4：Setter 注入	2-5
工作項目 5：Constructor 注入	2-7
練習 2：Bean 元件的管理	2-9
工作項目 1：以 XML Schema 建立 Bean 設定檔	2-9
工作項目 2：Bean Scope	2-10
工作項目 3：p 命名空間	2-12
工作項目 4：集合元件的管理	2-12
工作項目 5：透過工廠(Factory)建立元件	2-13
練習 3：剖面導向程式設計	2-14
工作項目 1：透過 XML 開發 AOP 程式	2-14
工作項目 2：透過 Annotation 開發 AOP 程式	2-15
實驗總結	2-16

實驗三.....	3-1
Spring 與 Java 永續性.....	3-1
實驗目標	3-1
準備工作	3-2
練習 1：取得 DataSource	3-3
工作項目 1：匯入練習用資料檔	3-3
工作項目 2：匯入練習專案	3-3
工作項目 3：建立及測試 DataSource	3-3
工作項目 4：使用資料庫連結池做為資料來源.....	3-4
練習 2：透過 DAO 管理物件的永續性.....	3-6
工作項目 1：建立 ProductDao 介面.....	3-6
工作項目 2：建立 ProductRowMapper 類別.....	3-6
工作項目 3：建立 ProductDao 介面實作類別	3-7
工作項目 4：新增資料	3-9
工作項目 5：更新資料	3-10
實驗四.....	4-1
Spring 與交易管控.....	4-1
實驗目標	4-1
準備工作	4-2
練習 1：由程式控制資料庫交易	4-3
工作項目 1：修改資料表型別	4-3
工作項目 2：匯入練習專案	4-3
工作項目 3：交易控制	4-4
工作項目 4：TransactionTemplate	4-5
練習 2：透過 tx 標籤進行交易管理.....	4-7
工作項目 1：建立 TxProductService 類別	4-7
工作項目 2：透過 tx 標籤管理交易	4-8
練習 3：透過 Annotation 進行交易管理	4-10
工作項目 1：修改 TxProductService 類別	4-10
工作項目 2：修改 Bean 設定檔.....	4-10
實驗五.....	5-1
Spring 與 Hibernate 整合應用	5-1
實驗目標	5-1
準備工作	5-2
練習 1：以 XML 對映檔設定 ORM	5-3
工作項目 1：匯入練習用資料檔	5-3

工作項目 2：匯入練習專案	5-3
工作項目 3：觀察領域模型及 DAO 介面	5-3
工作項目 4：寫作 Product 類別的 ORM 設定檔	5-4
工作項目 5：實作 ProductDao	5-5
工作項目 6：寫作 Bean 設定檔	5-7
練習 2：以 Annotation 設定 ORM	5-8
工作項目 1：為 Product 類別加入 Annotation	5-8
工作項目 2：實作 ProductDao	5-9
工作項目 3：寫作 Bean 設定檔	5-9
練習 3：Hibernate 交易管理	5-10
工作項目 1：修改 TxProductService 類別	5-10
工作項目 2：修改 Bean 設定檔	5-11
實驗六	6-1
Spring 與 Web 應用程式開發	6-1
實驗目標	6-1
準備工作	6-2
練習 1：安裝及設定 Web 開發環境	6-3
工作項目 1：安裝 Tomcat 5.5	6-3
工作項目 2：設定 Web 開發環境	6-4
工作項目 3：建立 Server 實體	6-4
工作項目 4：建立及測試一個簡單的 Web 應用程式	6-5
練習 2：Spring Web MVC	6-7
工作項目 1：建立新的 Dynamic Web Project	6-7
工作項目 2：編寫網頁與程式碼	6-8
工作項目 3：編寫設定檔	6-10
練習 3：ActionSupport 與 WebApplicationContext	6-12
工作項目 1：匯入專案及 web.xml 設定	6-12
工作項目 2：編寫服務程式碼	6-13
工作項目 3：編寫設定檔	6-14
練習 4：Spring 與 Struts 的非侵入性整合	6-15
工作項目 1：匯入專案及 web.xml 設定	6-16
工作項目 2：編寫服務程式碼	6-16
工作項目 3：編寫設定檔	6-17
實驗七	7-1
Spring 與 JMS	7-1
實驗目標	7-1

準備工作	7-2
練習 1：安裝及設定 ActiveMQ	7-3
工作項目 1：設定 JAVA_HOME 環境變數	7-3
工作項目 2：啟動 Active MQ Server.....	7-3
工作項目 3：使用 jconsole 監控及操作 ActiveMQ	7-3
練習 2：發送 JMS 訊息.....	7-6
工作項目 1：匯入練習專案	7-6
工作項目 2：建立 Administered Objects.....	7-6
工作項目 3：寫作一個簡單的訊息發送器.....	7-8
工作項目 4：透過 Spring 依賴注入設定 JMS	7-9
練習 3：接收 JMS 訊息.....	7-10
工作項目 1：寫作 MessageConsumer.....	7-10
練習 4：Message-Driven POJO	7-11
工作項目 1：寫作 Message-Driven POJO 元件.....	7-11
工作項目 2：寫作 MDP Container 元件	7-12
 實驗八.....	8-1
Spring 與排程管理.....	8-1
實驗目標	8-1
準備工作	8-2
練習 1：JDK TimerTask	8-3
工作項目 1：匯入練習專案	8-3
工作項目 2：建立一個簡單的 TimerTask.....	8-3
工作項目 3：POJO-based TimerTask	8-5
練習 2：Spring 與 Quartz 的整合運用	8-6
工作項目 1：建立一個簡單的 Quartz 應用程式.....	8-6
工作項目 2：CronTrigger	8-10
工作項目 3：使用 POJO 定義 QuartzJob	8-11
 實驗九.....	9-1
Spring Framework 整合應用	9-1
實驗目標	9-1
準備工作	9-2
練習 1：安裝及佈署 JPetStore 應用程式	9-3
工作項目 1：匯入練習用資料檔	9-3
工作項目 2：匯入練習專案	9-3
工作項目 3：測試 Web 應用程式	9-3
練習 2：資料存取層的建構及設定	9-4

工作項目 1：匯入練習專案	9-4
工作項目 2：以 JPA Annotations 設定 ORM	9-5
工作項目 3：資料存取層的 Bean 設定	9-6
練習 3：商務邏輯層及展現層的設定	9-8
工作項目 1：商務邏輯層的 Bean 設定	9-8
工作項目 2：展現層的設定	9-9
工作項目 3：測試 Web 應用程式	9-10



Tian Feng (tim9958@gmail.com) has a non-transferable license to
use this Student Guide.

實驗一

Spring Framework 入門

實驗目標

當完成本實驗後，您將能學習到：

- 了解 Spring Framework 的目錄結構。
- 了解如何設定 Spring Framework 的開發環境。
- 開發一個簡單的 Spring 應用程式。

我們在這個實驗單元中帶領您循序漸進，由建構 Spring 的開發環境到實際完成一個簡單的 Spring 應用程式。本實驗的步驟是未來開發 Spring 應用程式的基礎，建議您務必熟悉這些步驟。

相關資料

準備工作



本實驗假設您已經完成以下事項：

- 下載並安裝 **J2SE 5.0**，您可以在 <http://java.sun.com/> 上找到最新的版本。(注意：本教材以 J2SE 5.0 版本為實驗環境，請勿安裝其它版本)
- 下載並安裝 **Eclipse IDE 3.2.2**，請您將 `eclipse-SDK-3.2.2-win32.zip` 解壓縮至 `C:\eclipse` 下。
- 下載 **Spring Framework 2.0** 的完整包裝檔案，檔名為 `spring-framework-2.0.x-with-dependencies.zip`，您可以在 <http://www.springframework.org/> 上找到最新版本的。Spring Framework 2.0 下載完成後，請您將解壓縮檔至 `C:\spring-framework`。

練習 1：設定 Spring Framework 的開發環境

工作項目 1：認識 Spring Framework 的安裝目錄

1. 到 c:\spring-framework 底下瀏覽一下 Spring Framework 的目錄。
2. 請您首先找到 dist 目錄，這個目錄存放的是 Spring 類別的可執行檔，在 dist 下有一個 spring.jar 檔案包含了所有 Spring Framework 的內容。
3. 在 dist/modules 下有許多單獨包裝的 Spring 類別檔，如果您在實務上希望能夠降低程式類別檔案的大小，這個目錄提供您較小的 spring 封裝。
4. 最後請您瀏覽 dist/resources，這個目錄包括所有 Spring Framework Bean 設定檔的 DTD 與 XML Schema 檔案，這些檔案中包含完整詳細的註解，如果您想要深入了解所有 Bean 的設定功能，這些註解提供了豐富的資源。
5. 接下來請您瀏覽 c:\spring-framework\lib，底下提供了 Spring 所依賴的相關類別，當您使用到特定的功能時，需要引入相關的 jar 檔，Spring 將所有的相依類別都整理在這裏，可以省去 Spring 開發人員自行個別搜尋並下傳的麻煩。
6. 接下來請您瀏覽 c:\spring-framework\doc，這裏提供了 Spring 的 javadoc 與參考手冊，參考手冊有二種版本：pdf 與 html 檔。
7. 最後請您瀏覽 c:\spring-framework\src，底下提供了 Spring Framework 所有原始碼。觀察 Spring 原始碼是了解其底層運作機制最詳細的資訊來源。

工作項目 2：安裝 Eclipse JST 附加元件

本工作項目會指引您完成 J2EE Standard Tools(JST)的安裝。

1. 啓動 c:\eclipse 的 Eclipse IDE。
2. 透過「Help」→「Software Updates」→「Find and Install...」打開 Install/Update 視窗。
3. 選擇 Search for new features to install，按「Next」。
4. 在 Sites to include in search 中勾選「Callisto Discovery Site」，按「Finish」。

練習 1：設定 Spring Framework 的開發環境

5. 在 Update Site Mirrors 視窗中選擇最下方的「Callisto Discovery Site」，按「OK」。
6. 接著會出現 Updates 視窗，在 Select the features to install 中將 Callisto Discovery Site 展開，勾選 Java Development 子項目下的 J2EE Standard Tools (JST) Project。
7. 此時會出現警告訊息，點選右方的 Select Required，系統會自行幫您處理相依性問題，按「Next」。

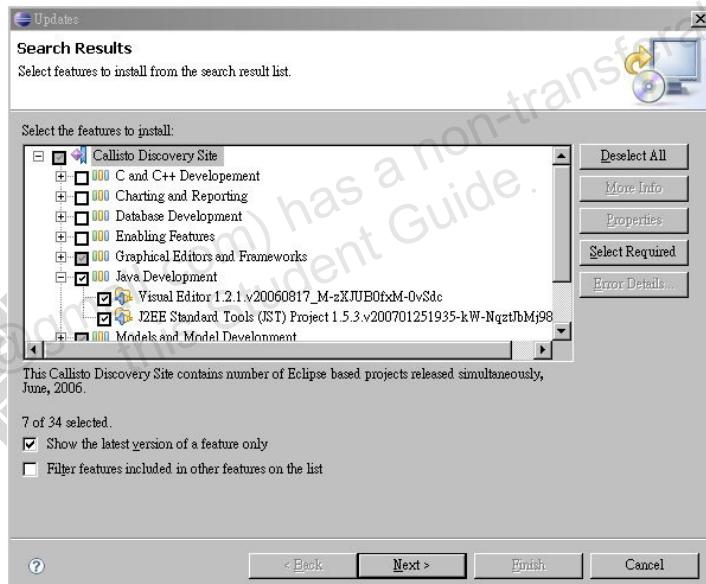


圖 1-1 選擇 JST 安裝

8. 在授權畫面中選擇「I accept the terms...」，按「Next」。
9. 接下來系統與跟您確認要安裝的元件，請點選「Finish」。
10. 點選完成之後，Eclipse 會在停滯一段時間後開始下載剛才所選擇的元件。

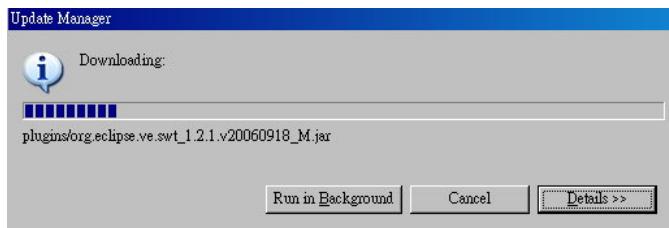


圖 1-2 下載 JST 元件

11. 下載完成後，會出現 Verification 畫面，選擇「Install All」。
12. 最後系統會建議您重新啟動 Eclipse，請依指示重新啟動後即完成安裝。



重點提示－在前幾個實驗單元中，我們會針對各個步驟做較詳細的提示，為不造成參閱時的困擾，在往後幾個單元，我們會漸漸減少瑣碎的提示，如果您在操作上有疑問，可參閱前幾個實驗單元的內容或附錄 1(Eclipse 入門)。

工作項目 3：安裝 Eclipse Spring IDE 附加元件

1. 透過「Help」→「Software Updates」→「Find and Install...」打開 Install/Update 視窗。
2. 選擇 Search for new features to install，按「Next」。
3. 在 Install 視窗中選擇右方的「New Remote Site...」。
4. 在 New Update Site 視窗中輸入下面的資訊(如圖 1-3)，輸入完成後按「OK」：
Name : Spring IDE
URL : http://springide.org/updatesite_dev/



重點提示－Spring IDE 的 2.x 版本目前還在開發階段，所以必須經由 updatesite_dev 目錄才能取得。

5. 選擇完成後按 Finish，接下來會出現 Search Results 視窗，請勾選 Spring IDE 中的最新版本。
6. 在授權畫面中選擇「I accept the terms...」，按「Next」。
7. 接下來系統與跟您確認要安裝的元件，請點選「Finish」。
8. 點選完成之後，Eclipse 會下載剛才所選擇的元件。
9. 下載完成後，會出現 Verification 畫面，選擇「Install All」。
10. 最後系統會建議您重新啟動 Eclipse，請依指示重新啟動後即完成安裝。

練習 2：寫作一個簡單的 Spring 應用程式

本練習的目的是使您瞭解一個典型 Spring 應用程式的開發步驟。在這個練習中，我們會寫作一個簡單的 Spring 計算服務，計算服務必須經由 Calculator 介面存取，透過 Spring 的 Bean 設定檔我們可以自由更換 Calculator 的實作品 (Implementation)，讓 Calculator 展現不同功能。

練習 2：寫作一個簡單的 Spring 應用程式

工作項目 1：設定 Spring 為 User Library

我們可以將常用到的函式庫設定為 User Library，如此一來在開發新專案時不需要每一次都將同樣的 Library 拷貝一次。User Library 實際上比較像是一個 Library 的位置指標，指向存放該 Library jar 檔的位置。

1. 透過「Windows」→「Preferences」打開 Preference 視窗。
2. 展開視窗左側的樹狀結構，找到「Java」→「Build Path」→「User Library」，此時右方會變成 User Libraries 視窗。
3. 點選「New...」開啟 New User Library 視窗。
4. **User library name** 填入「Spring Framework 2.0」。
5. 接下來按「Add JARs...」，透過檔案視窗加入 C:\spring-framework\dist\spring.jar 檔，按「OK」。
6. 和第 5 步驟相同的方式將「C:\spring-framework\lib\jakarta-commons\commons-logging.jar」加入 Spring Library 中，按「OK」→「Finished」。

工作項目 2：建立 Spring 專案

在這個工作項目中我們將建立一個新的 Java 專案，並加入 Spring 的功能(Add Spring Nature)。

1. 根據下面的資訊，建立一個新的 Java Project：

Project name : Lab1

Contents : Create project from existing source

Directory : C:\my_spring_project\Lab1\

JRE : Use default JRE (請檢查是否為 1.5 以上)

Project layout : Create separate source and output folders
此時 Eclipse 會在指定的目錄下新建一個專案目錄。

2. 加入 Spring IDE 支援：在 Package View 中，使用右鍵點選專案，選取「Add Spring Project Nature」。

練習 2：寫作一個簡單的 Spring 應用程式

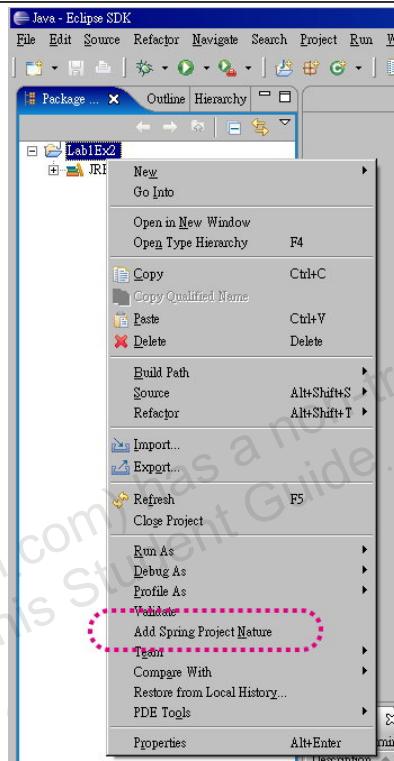


圖 1-3 Add Spring Project Nature

3. 接著要加入 Spring Library，首先在 Package View 中，使用右鍵點選專案，選取「Properties」，打開 Properties for Lab1 視窗。
4. 在左邊選取 Java Build Path，右方出現 Java Build Path 設定區塊。
5. 選取 Libraries 的 Tab，按下「Add Library...」
6. 在 Add Library 視窗中選取「User Library」，按「Next」。
7. 在 Spring Framework 2.0 上打勾，按「Finish」→「OK」。

工作項目 3：寫作一個簡單的 Spring 服務

接下來我們會先寫作一個提供計算功能介面，但是透過不同的實作品，可提供加減乘除等不同功能。

1. 建立 Calculator 介面：在 Package View 中，在 src 上按右鍵→「New」→「Interface」，根據下列資料建立一個

練習 2：寫作一個簡單的 Spring 應用程式

Calculator 介面：

Source folder : Lab1/src

Package : lab1

Name : Calculator

2. 在 Calculator 中加入 calculate 方法(請參考程式碼 1-1)：

程式碼 1-1

```

1 package lab1;
2
3 public interface Calculator
4 {
5     public int calculate(int input1, int input2);
6     public String getSymbol();
}

```

3. 建立二個 Calculator 介面的實作品，一個提供加法功能，一個提供減法功能，在 Package View 中，在 src 上按右鍵 → 「New」→ 「Class」，根據下列資料建立一個 Adder 類別：

Source folder : Lab1/src

Package : lab1

Name : Adder

4. 在 Adder 類別中實作 Calculator 介面(請參考程式碼 1-2)：

程式碼 1-2

```

1 public class Adder implements Calculator
2 {
3
4     public int calculate(int input1, int input2)
5     {
6         return input1+input2;
7     }
8     public String getSymbol()
9     {
10        return "+";
11    }
}

```

練習 2：寫作一個簡單的 Spring 應用程式

}

5. 重覆步驟 3~4，建立 Multiplier 類別(請參考程式碼 1-3)：

程式碼 1-3

```

1 public class Multiplier implements Calculator
2 {
3
4     public int calculate(int input1, int input2)
5     {
6         return input1*input2;
7     }
8     public String getSymbol()
9     {
10        return "*";
11    }
12 }
13 }
```

6. 接下來我們要建立一個 Bean 設定檔來組裝所寫好的元件，Bean 設定檔的名稱為：beans-config.xml。首先在 Package View 中，在 src 上按右鍵→「New」→「Other...」開啟 New 視窗。
7. 在 New 視窗中展開「XML」資料夾，選取「XML」項目後按「Next」。

練習 2：寫作一個簡單的 Spring 應用程式

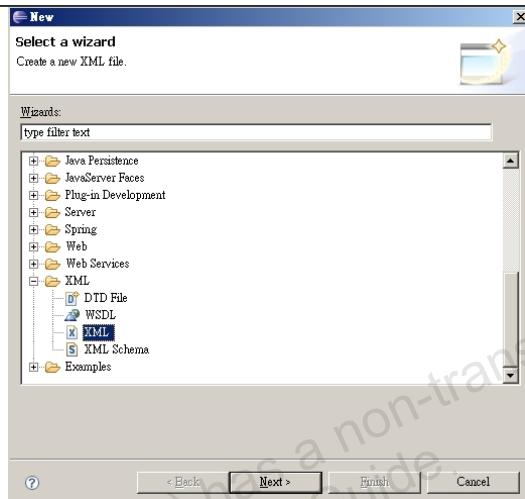


圖 1-4 建立 Bean 設定檔

8. 在 Create XML File 視窗中選取「Create XML file form a DTD file」，按「Next」。
9. 系統會詢問 Bean 設定檔的位置與檔名，首先將反白移到 Lab1 下的 src 上，在 File name 欄位輸入 beans-config.xml，按「Next」。

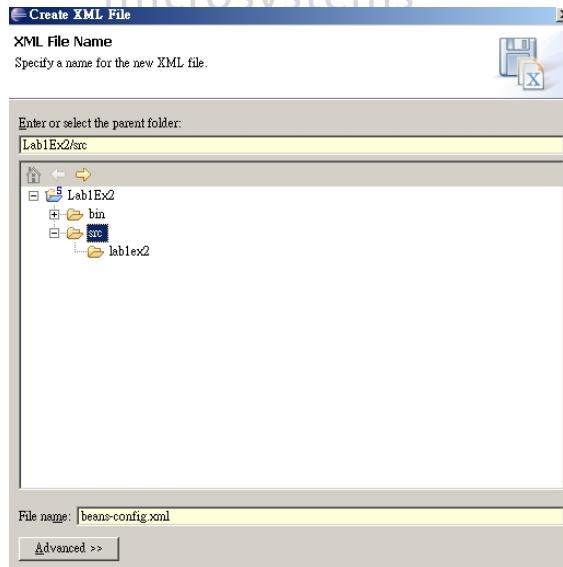


圖 1-5 設定 Bean 設定檔的檔名及存放位置

10. 系統要求您選取 DTD 檔案，請選取「Select XML Catalog

練習 2：寫作一個簡單的 Spring 應用程式

entry」後在 XML Catalog 表格中點選-//SPRING//DTD BEAN 2.0//EN 的項目後按「Next」。

11. 接下來系統要求您設定 Root Element，我們在此採用預設為的「beans」元素，所以直接按「Finish」。按下後 Eclipse 會在 src 下根據 DTD 建立一個 beans-config.xml 檔案的樣版。
12. 打開 beans-config.xml，輸入程式碼 1-4 中的設定：

程式碼 1-4

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"
3           "http://www.springframework.org/dtd/
4           spring-beans-2.0.dtd" >
5 <beans>
6   <bean id="calculator" class="lab1.Adder" />
</beans>
```

13. 最後我們建立一個 Client 類別來測試 Calculator 的功能。

14. 在 Package View 中，在 src 上按右鍵→「New」→「Class」，根據下列資料建立一個 Client 類別：

Source folder : Lab1/src

Package : lab1

Name : Client

Which method stubs would you like to create : 勾選「public static void main(String[] args)」

15. 寫作 Client 類別的內容(請參考程式碼 1-5)：

程式碼 1-5

```

1 public class Client{
2     private static final int INPUT1 = 3;
3     private static final int INPUT2 = 6;
4     public static void main(String[] args){
5         ApplicationContext container = new
6             ClassPathXmlApplicationContext(
7                 "beans-config.xml");
8         Calculator calculator = (Calculator)
9             container.getBean("calculator");
10        System.out.println(
11            INPUT1+calculator.getSymbol()+INPUT2+
```

練習 2：寫作一個簡單的 Spring 應用程式

```

        "="+calculator.calculate(INPUT1, INPUT2));
    }
}

```

16. 在 Client 上按右鍵 → 「Run as...」→ 「Java Application」

執行程式，可以看到輸出結果為「 $3+6=9$ 」

17. 修改 Bean 設定檔將 Calculator 實作品修改為 Mulpilier 類別(程式碼 1-6)：

程式碼 1-6

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"
3           "http://www.springframework.org/dtd/
4           spring-beans-2.0.dtd" >
5 <beans>
6   <bean id="calculator" class="lablex2.Multipiler" />
</beans>

```

18. 在 Client 上按右鍵 → 「Run as...」→ 「Java Application」

執行程式，可以看到輸出結果為「 $3 \times 6=18$ 」



動動腦－透過 Spring Framework，更動介面實作時只要修改 Bean 設定檔，請思考以下問題：

1. 如果不透過 Spring Framework，Client 類別必須如何改寫？
2. 更動介面實作時必須要動到那裏？
3. 這些類別、介面間的依賴關係因為 Spring 的導入，受到了什麼影響？

實驗二

Spring 元件與容器管理

實驗目標

當完成本實驗後，您將能學習到：

- Bean 容器及 Bean 元件的基本管理技巧。
- 透過 XML Schema 元件來寫作 Bean 設定檔。
- 集合 Bean 元件的管理。
- 透過工廠類別取得 Bean 元件。
- 剖面導向程式設計的二種開發方式。

我們在這個實驗單元中帶領您由基本的 Bean 設定開始，循序練習較進階的管理技巧，在本實驗中也將練習剖面導向程式設計的基本開發方式，順利完成本實驗後，可習得開發基本 Spring 應用程式的能力。

相關資料

準備工作



本實驗假設您已經完成以下事項：

- 依照實驗一的指示完成 Spring 及相關 Eclipse 插件的安裝及設定。

練習 1 : Bean 容器基本操作

在本練習中，您將練習透過二種不同型式的 Bean 容器，進行 Spring 的 Setter 注入及 Constructor 注入練習。

工作項目 1 : 匯入練習專案

1. 將 Lab2 的練習用專案目錄拷貝到 c:\spring-framework 底下，使得其路徑為 c:\spring-framework\Lab2。
2. 將 Lab2 專案檔匯入：在 Package View 空白處按右鍵，選擇「Import...」，在 Import 視窗點選「General」→「Existing Projects into Workspace」，選擇「Next」。
3. 點選「Select root directory:」，按下右方「Browse...」鍵 選取 c:\spring-framework\Lab2，按「OK」。
4. 按「Finish」匯入專案。

工作項目 2 : 使用 BeanFactory

BeanFactory 是 Spring Framework 最基本的 Bean 容器，本工作項目中，您將會練習如何透過 BeanFactory 取得 Bean 元件。

1. 取得 BeanFactory 實體：打開 Task2Client.java，建立一個指向 lab2/ex1/beans-config.xml 的 Resource，傳入 XmlBeanFactory 的建構子以建立 BeanFactory 實體。

提示

```
Resource resource = new  
ClassPathResource(...Bean 設定檔路徑...);  
BeanFactory bf = new XmlBeanFactory(resource);
```

2. 寫作 Bean 設定檔：打開 lab2/ex1/beans-config.xml 檔，參考 lab2.ex1.Product 類別，宣告一個 Bean 並為此 Bean 指定下列二組 Beans：

Bean id:productA

Bean class:lab2.ex1.Product

Product id:10001

Name:dog

練習 1 : Bean 容器基本操作**Description:** a dog

Bean id:productB
Bean class:lab2.ex1.Product
Product id:10002
Name:cat
Description: a cat

提示

```
<bean id="..." class="...">
    <property name="..." value="..." />
    ... (其它 properties) ...
</bean>
```

3. 取得 **Bean** : 打開 Task2Client.java , 寫作一段程式取得 productA , 並將其屬性值列印到 Console 中。

提示

使用 BeanFactory 的 getBean 方法來取得 Bean 元件實體。

4. 測試 **Bean** : 在 Package View 的 Task2Client.java 上按右鍵 , 選取「Run as」→「Java Application」來執行程式並觀察輸出結果。
5. 修改 getBean 的參數以取得 productB , 選取「Run as」→「Java Application」來執行程式並觀察輸出結果。

工作項目 3：使用 ApplicationContext

ApplicationContext 和 BeanFactory 相較之下，是較常被使用的 Bean 容器，本工作項目會請您以和 BeanFactory 類似的方式練習使用 ApplicationContext 做為 Bean 容器。

1. 取 得 **ApplicationContext 實 體** : 以 lab2/ex1/beans-config.xml 做為建構子參數，建立一個 ApplicationContext 的實體。

提示

```
ApplicationContext context = new
```

練習 1 : Bean 容器基本操作

```
ClassPathXmlApplicationContext (...) ;
```

2. 取得 Bean : 打開 Task3Client.java , 寫作一段程式取得 productA , 並將其屬性值列印到 Console 中。
3. 測試 Bean : 在 Package View 的 Task3Client.java 上按右鍵 , 選取「Run as」→「Java Application」來執行程式並觀察輸出結果。

提示

使用 ApplicationContext 的 getBean 方法來取得 Bean 元件實體。

工作項目 4 : Setter 注入

在接下來二個工作項目中，我們將要練習二種主要的依賴注入型式：setter 注入及 constructor 注入。在這個工作項目中新增了一個用來展示 Product 內容的 ProductViewer 介面，它只包含單一的 showProduct 方法。在接下來的練習中，必須將之前設定好的 Product 注入 ProductViewer 介面的實作類別中，如圖 2-1 所示，您將透過二種不同的實作品來實施 setter 注入及 constructor 注入工作。

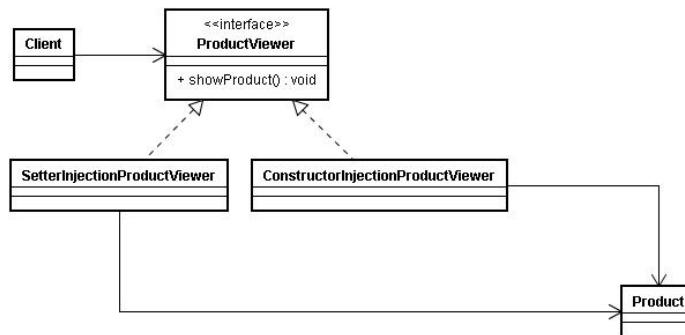


圖 2-1 Add Spring Project Nature

1. 建立 SetterInjectionProductViewer 類別，這個類別必須實作 ProductViewer 介面，並內含一個 Product 類別的實體及其 getter/setter(請參考程式碼 2-1)，以利我們進行 Setter Injection。

練習 1 : Bean 容器基本操作

程式碼 2-1 SetterInjectionProductViewer

```

1 package lab2.ex1;
2
3 public class SetterInjectionProductViewer
4     implements ProductViewer {
5     private Product product;
6
7     public Product getProduct() {
8         return product;
9     }
10
11    public void setProduct(Product product) {
12        this.product = product;
13    }
14
15    public void showProduct() {
16        System.out.println(product);
17    }
18
19 }
20

```

2. 寫作 Bean 設定檔：打開 lab2/ex1/beans-config.xml 檔，宣告 SetterInjectionProductViewer 做為一個 Bean 並將 productA 注入，請參考下列設定資訊：

Bean id:viewer1

Bean class: lab2.ex1.SetterInjectionProductViewer

依賴注入：product 屬性注入 productA

提示

```

<bean id="viewer1" class="...">
    <property name="product">
        <ref bean="..." />
    </property>
</bean>

```

練習 1 : Bean 容器基本操作

3. 建立一個名稱為 Task4Client 的類別，並加入 main()方法。
4. 仿照工作項目 3 在 main() 方法中加入取得 ApplicationContext 實體的程式碼。
5. 取得 Bean：寫作一段程式取得 viewer1，並將其屬性值列印到 Console 中。
6. 測試 Bean：在 Package View 的 Task4Client.java 上按右鍵，選取「Run as」 「Java Application」來執行程式並觀察輸出結果。
7. 打開 lab2/ex1/beans-config.xml 檔，將 viewer1 的 product 屬性改為 productB 後再重覆執行 Task4Client，觀察輸出是否有所不同。



動動腦－使用 Spring 之後，改動所使用的介面實作的代價是否降低了呢？Spring 的程式設計方式在開發過程中產生了什麼助益？

工作項目 5 : Constructor 注入

1. 建立 ConstructorInjectionProductViewer 類別，這個類別必須實作 ProductViewer 介面，它的建構子內含一個 Product 類別的參數(請參考程式碼 2-2)，以利進行 Constructor 注入。

程式碼 2-2 ConstructorInjectionProductViewer

```

1 package lab2.ex1;
2
3 public class ConstructorInjectionProductViewer
4 implements ProductViewer {
5     private Product product;
6     public ConstructorInjectionProductViewer(
7         Product product) {
8         this.product = product;
9     }
10
11    public void showProduct() {
12        System.out.println(product);
13    }
14

```

練習 1 : Bean 容器基本操作

15 }

2. 寫作 Bean 設定檔：打開 lab2/ex1/beans-config.xml 檔，宣告 ConstructorInjectionProductViewer 做為一個 Bean 並將 productA 注入，請參考下列設定資訊：

Bean id:viewer2**Bean class:** lab2.ex1ConstructorInjectionProductViewer
依賴注入：product 屬性注入 productA**提示**

```
<bean id="viewer2"
      class="lab2.ex1.
      ConstructorInjectionProductViewer">
  <constructor-arg ref="productA"/>
</bean>
```

3. 修改 Task4Client 的類別，使其取出 viewer2 並印出所注入 Product 的內容。

提示

修改 ApplicationContext 的 getBean 方法的參數內容。

4. 測試 Bean：在 Package View 的 Task4Client.java 上按右鍵，選取「Run as」、「Java Application」來執行程式並觀察輸出結果。
5. 打開 lab2/ex1/beans-config.xml 檔，將 viewer1 的 product 屬性改為 productB 後再重覆執行 Task4Client，觀察輸出是否有所不同。



動動腦－我們只修改了 Task4Client 中 getBean 方法的參數就可以將 Product 從 setter 注入改為 constructor 注入。想一想使用 Spring 之後在程式設計方式在開發過程中產生了什麼助益？

練習 2 : Bean 元件的管理

本練習的目的是使您瞭解如何透過 XML Schema 簡化 Bean 設定檔，透過 XML Schema 及 XML Namespace 概念的引入，除了可以加入許多新語法之外，開發人員也可自行擴充 Bean 設定檔。此外，您也將練習進階的 Bean 元件管理，包括了集合元件及透過工廠類別來取得 Bean。

工作項目 1：以 XML Schema 建立 Bean 設定檔

Spring Framework 2.0 開始可以採用 XML Schema 來規範 Bean 設定檔，透過 XML Schema 及不同的命名空間，我們可以在 Bean 設定檔中加入許多由 Spring 提供及廠商提供的新功能，而且在語法的撰寫上也較為簡潔。本工作項目會讓您練習如何產生並編輯以 XML Schema 為基礎的 Bean 設定檔。

1. 展開視窗左側 Package View 中的樹狀結構，展開「lab2.ex2」這個 package，您會看到 Task1Client.java 及 Product.java 二個檔案。
2. 「lab2.ex2」這個 package 的圖示上按右鍵按右鍵 → 「New」→ 「Other...」開啟 New 視窗。
3. 在 New 視窗中展開「XML」資料夾，選取「XML」項目後按「Next」。
4. 在 Create XML File 視窗中選取「Create XML file from an XML schema file」，按「Next」。
5. 系統會詢問 Bean 設定檔的位置與檔名，首先將反白移到 Lab2/src/lab2/ex2 上，在 File name 欄位輸入 beans-config.xml，按「Next」。
6. 系統要求您選取 Schema 檔案，請選取「Select XML Catalog entry」後在 XML Catalog 表格中點選 <http://www.springframework.org/schema/beans> 的項目後按「Next」。
7. 接下來系統要求您設定 Root Element，請選取「beans」元素。
8. 在視窗下方的 Namespace Information 中選取 Namespace Name 為 <http://www.springframework.org/schema/beans> 的項目，

練習 2 : Bean 元件的管理

選擇右方的「Edit」，打開 New Namespace Information 視窗。

9. 此時會看到原來的 Prefix 是「p」，將它刪除後按「OK」，此時它會顯示<no prefix>，並在下方出現錯訊息。
10. 接下來選取 Namespace Name 為 <http://www.w3.org/XML/1999> 的項目，選「Edit」。
11. 此時會看到原來的 Prefix 是「p0」，將它改成「xsi」，按「OK」。
12. 接下來選取右方的「Add」來建立一個新的 namespace。
13. 選取 Specify New Namespace，在文字區輸入以下資訊，按「OK」：

Prefix: p

Namespace Name: <http://www.springframework.org/schema/p>

Location Hint: (空白)

14. 按「Finish」之後，Eclipse 會產生一個 beans-config.xml 的樣版，請在<beans>...</beans>間輸入以下資訊：

```
<bean id="productA" class="lab2.ex2.Product">
    <property name="productId" value="10001" />
    <property name="name" value="dog" />
    <property name="description" value="a dog" />
</bean>
```

15. 打開 Task1Client.java，寫作程式碼來取得 productA，並且將它的內容印出。

提示

```
Product productA1 = (Product)
    context.getBean(...);
System.out.println(...);
```

16. 執行 Task1Client，觀察輸出結果。

工作項目 2 : Bean Scope

在這個工作項目中我們將延續上一個工作項目，練習設定

練習 2 : Bean 元件的管理

Spring 中的主要二種 Bean Scope : Singleton 與 Prototype。

- 打開 lab2/ex2/beans-config.xml，加入下列 Bean 設定：

Bean ID: productB

Bean Class: lab2.ex2.Product

Bean Scope: prototype

屬性 productId: 10002

屬性 name: cat

屬性 description: a cat

提示

```
<bean id="..." class="..."  
      scope="...">  
      ...  
</bean>
```

- 打開 lab2/ex2/Task2Client.java，寫作程式碼，連續將 productB 取出二次，存放在二個不同的變數中，並檢驗其記憶體位置是否相同。

提示

```
Product productB1 =...
```

```
Product productB2 =...
```

```
System.out.println("偵測 productB1 與 productB2 的記憶體位置是否相同:" + (productB1 == productB2));
```

- 將「productA」的 Bean 設定改為「singleton」的 scope。

提示

Spring Framework 預設就是 Singleton Scope

- 打開 Task2Client.java，寫作程式碼，連續將 productA 取出二次，存放在二個不同的變數中，並檢驗其記憶體位置是否相同。
- 執行 Task2Client，觀察輸出結果。



動動腦 – 二個 Bean 的記憶體位址相同代表什麼意義？

練習 2 : Bean 元件的管理

工作項目 3：p 命名空間

這個工作項目中我們將練習使用 Spring 的 p 命名空間，善用 p 命名空間可以有效減少 Bean 設定檔的長度，並使得 Bean 設定檔比較容易維護。

- 打開 lab2/ex2/beans-config.xml，以 p 命名空間改寫 productA 的屬性，簡化它的 Bean 設定。

提示

```
<bean id="..." class="..."
      p:productId="..." p:name="..." ... />
```

- 執行 Task1Client，觀察輸出結果。

工作項目 4：集合元件的管理

本工作項目中您會練習如何注入 List 及 Map 二種集合類型的元件，請依照下列步驟操作：

- 打開 lab2/ex2/beans-config.xml，參考 ProductCollection.java 的內容在 Bean 設定檔中宣告一個新的 Bean，其中 list 屬性的型別是 java.util.List，可透過 <list> 將之前宣告的 productA 與 productB 加入：

Bean ID: productList

Bean Class: lab2.ex2.ProductList

屬性 content: 透過<list>加入 productA 及 productB 的參考。

提示

```
<list>
    <ref bean="productA" />
    <ref bean="productB" />
</list>
```

- 執行 Task4Client，觀察輸出結果。

- 接下來為加入 map 的屬性設定，在每個 entry 中，key 值

練習 2 : Bean 元件的管理

請以 bean 的 id 命名(productA,ProductB), value 值請直接傳入 bean 參考。

提示

```
<property name="map">
    <map>
        <entry>
            <key>
                <value>productA</value>
            </key>
            <ref bean="productA" />
        </entry>
        ... (productB 的設定)
    </map>
</property>
```

4. 執行 Task4Client，觀察輸出結果。

工作項目 5：透過工廠(Factory)建立元件

本工作項目中您會練習透過工廠類別及工廠方法來建構 Bean 的實體，請依照下列步驟操作：

1. 打開 lab2/ex2/beans-config.xml，參考 ProductFactory.java 的內容在 Bean 設定檔中宣告一個新的 Bean：

Bean ID: productFactory

Bean Class: lab2.ex2.ProductFactory

2. 接下來宣告一個新的 productBean 元件，設定它的 factory-bean 為 productFactory，factory-method 為 productFactory 的 getInstance 方法。

提示

```
<bean id="productFactory"
      class="lab2.ex2.ProductFactory" />
<bean id="product" class="lab2.ex2.Product"
```

練習 3：剖面導向程式設計

```
factory-bean="..."
factory-method="..." />
```

3. 執行 Task5Client，觀察輸出結果。



動動腦－想一想在什麼情況下我們必須用到 factory-bean 及 facotry-method?

練習 3：剖面導向程式設計

在這個練習中，我們將實作二種型式的 Spring AOP 機制，熟悉了這二種機制後，您可以在不同專案需求中，採取不同的策略。

工作項目 1：透過 XML 開發 AOP 程式

在工作項目 1 中會讓您練習如何以 XML 設定來開發 Spring 的 AOP 應用程式。

1. 請依據程式碼 2-3 建立 PrintBeforeAdvice 類別。

程式碼 2-3 PrintBeforeAdvice

```

1 package lab2.ex3;
2
3 import org.aspectj.lang.JoinPoint;
4
5 public class PrintBeforeAdvice {
6     public void before(JoinPoint jointPoint) {
7         System.out.println(
8             "這是 PrintBeforeAdvice 所加入的訊息：" +
9             jointPoint.getSignature() +
10            getDeclaringTypeName() +
11            "." +
12            jointPoint.getSignature().getName());
13    }
14 }
```

15

2. 打開 lab2/ex3/beans-config.xml，加入 PrintBeforeAdvice 類別的 Bean 宣告。

提示

```
<bean id="printBeforeAdvice"
      class="lab2.ex3.PrintBeforeAdvice" />
```

3. 透過 AOP 標籤，宣告 Aspect 的執行時期，程式碼 2-4，將設定加入 Bean 設定檔中。

程式碼 2-4

```
1 <aop:config>
2   <aop:aspect id="print" ref="printBeforeAdvice">
3     <aop:before pointcut="
4       execution (* lab2.ex3.Product.*(..))"
5         method="before" />
6   </aop:aspect>
7 </aop:config>
```

4. 執行 Task1Client，觀察輸出結果。

工作項目 2：透過 Annotation 開發 AOP 程式

本工作項目中您會練習透過工廠類別及工廠方法來建構 Bean 的實體，請依照下列步驟操作：

1. 請參考程式碼 2-5，在 PrintBeforeAdvice 類別中加入 Annotation。

程式碼 2-3 PrintBeforeAdvice

```
1 package lab2.ex3;
2
3 import org.aspectj.lang.JoinPoint;
4 @Aspect
5 public class PrintBeforeAdvice {
```

練習 3：剖面導向程式設計

```

6   @Before("execution(* lab2.ex3.Product.*(..))")
7   public void before(JoinPoint jointPoint) {
8       System.out.println(
9           "這是 PrintBeforeAdvice 所加入的訊息：" +
10          jointPoint.getSignature() +
11              getDeclaringTypeName() +
12              "." +
13              jointPoint.getSignature().getName());
14      }
15  }

```

2. 打開 `lab2/ex3/beans-config.xml`，將 `<aop:config>...</aop:config>` 部份宣告刪除。在後面加入 `<aop:aspectj-autoproxy />`。
3. 執行 `Task1Client`，觀察輸出結果。



動動腦—討論透過 XML 及 Annotation 二種使用 AOP 的機制的優缺點。

**實驗總結**

在實驗二中我們練習了幾項重要的 Bean 管理技巧，請試著思考下列問題：



動動腦—

1. 開發 Spring 應用程式時，我最常遇到的錯誤是那些？原因是什麼？有沒有方法可以避免或減少這些錯誤。
2. 透過 XML Schema 帶來了那些好處？
3. 那些情況下使用 AOP 技術會很明顯地讓整個應用程式架構變得清晰且容易維護。

實驗三

Spring 與 Java 永續性

實驗目標

當完成本實驗後，您將能學習到：

- 透過 Spring Framework 提供的依賴注入機制取得及操作 DataSource。
- 透過 Spring Framework 提供的依賴注入機制取得及操作支援資料庫連結池的 DataSource。
- Dao 方式實作永續性架構。
- 集合 Bean 元件的管理。
- 使用 JdbcTemplate 及 NamedJdbcTemplate 做資料庫的新增、查詢及更新動作。

相關資料

準備工作



本實驗假設您已經完成以下事項：

- 安裝 MySQL4.1.22 版本至 C:\MySQL 及 MySQL Control Center
- 設定好以下的帳號密碼：
 - 帳號：root
 - 密碼：springframework
- 建立一個名稱為「springdb 的資料庫」

練習 1：取得 DataSource

DataSource 是 Java EE 中取得資料庫連結(Connection)的標準做法。在本練習中，您將透過 Spring 提供的依賴注入機制實際設定並使用 DataSource。

工作項目 1：匯入練習用資料檔

1. 從 Windows 的「開始」→「執行」，打開執行視窗，輸入「cmd」開啟命令列視窗。
2. 將目錄切換至「C:\MySQL」。 (使用 CD 指令)
3. 拷貝專案目錄下的「/sql/springdb.sql」到「C:\MySQL\bin 中」。
4. 在命令列視窗鍵入指令「mysql < springdb.sql -u root -p」。
5. 接下來系統會詢問密碼，請輸入「springframework」。
6. 在命令列視窗鍵入指令「mysql -u root -p」，並輸入密碼，登入 MySQL 的命令列控制介面。
7. 鍵入「use springdb;」。 (記得最後面要加分號)
8. 鍵入「show tables;」，觀察輸出結果。
9. 鍵入「select * from product;」，觀察輸出結果。

工作項目 2：匯入練習專案

1. 將 Lab3 的練習用專案目錄拷貝到 c:\spring-framework 底下，使得其路徑為 c:\spring-framework\Lab3。
2. 將 Lab3 專案檔匯入：在 Package View 空白處按右鍵，選擇「Import...」，在 Import 視窗點選「General」→「Existing Projects into Workspace」，選擇「Next」。
3. 點選「Select root directory:」，按下右方「Browse...」鍵選取 c:\spring-framework\Lab3，按「OK」。
4. 按「Finish」匯入專案。

工作項目 3：建立及測試 DataSource

在這個工作項目中您將會練習如何透過 Spring Framework 取得 DriverManagerDataSource。

練習 1：取得 DataSource

1. 在 lab3/ex1/下建立一個 beans-config.xml 的設定檔。(請參考實驗二，透過 New→Other→XML 使用精靈來建立)
2. 在 Bean 設定檔中設定一個名為「dataSource」的資料來源，設定如下：

Bean id:dataSource

Bean class:org.springframework.jdbc.datasource.DriverManagerDataSource

Bean class 的 destroy-method: close

(下面為 dataSource 的屬性)

driverClassName: com.mysql.jdbc.Driver

url: jdbc:mysql://localhost:3306/springdb

username: root

password: springframework

提示

```
<bean id="..." class="..." destroy-method="...">
    <property name="..." value="..." />
    ... (其它 properties) ...
</bean>
```

3. 打開 Task3Client.java，寫作一段程式來測試 ProductDao 介面的三個方法。

提示

使用 **getBean** 方法來取得 Bean 元件實體。

4. **測試 Bean :** 在 Package View 的 Task3Client.java 上按右鍵，選取「Run as」→「Java Application」來執行程式並觀察輸出結果。

工作項目 4：使用資料庫連結池做為資料來源

1. 在 Bean 設定檔中加入一個名為「pooledDataSource」的 Bean，其設定和「dataSource」相同，但 Bean class 的名稱改為「org.apache.commons.dbcp.BasicDataSource」，並加

練習 1：取得 DataSource

下列額外的資料庫連結池屬性：

initialSize: 2 (一開始預建 2 個資料庫連結)

maxActive: 5 (最多保留 5 個資料庫連結)

提示

可參考 <http://jakarta.apache.org/commons/dbcp/configuration.html> 以取得更詳細的連結池參數設定資訊。

2. 取得 Bean：寫作一段程式命名為 Task4Client.java，用來取得 pooledDataSource，並將其屬性值列印到 Console 中。

提示

可參考 Task3Client.java，並修改 Bean 的名稱。

3. 測試 Bean：在 Package View 的 Task4Client.java 上按右鍵，選取「Run as」→「Java Application」來執行程式並觀察輸出結果。
4. 測試資料庫連結的最大限度：之前我們透過設定 maxActive 為 5，現在我們測試如果同時超過 6 個資料庫連結會發生什麼事，首先在 Task4Client.java 中加入下列程式碼。

```
try {
    Connection c1 = ds.getConnection();
    Connection c2 = ds.getConnection();
    Connection c3 = ds.getConnection();
    Connection c4 = ds.getConnection();
    Connection c5 = ds.getConnection();
} catch (SQLException e) {
    e.printStackTrace();
}
```

5. 在 Package View 的 Task4Client.java 上按右鍵，選取「Run as」→「Java Application」來執行程式並觀察輸出結果。
6. 再加入一行：「Connection c6 = ds.getConnection();」
7. 再度執行 Task4Client.java，觀察結果有什麼不同。

提示

練習 2：透過 DAO 管理物件的永續性

當所要求的資料庫連結數量超過 maxActive 的設定值時，系統會將後來才進來的要求 block 住，直到之前的資料庫連結被 release 為止。



重點提示— 請務必察看專案目錄下的/lib 目錄，確認您瞭解進行上述資料庫操作所需要引入的函式庫：

1. mysql 的 JDBC 驅動程式。
2. commons-pool.jar。
3. commons-dbcp.jar。

練習 2：透過 DAO 管理物件的永續性

在本練習中您將熟悉如何透過 DAO 架構樣式來存取資料庫中的領域模型。

工作項目 1：建立 ProductDao 介面

1. 在 Package Explorer 中，切換至 lab3.ex2 中。
2. 根據下列規格，建立 ProductDao 介面：

Interface 名稱：ProductDao

Interface 方法：

方法名稱	參數	回傳值
getProduct	int id	Product
getProducts	無	List<Product>

工作項目 2：建立 ProductRowMapper 類別

1. 根據下列規格，建立 ProductRowMapper 類別：

類別名稱：ProductRowMapper

Implements : org.springframework.jdbc.core.

練習 2：透過 DAO 管理物件的永續性

RowMapper

類別方法：

方法名稱	參數	回傳值
mapRow	ResultSet rs	Object
throws	int rowCount	
SQLException		

2. 在 mapRow 中輸入程式碼，將 ResultSet 中的資料對映到 Product 中。

提示

```
Product product = new Product();
product.setProductId(rs.getInt("id"));
product.setDescription(rs.getString("desc"));
product.setName(rs.getString("name"));
return product;
```

工作項目 3：建立 ProductDao 介面實作類別

1. 根據下列規格，建立 ProductDaoImpl 類別：

類別名稱：ProductDaoImpl

Extends : NamedParameterJdbcDaoSupport**Implements** : ProductDao

類別屬性：

屬性型別	屬性名稱	保護等級
ProductRowMapper	productRowMapper	private

類別方法：

方法名稱	參數	回傳值
getProduct	int id	Product
getProducts	無	List<Product>
setProductRowMapper	ProductRowMapper	無 productRowMapper

練習 2：透過 DAO 管理物件的永續性

提示

加入 `implements` 敘述後，Eclipse 會出現錯誤提示，直接在錯誤提示上按左鍵，選擇適當的處理方式就可以自動依照實作介面產生相對應的類別方法。

為了測試方便，可以先暫時使有回傳值的方法回傳 `null`。



動動腦 – 查閱一下 `NamedParameterJdbcDaoSupport` 的 Javadoc，看看它提供了那些方法。

2. 實作 `setProductRowMapper` 方法，將注入的 `ProductRowMapper` 實體設定到 `productRowMapper` 屬性中。

提示

```
this.productRowMapper = productRowMapper;
```

3. 使用 `JdbcTemplate` 類別的 `query` 方法來實作 `getProducts`，`getProducts` 主要目的為取得 `product` 資料表中所有資料，並透過 `ProductRowMapper` 將之轉換成 `Product` 類別實體後回傳。`query` 方法的簽章如下：

```
List query(String sql, RowMapper rowMapper)
```

提示

```
return (List<Product>) getJdbcTemplate().query(
    "select * from product", productRowMapper);
```

4. 打開 `beans-config.xml`，設定一個名稱為 `productDao` 的 bean，並將 `dataSource` 與 `productRowMapper` 注入。

提示

```
<bean id="productDao"
      class="lab3.ex2.ProductDaoImpl">
    <property name="dataSource" ref="dataSource" />
    <property name="productRowMapper"
              ref="productRowMapper" />
```

練習 2：透過 DAO 管理物件的永續性

</bean>

5. 打開 Task3Client，加入程式碼來測試 getProducts 方法。
6. 在 Package View 的 Task4Client.java 上按右鍵，選取「Run as」→「Java Application」來執行程式並觀察輸出結果。
7. 使 用 NamedParamenterJdbcTemplate 類 別 的 queryForObject 方法來實作 getProduct，getProduct 的主要目的是將所傳入的參數做為 key，從資料庫中取得一筆符合的資料，透過 ProductRowMapper 將之轉換成 Product 類別實體並回傳。queryForObject 方法的簽章如下：

```
public Object queryForObject(
    String sql,
    Map paramMap,
    RowMapper rowMapper)
```

提示
請特別留意 SQL 語法中指定參數的方式。
`SqlParameterSource namedParameters =`
`new MapSqlParameterSource(`
 `"productId", productId);`

```
return (Product) getNamedParameterJdbcTemplate().
    queryForObject(
        "select * from product where id
        = :productId", namedParameters,
        productRowMapper);
```

8. 打開 Task3Client，加入程式碼來測試 getProduct 方法。
9. 在 Package View 的 Task3Client.java 上按右鍵，選取「Run as」→「Java Application」來執行程式並觀察輸出結果。



動動腦 – 想一想，為什麼在這裏我們要使用 NamedParamenterJdbcTemplate 類別，而不是一般的 JdbcTemplate？

工作項目 4：新增資料

練習 2：透過 DAO 管理物件的永續性

- 在 ProductDao 中加入一個可以讓我們加入新的 Product 的方法宣告(回傳值 int 用來取得資料庫自動產生的 key)：

```
public int insert(Product product);
```

- 在 ProductDaolmpl 類別中利用 SqlParameterSource 及 NamedParameterJdbcTemplate 的 update 方法 實作 insert()。

提示

請特別留意語法中指定參數的方式以及它們彼此的對應。

```
KeyHolder keyHolder = new GeneratedKeyHolder();
SqlParameterSource params = new
MapSqlParameterSource("name",
product.getName()).addValue(
"desc",product.getDescription());
getNamedParameterJdbcTemplate().update(
"insert into product (`name`, `desc`)
values (:name,:desc)",params,keyHolder);
return keyHolder.getKey().intValue();
```

- 打開 Task4Client，加入程式碼來測試 getProduct 方法。

提示

```
Product p = new Product();
p.setName("dog2");
p.setDescription("dog no. 2");
int key = productDao.insert(p);
System.out.println(key);
```

- 在 Package View 的 Task4Client.java 上按右鍵，選取「Run as」 「Java Application」來執行程式並觀察輸出結果。

工作項目 5：更新資料

練習 2：透過 DAO 管理物件的永續性

- 在 ProductDao 中加入一個可以讓我們加入新的 Product 的方法宣告：

```
public void update(Product product);
```

- 在 ProductDaolmpl 類別中利用 SqlParameterSource 及 NamedParameterJdbcTemplate 的 update 方法 實作 update()。

提示

請特別留意語法中指定參數的方式以及它們彼此的對應。

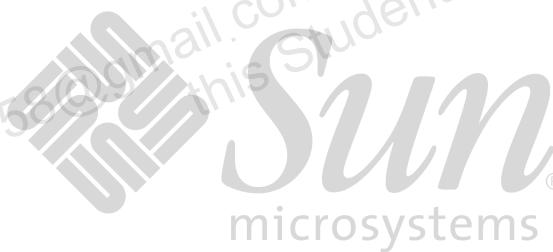
```
SqlParameterSource params =
    new MapSqlParameterSource("id",
        product.getProductId())
    .addValue("name", product.getName())
    .addValue("desc", product.getDescription());
getNamedParameterJdbcTemplate().update(
    "update product set product.name=:name,
    product.desc=:desc where id=:id", params);
```

- 打開 Task5Client，加入程式碼來將 productId 為 1 的貨物 name 改為「dog3」，description 改為「dog no. 3」。

提示

```
Product p = new Product();
p.setProductId(1);
p.setName("dog3");
p.setDescription("dog no. 3");
productDao.update(p);
```

- 在 Package View 的 Task5Client.java 上按右鍵，選取「Run as」 「Java Application」來執行程式並觀察輸出結果。



Tian Feng (tim9958@gmail.com) has a non-transferable license to
use this Student Guide.

實驗四

Spring 與交易管控

實驗目標

當完成本實驗後，您將能學習到下列三種控制資料庫交易的方式：

- 以程式方式控制資料庫交易。
- 以 Spring Framework 內建的 tx 標籤控制資料庫交易。
- 以 Annotation 控制資料庫交易。

除了第一種方式之後，這裏所練習的機制都能夠達成透通的交易控制，也就是在不影響原有的程式碼的前提下，完成交易屬性及交易範圍的設定。

相關資料

準備工作



本實驗假設您已經完成以下事項：

- 安裝 MySQL4.1.22 版本至 C:\MySQL 及 MySQL Control Center
- 設定好以下的帳號密碼：
 - 帳號：root
 - 密碼：springframework
- 建立一個名稱為「springdb 的資料庫」
- 根據實驗三的指示，匯入 springdb.sql。

練習 1：由程式控制資料庫交易

在本練習中，您將練習透過程式來控制資料庫的交易。

工作項目 1：修改資料表型別

由於 MySQL 資料庫預設的 MyISAM 資料表型別不支援交易功能，所以我們必須先將 product 資料表改成支援交易的 InnoDB 型別。

1. 從 Windows 的「開始」→「執行」，打開執行視窗，輸入「cmd」開啟命令列視窗。
2. 將目錄切換至「C:\MySQL\bin」。(使用 CD 指令)
3. 在命令列視窗鍵入指令「mysql -u root -p」。
4. 接下來系統會詢問密碼，請輸入「springframework」。
5. 鍵入「use springdb;」。(記得最後面要加分號)
6. 鍵入「show table status;」，確定 product 資料表存在，否則請參閱實驗三的練習一重新匯入 springdb.sql。
7. 鍵入「ALTER TABLE product ENGINE=InnoDB」，觀察輸出結果。
8. 鍵入「show table status;」，確定 product 資料表的 Engine 已改為 InnoDB。

提示

如果輸入 show table stauts 後畫面亂掉，可調整命令列內容的版面設定，將螢幕緩衝區大小的寬度加到大約 300 左右即可。

工作項目 2：匯入練習專案

1. 將 Lab4 的練習用專案目錄拷貝到 c:\spring-framework 底下，使得其路徑為 c:\spring-framework\Lab4。
2. 將 Lab4 專案檔匯入：在 Package View 空白處按右鍵，選擇「Import...」，在 Import 視窗點選「General」→「Existing Projects into Workspace」，選擇「Next」。
3. 點選「Select root directory:」，按下右方「Browse...」鍵選取 c:\spring-framework\Lab4，按「OK」。
4. 按「Finish」匯入專案。

練習 1：由程式控制資料庫交易

工作項目 3：交易控制

1. 打開 lab4/ex1/下的 beans-config.xml 設定檔。
2. 在 Bean 設定檔中加入一個名為「txManager」的交易管理員，設定如下：

```
Bean id:txManager
Bean class:org.springframework.jdbc.
    datasource.DataSourceTransactionManager
dataSource: dataSource
```

提示

```
<bean id="..." class="...">
    <property name="..." ref="..." />
</bean>
```

3. 取得交易管理員：打開 Task3Client.java，寫作一段程式取得 PlatformTransactionManager 的實體。

提示

```
PlatformTransactionManager txManager =
    (PlatformTransactionManager)
        context.getBean("txManager");
```

4. 宣告交易區間：加入程式碼來宣告一個交易區間(請參考程式碼 4-1)。請特別注意 txManager 的 getTransaction、rollback 及 commit 方法呼叫的位置。

程式碼 4-1

```
1 TransactionDefinition txDef = new
2                     DefaultTransactionDefinition();
3 TransactionStatus txStatus =
4                     txManager.getTransaction(txDef);
5 try {
6     // 這裏是交易區間
7 } catch (Exception ex) {
8     ex.printStackTrace();
```

練習 1：由程式控制資料庫交易

```

9     txManager.rollback(txStatus);
10 }
11     txManager.commit(txStatus);

```

5. 加入交易邏輯：在交易區間中加入一段程式碼，將 productId 為 1 的貨品名稱改為「myname」，在最後加入一行 `raiseException()` 敘述，故意丟出例外。

提示

```

Product p = productDao.getProduct(1);
p.setName("myname");
productDao.update(p);
raiseException();

```

6. 執行 Task3Client，並透過 MySQL 命令列觀察交易是否確實被 rollback。

提示

- 透過 mysql -u root -p 進入 MySQL 命令列介面
 - use springdb;
 - select * from product;
 - 觀察 id 為 1 的貨品欄位，如果沒有被改成「myname」即為正確。
-

7. 將 `raiseException()` 註解起來，再重複執行一次程式，並透過 MySQL 命令列觀察交易是否成功。

工作項目 4：TransactionTemplate

在這個工作項目中，您將透過 Spring 內建的 TransactionTemplate 來控制交易。

- 打開 lab4/ex1/下的 beans-config.xml 設定檔。
- 在 Bean 設定檔中加入一個名為「txTemplate」的 TransactionTemplate，設定如下：

Bean id:txTemplate

練習 1：由程式控制資料庫交易

Bean class: org.springframework.transaction.support
.TransactionTemplate
transactionManager: transactionManager

提示

```
<bean id="..." class="...">
    <property name="..." ref="..." />
</bean>
```

3. 打開 Task4Client.java，使用 TransactionTemplate 告知一個交易區間。(程式碼 4-2)

程式碼 4-2

```
1 TransactionTemplate txTemplate =
2             (TransactionTemplate)
3             context.getBean("txTemplate");
4 txTemplate.execute(new TransactionCallback() {
5     public Object doInTransaction(
6         TransactionStatus status) {
7         // 這裏是交易區間
8         return ...;
9     } // end doInTransaction()
10 });
11
```

4. 加入交易邏輯:在交易區間中加入一段程式碼，將 productId 為 1 的貨品名稱改為「myname-**txtemplate**」，在最後加入一行 raiseException 敘述，故意丟出例外。

提示

```
Product p = productDao.getProduct(1);
p.setName("myname-txtemplate");
productDao.update(p);
raiseException();
```

5. 執行 Task4Client，並透過 MySQL 命令列觀察交易是否確實被 rollback。

練習 2：透過 tx 標籤進行交易管理

提示

- a. 透過 mysql -u root -p 進入 MySQL 命令列介面
- b. use springdb;
- c. select * from product;
- d. 觀察 id 為 1 的貨品欄位，如果沒有被改成
「myname-txtemplate」即為正確。

6. 將 raiseException() 註解起來，再重複執行一次程式，並透過 MySQL 命令列觀察交易是否成功。

練習 2：透過 tx 標籤進行交易管理

在這個練習中，您將透過 Spring 的 tx 標籤來管理交易的進行。

首先我們必須建立 TxProductService 類別，再透過 Bean 設定檔中的 tx 標籤來設定交易的範圍。

工作項目 1：建立 TxProductService 類別

1. 在 Package Explorer 中，切換至 lab4.ex2。
2. 根據下列規格，建立 TxProductService 類別：

類別名稱：TxProductService

類別屬性：

屬性型別	屬性名稱	保護等級
ProductDao	productDao	private

類別方法：

方法名稱	參數	回傳值
setProductDao	ProductDao	無
	productDao	
testTx	無	無
raiseException	無	無

3. 在 setProductDao() 中輸入程式碼，將傳入的實體設定到

練習 2：透過 tx 標籤進行交易管理

productDao 私有屬性中。

提示

```
this.productDao = productDao;
```

4. 在 `raiseException()` 中輸入程式碼，丟出一個 `RuntimeException`。

提示

```
throw new RuntimeException();
```

5. 在 `testTx()` 中輸入程式碼，加入交易邏輯。將 `productId` 為 1 的貨品名稱改為「`myname-txtemplate`」，在最後加入一行 `raiseException` 敘述，故意丟出例外。

提示

```
Product p = productDao.getProduct(1);
p.setName("myname-tx");
productDao.update(p);
raiseException();
```

工作項目 2：透過 tx 標籤管理交易

1. 開啟 `lab4/ex2/beans-config.xml` 檔案。
2. 在 Bean 設定檔中設定「`tx:advice`」標籤：

id:txAdvice

transaction-manager:txManager

3. 在 `<tx:advice>` 中加入一個 `<tx:attributes>` 元素，並在 `<tx:attributes>` 中加入 `<tx:method>`，其屬性設定如下：

name: test*

propagation:REQUIRED

isolation:DEFAULT

提示

練習 2：透過 tx 標籤進行交易管理

```
<tx:advice id="txAdvice"
    transaction-manager="txManager">

<tx:attributes>
    <tx:method name="test*" propagation="REQUIRED"
        isolation="DEFAULT" />
</tx:attributes>
</tx:advice>
```

4. 在 Bean 設定檔中設定「aop:config」標籤，並在 `<aop:config>` 中加入 `<aop:advisor>`，其屬性設定如下::

pointcut: execution(* lab4.ex2.TxProductService.*(..))
advice-ref: txAdvice

提示

```
<aop:config>
    <aop:advisor pointcut="execution(
        * lab4.ex2.TxProductService.*(..))"
        advice-ref="txAdvice" />
</aop:config>
```

5. 執行 Task1Client，並透過 MySQL 命令列觀察交易是否確實被 rollback。

提示

- a. 透過 mysql -u root -p 進入 MySQL 命令列介面
 - b. use springdb;
 - c. select * from product;
 - d. 觀察 id 為 1 的貨品欄位，如果沒有被改成「myname-tx」即為正確。
-

6. 將 `raiseException()` 註解起來，再重複執行一次程式，並透過 MySQL 命令列觀察交易是否成功。



重點提示 – 請務必察看專案目錄下的/lib 目錄，確認您瞭解進行宣告式交易管理所需要的函式庫：

1. mysql 的 JDBC 驅動程式。

練習 3：透過 Annotation 進行交易管理

-
2. aspectjrt.jar。
 3. aspectjweaver.jar。
 4. cglib-nodep-2.1_3
-

練習 3：透過 Annotation 進行交易管理

在這個練習中，您將透過 JDK 5.0 新增的 Annotation 特色管理交易的進行。首先我們必須改寫 TxProductService 類別，再透過 Bean 設定檔中的<tx:annotation-driven>標籤來啟動 Annotation 交易機制。

工作項目 1：修改 TxProductService 類別

1. 在 Package Explorer 中，切換至 lab4.ex3。
2. 在 TxProductService 類別的 textTx()方法的上方加入下列 Annotation：

```
@Transactional(  
    readOnly = false,  
    propagation = Propagation.REQUIRED,  
    isolation = Isolation.DEFAULT)
```

工作項目 2：修改 Bean 設定檔

1. 開啟 lab4/ex3/beans-config.xml 檔案。
2. 在 Bean 設定檔中加入下列設定：

```
<tx:annotation-driven  
    transaction-manager="txManager" />
```

3. 執行 Task1Client，並透過 MySQL 命令列觀察交易是否確實被 rollback。

提示

- a. 透過 mysql -u root -p 進入 MySQL 命令列介面
- b. use springdb;
- c. select * from product;

練習 3：透過 Annotation 進行交易管理

-
- d. 觀察 id 為 1 的貨品欄位，如果沒有被改成「myname-tx」即為正確。
-

- 4. 將 `raiseException()` 註解起來，再重複執行一次程式，並透過 MySQL 命令列觀察交易是否成功。



動動腦－比較一下練習 1 到 3 的三種交易管理機制，它們的適用場合各是什麼？



Tian Feng (tim9958@gmail.com) has a non-transferable license to
use this Student Guide.

實驗五

Spring 與 Hibernate 整合應用

實驗目標

當完成本實驗後，您將能學習到：

- 透過 XML 來設定物件-關連式對映的方式進行 Spring-Hibernate 的整合。
- 透過 Annotation，直接將物件-關連式對映內嵌在類別定義中，並進行 Spring-Hibernate 的整合。
- Hibernate 的交易管理。

您這個實驗單元中將實際練習二種主要的 Spring-Hibernate 整合機制，此外也將實際操作在 Hibernate 與 Spring 宣告式交易的整合方式。

相關資料

準備工作



本實驗假設您已經完成以下事項：

- 安裝 MySQL4.1.22 版本至 C:\MySQL 及 MySQL Control Center
- 設定好以下的帳號密碼：
 - 帳號：root
 - 密碼：springframework
- 建立一個名稱為「springdb2 的資料庫」

練習 1：以 XML 對映檔設定 ORM

工作項目 1：匯入練習用資料檔

1. 從 Windows 的「開始」→「執行」，打開執行視窗，輸入「cmd」開啟命令列視窗。
2. 將目錄切換至「C:\MySQL」。 (使用 CD 指令)
3. 拷貝專案目錄下的「/sql/springdb2.sql」到「C:\MySQL\bin 中」。
4. 在命令列視窗鍵入指令「mysql < springdb2.sql -u root -p」。
5. 接下來系統會詢問密碼，請輸入「springframework」。
6. 在命令列視窗鍵入指令「mysql-u root -p」，並輸入密碼，登入 MySQL 的命令列控制介面。
7. 鍵入「use springdb2;」。 (記得最後面要加分號)
8. 鍵入「show tables;」，觀察輸出結果。
9. 鍵入「select * from product;」，觀察輸出結果。

工作項目 2：匯入練習專案

1. 將 Lab5 的練習用專案目錄拷貝到 c:\spring-framework 底下，使得其路徑為 c:\spring-framework\Lab5。
2. 將 Lab5 專案檔匯入：在 Package View 空白處按右鍵，選擇「Import...」，在 Import 視窗點選「General」→「Existing Projects into Workspace」，選擇「Next」。
3. 點選「Select root directory:」，按下右方「Browse...」鍵選取 c:\spring-framework\Lab5，按「OK」。
4. 按「Finish」匯入專案。

工作項目 3：觀察領域模型及 DAO 介面

1. 開啟 lab5.ex1.Product 類別的原始碼，觀察一下 Product 所具有的屬性，在第五章中所用的 Product 類別和前幾章有所不同。
2. 開啟 lab5.ex1.ProductDao 介面的原始碼，觀察一下 ProductDao 介面具有那些方法。

練習 1：以 XML 對映檔設定 ORM



重點提示— 請務必察看專案目錄下的/lib 目錄，確認您瞭解進行 Hibernate 相關操作所需要引入的函式庫(xxx 部份可能依 Hibernate3 的各子版本不同而有不同編號)：

1. hibernate3.jar
2. cglib-x.x.x.jar
3. antlr.x.x.x.jar
4. asm-attrs.jar
5. asm.jar
6. commons-logging-x.x.x.jar
7. commons-collections-x.x.x.jar
8. dom4j-x.x.x.jar
9. log4j-x.x.x.jar
10. jta.jar

工作項目 4：寫作 Product 類別的 ORM 設定檔

在這個工作項目中我們將以 XML 的型式撰寫 Product 類別的設定檔案。

1. 在 lab5/ex1/下新增一個「Product.hbm.xml」的空白 XML 檔案。
2. 在標頭加入下列宣告：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
"-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/
hibernate-mapping-3.0.dtd">
```

3. 在 Product.hbm.xml 檔案中加入 Product 類別的 ORM 敘述：

```
<hibernate-mapping>
  <class name="lab5.ex1.Product" table="product">
    <id name="productId" type="int" column="id">
      <generator class="native" />
```

練習 1：以 XML 對映檔設定 ORM

```

</id>
<property name="name" column="name"
          type="string" />
<property name="price" column="price"
          type="int" />
</class>
</hibernate-mapping>

```

工作項目 5：實作 ProductDao

在這個工作項目中我們將使用 Spring 提供的 HibernateDaoSupport 及 HibernateTemplate 建立 ProductDao 的實作類別。

- 根據下列規格，建立 ProductDaoHibernateImpl 類別：

類別名稱：ProductDaoHibernateImpl

Extends：HibernateDaoSupport

Implements：ProductDao

類別方法：

方法名稱	參數	回傳值
getProduct	int id	Product
getProducts	無	List<Product>
save	Product product	無

提示

加入 implements 敘述後，Eclipse 會出現錯誤提示，直接在錯誤提示上按左鍵，選擇適當的處理方式就可以自動依照實作介面產生相對應的類別方法。

為了測試方便，可以先暫時使有回傳值的方法回傳 null。

- 呼叫 HibernateDaoSupport 的 getHibernateTemplate()，透過 Criteria 的方式實作 getProduct 方法。

練習 1：以 XML 對映檔設定 ORM**提示**

```

return (Product) getHibernateTemplate().execute(
    new HibernateCallback() {
        public Object doInHibernate(Session session)
            throws HibernateException
        {
            return session.createCriteria(Product.class)
                .add(Expression.eq("id", id))
                .uniqueResult();
        }
    });
}

```

請您留意這段程式碼的幾個重點：

- a. HibernateTemplate 及 HibernateCallBack 的配合。
- b. 如何透過 session 建構 Criteria。
- c. 使用 uniqueResult()來限制 session 回傳單一的結果。
3. 透過 HQL 的方式來實作 getProducts 方法。

提示

```

return (List<Product>)
    getHibernateTemplate().find("from Product p");

```



動動腦－比較以上以 Criteria 為主及以 HQL 為主的資料查詢方法，他們的適用場合各在那裏。

4. 透過 HibernateTemplate 的 saveOrUpdate()，實作 save 方法。

提示

```
getHibernateTemplate().saveOrUpdate(product);
```

工作項目 6：寫作 Bean 設定檔

- 打開 lab5/ex1/中的 bean-config.xml 檔案，加入一個名為「sessionFactory」的 Bean，設定如下：

Bean id: sessionFactory

Bean class: org.springframework.orm
.hibernate3.LocalSessionFactoryBean

Bean class 的 destroy-method: close

(下面為屬性設定)

dataSource: dataSource (ref)

mappingResources(List): (1)lab5/ex1/Product.hbm.xml

hibernateProperties:

(1)hibernate.dialect=org.hibernate.dialect.MySQLDialect

(2)hibernate.show_sql=true

提示

請留意 List 及 Properties 的設定方式

```
<bean id="sessionFactory"
      class="..." destroy-method="...">
    <property name="dataSource" ref="..." />
    <property name="mappingResources">
      <list>
        <value>
          lab5/ex1/Product.hbm.xml
        </value>
      </list>
    </property>
    <property name="hibernateProperties">
      <value>
        hibernate.dialect=org.hibernate
          .dialect.MySQLDialect
        hibernate.show_sql=true
      </value>
    </property>
  </bean>
```

練習 2：以 Annotation 設定 ORM

-
2. 打開 lab5/ex1/Client.java，寫作一段程式來測試 ProductDao 介面的三個方法。

提示

```
ProductDao productDao = (ProductDao)
context.getBean("productDao");

Product product1 = productDao.getProduct(1);
product1.setName("Hibernate");
product1.setPrice(120);
productDao.save(product1);

List<Product> list = productDao.getProducts();
... (利用 for 迴圈列印所有元素) ...
```

3. 測試 Bean：在 Package View 的 Client.java 上按右鍵，選取「Run as」→「Java Application」來執行程式並觀察輸出結果。

練習 2：以 Annotation 設定 ORM**工作項目 1：為 Product 類別加入 Annotation**

在這個工作項目中我們將以 Annotation 型式來設定 Product 類別的 ORM 設定。

1. 開啟 lab5/ex2/下的 Product 類別。
2. 為 Product 類別加入 ORM 的 Annotation。

```
@Entity
@Table(name = "product")
public class Product {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "id")
```

練習 2：以 Annotation 設定 ORM

```

private int productId;

@Column(name = "name")
private String name;

@Column(name = "price")
private int price;

...(其它 getter 及 setter 方法)...
}

```

工作項目 2：實作 ProductDao

- 根據下列規格，建立 ProductDaoHibernateImpl 類別：

類別名稱：ProductDaoHibernateImpl

Extends : HibernateDaoSupport

Implements : ProductDao

類別方法：

方法名稱	參數	回傳值
getProduct	int id	Product
getProducts	無	List<Product>
save	Product product	無

- 模仿 lab5/ex1 中的 ProductDaoHibernateImpl 將同樣的實作方式完成各項方法的實作。

工作項目 3：寫作 Bean 設定檔

- 打開 lab5/ex2/中的 bean-config.xml 檔案，加入一個名為「sessionFactory」的 Bean，設定如下：

Bean id: sessionFactory

Bean class: org.springframework.orm.hibernate3

.annotation.AnnotationSessionFactoryBean

Bean class 的 destroy-method: close

練習 3 : Hibernate 交易管理

(下面為屬性設定)

dataSource: dataSource (ref)
annotatedClasses (List): (1) lab5.ex2.Product
hibernateProperties:
(1) hibernate.dialect=org.hibernate.dialect.MySQLDialect
(2) hibernate.show_sql=true

提示

```
<bean id="sessionFactory"
      class="..." destroy-method="...">
    <property name="dataSource" ref="..." />
    <property name="annotatedClasses">
        <list>
            <value>...</value>
        </list>
    </property>
    <property name="hibernateProperties">
        ...
    </property>
</bean>
```

2. 打開 lab5/ex2/Client.java，寫作一段程式來測試 ProductDao 介面的三個方法。
3. 在 Package View 的 Client.java 上按右鍵，選取「Run as」→「Java Application」來執行程式並觀察輸出結果。

練習 3 : Hibernate 交易管理

工作項目 1：修改 TxProductService 類別

1. 在 Package Explorer 中，切換至 lab5.ex3。
2. 在 TxProductService 類別的 textTx()方法上方加入下列 Annotation :

```
@Transactional(
    readOnly = false,
```

練習 3 : Hibernate 交易管理

```
propagation = Propagation.REQUIRED,
isolation = Isolation.DEFAULT)
```

工作項目 2：修改 Bean 設定檔

1. 開啟 lab5/ex3/beans-config.xml 檔案。
2. 在 Bean 設定檔中加入下列設定：

```
<tx:annotation-driven
    transaction-manager="txManager" />

<bean id="txManager"
    class="org.springframework.orm
        .hibernate3.HibernateTransactionManager">
    <property name="sessionFactory"
        ref="sessionFactory" />
</bean>
```

3. 執行 Client 類別，並透過 MySQL 命令列觀察交易是否確實被 rollback。

提示

- a. 透過 mysql -u root -p 進入 MySQL 命令列介面
 - b. use springdb2;
 - c. select * from product;
 - d. 觀察 id 為 1 的貨品欄位，如果沒有被改成「myname」即為正確。
-
4. 將 raiseException()註解起來，再重複執行一次程式，並透過 MySQL 命令列觀察交易是否成功。



Tian Feng (tim9958@gmail.com) has a non-transferable license to
use this Student Guide.

實驗六

Spring 與 Web 應用程式開發

實驗目標

當完成本實驗後，您將能學習到：

- 安裝及設定動態 Web 專案的開發環境。
- 使用 Eclipse IDE 實作動態 Web 應用程式。
- 寫作一個簡單的 Spring MVC 應用程式。
- 透過 ApplicationContext 及 ContextListener 整合 Spring 與 Struts。
- Spring 與 Struts 的非侵入性整合方式。

我們在這個實驗單元中將帶領您由最簡單 Web 應用程式開始，實際體驗 Spring MVC 及 Spring 與 Struts 的二種整合方式，讓您可以在實際應用挑選最合適的解決方案。

相關資料

準備工作



本實驗假設您已經完成以下事項：

- 已經在 Eclipse 上安裝了 WST (Web Standard Tools)。如果您在實驗一中曾經安裝過 JST (J2EE Standard Tools)，則 WST 就已自動被安裝完成，不需再另行安裝。
- 下傳 Apache Tomcat 5.5.x 的 Windows 版本。

練習 1：安裝及設定 Web 開發環境

在本練習中，您將安裝 Tomcat 5.5 做 Web 容器，透過 Eclipse WST 的支援，開發人員可以在 Eclipse 中對 Web 應用程式進行線上除錯。

工作項目 1：安裝 Tomcat 5.5

1. 從 Apache 網站上下傳 Tomcat 5.5 的最新版本(檔名為 apache-tomcat-5.5.xx.exe，其中 xx 代表版本次編號)。
2. 執行 apache-tomcat-5.5.xx.exe，啟動安裝精靈。
3. 選擇同意版權宣告之後，會進入選擇安裝元件的畫面，請選擇「Minimum」。

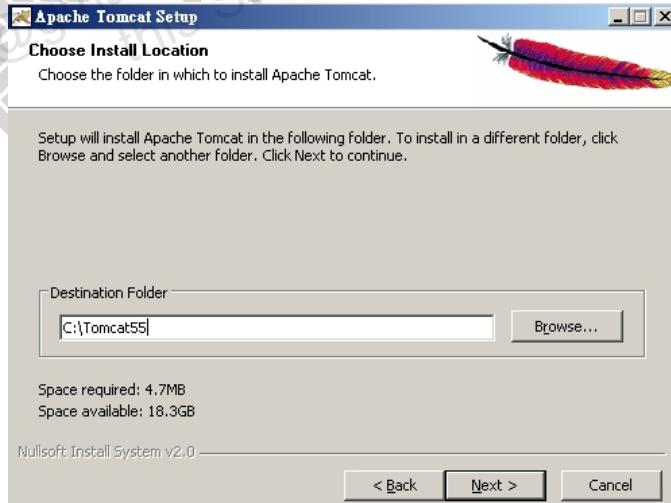


圖 6-1

4. 在安裝目錄的設定畫面，將安裝目錄設定為「C:\Tomcat55」(如圖 6-1)。
5. 接下來出現帳號密碼及連接埠畫面，我們在此帳號及連接埠採用預設的「admin」及「8080」。密碼則設定為「springframework」。
6. 接下來系統會詢問是否開啟 Tomcat，請選則「否」。由於我們會透過 Eclipse 來啟動 Tomcat，所以請選擇「否」，

練習 1：安裝及設定 Web 開發環境

接下來系統會出現安裝完成的提示訊息結束安裝。

工作項目 2：設定 Web 開發環境

1. 開啟 Eclipse。
2. 透過「Window」→「Preferences」打開 Preferences 視窗。
3. 在 Preferences 視窗選擇「Server」→「Installed Runtimes」。此時右方出現「Installed Server Runtime Environments」面板。
4. 點選「Add...」開啟 New Server Runtime 視窗。
5. 在伺服器列表中選取「Apache」→「Apache Tomcat v5.5」，點選「Next」。
6. 在 Tomcat Server 設定選單中，確認以下資訊後選取「Finish」：

Name: Apache Tomcat v5.5
Tomcat installation directory: C:\Tomcat55
JRE: Workbench default JRE
7. 按「OK」關閉 Preference 視窗。

工作項目 3：建立 Server 實體

在這個工作項目中您將依據前一項目所定義的 Server 設定建立一個 Server 實體。

1. 透過「Window」→「Show View...」→「Servers」打開 Servers View。
2. 在 Servers View 中以右鍵→「New」→「Server」打開「Define a New Server」視窗。
3. Server host name 選取「localhost」，server type 選取「Tomcat v5.5 Server」，Server runtime 選取「Apache Tomcat v5.5」，取「Next」。
4. 接下來進入 Web 專案的視窗，在這個視窗中可加入必須受到管理的 Web 專案，目前我們還沒開發任何 Web 專案，直接按「Next」跳過。
5. 選「Finish」關閉 New Server 視窗。
6. 這個時候在 Server View 中會出現 Tomcat v5.5 實例圖示，在上面按右鍵，選取 Start 來開啟 Server。

練習 1：安裝及設定 Web 開發環境

-
7. 如果設定正確，您應該可以在 Console View 中看到 Tomcat Server 已成功啟動的訊息。

工作項目 4：建立及測試一個簡單的 Web 應用程式

在這個工作項目中我們將建立一個最簡單的 Web 應用程式，並佈署在前一個工作項目所建立的 Server 實體上。

1. 在 Package View 上透過右鍵選取「New」→「Project...」→「Web / Dynamic Web Project」。
2. 根據下面的資訊，建立一個新的 Dynamic Web Project：

Project name : lab6-1

Project contents : C:\myspring_project\lab6-1

Target Runtime : Apache Tomcat v5.5

Configurations : <custom>

Project layout : Create separate source and output folders

Content Directory: WebContent

Java Source Directory: src

提示

在 Project Facets 對話窗中不需要修改任何選項，直接選擇「Next」即可。

3. 專案建置完成後，系統會詢問是否切換至「J2EE Perspective」，請選擇「Yes」，選擇完成後，您會發現在下方會開出包含「Servers」、「Database Explorer」等多個新的 Views，您可以視情況將不需要的先關起來，在需要時再透過「Window」→「Show View」打開。
4. 透過 Project Explorer 您應該可以看到類似圖 6-2 的幾個元素，其中 WebContent 目錄就是 Web 應用程式的主目錄。此外，透過點選「Deployment Descriptor: lab6-1」則可以打開 web.xml 檔案。

練習 1：安裝及設定 Web 開發環境



圖 6-2

5. 透過 Project Explorer 或 Package Explorer 將 WebContent 目錄展開，在根目錄按右鍵選擇「New」→「JSP」開啟 JavaServer Page 選單。
6. 「File name 請填入 index.jsp」，選「Next」。
7. 在 Select JSP Tempalte 選單中，請勾選「Use JSP Template」，並選取「New JSP File (html)」，按「Finish」之後，系統會自動產生一個新的 JSP 樣版，並命名為 index.jsp。
8. 打開 index.jsp 檔，在<body> ...</body>之間加入下列網頁程式碼：

```
<H1> <%="Hello World!"%> </H1>
```

9. 在 Project Explorer 中選取 index.jsp，按右鍵→「Run as...」→「Run on Server」，打開 Server 選單。
10. 在前一個工作項目中我們已經定義了一個新的 Server，所以請直接選取「Choose an existing server」，點選「Tomcat v5.5 Server @ localhost」，選「Next」。
11. 在 Add and Remove Projects 選單中，確認「lab6-1」專案已加入右方的「Configured projects」中，點選「Finish 完成設定」。
12. 如果設定正確，接下來系統會在 Console View 中顯示 Tomcat 的執行情況，並自動開啟一個 Browser 視窗，您可以從 Browser 中看到執行結果，或是另行開啟一個 Browser，輸入網址「<http://localhost:8080/lab6-1/index.jsp>」，將得到相同的結果。
13. 點選 Console View 視窗中的「Terminate」選項將 Tomcat 關閉(如圖 6-3)。

練習 2 : Spring Web MVC



```

Problems Properties Servers Database Explorer Susses Console 
Tomcat v5.5 Server _localhost [Apache Tomcat] C:\Program Files\java\j2se\1.5.0_11\bin\view.exe (2007/4/25 下午 8:56:22)
資訊: XML validation disabled
2007/4/25 下午 8:56:24 org.apache.coyote.http11.Http11BaseProtocol start
資訊: Starting Coyote HTTP/1.1 on http-8080
2007/4/25 下午 8:56:24 org.apache.common.ChannelSocket init
資訊: JK: ap11 listening on /0.0.0.0:8080
2007/4/25 下午 8:56:24 org.apache.jk.server.JkMain start
資訊: JK running ID=0 time=0/63 config=null
2007/4/25 下午 8:56:24 org.apache.catalina.storeconfig.StoreLoader load
資訊: Find registered Server-registry.xml at classpath resource
2007/4/25 下午 8:56:24 org.apache.catalina.startup.Catalina start
資訊: Server startup in 1092 ms

```

圖 6-3

練習 2 : Spring Web MVC

在這個練習中，您將透過 Spring Web MVC 的機制，建立一個簡單的登入用網頁，透過邏輯處理之後，依照是否通過認證顯示不同的網頁。練習 2 中的 Web 應用程式架構如圖 6-4 所示。

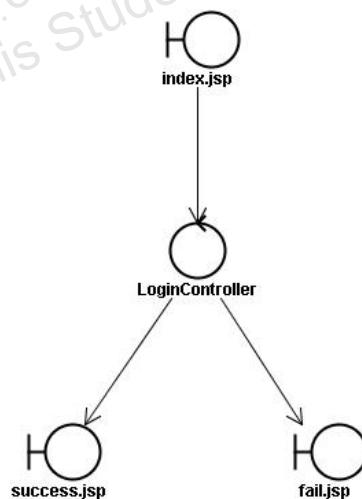


圖 6-4

工作項目 1：建立新的 Dynamic Web Project

- 參考練習 1，根據下列資訊建立一個新的 Dynamic Web Project :

Project name : lab6-2

Project contents : C:\myspring_project\lab6-2

練習 2 : Spring Web MVC

Target Runtime : Apache Tomcat v5.5**Configurations :** <custom>**Project layout :** Create separate source and output folders**Content Directory:** WebContent**Java Source Directory:** src

2. 將 spring.jar 拷貝到 lab6-2\WebContent\WEB-INF\lib 下。

提示

spring.jar 可在 spring framework 安裝中的 dist 目錄下找到。

工作項目 2：編寫網頁與程式碼

1. 在 Java Resources :src 圖示下，建立一個新的類別，如
程式碼 6-1 所示：

程式碼 6-1

```

1 package lab62;
2
3 import javax.servlet.http.HttpServletRequest;
4 import javax.servlet.http.HttpServletResponse;
5 import org.springframework.web.servlet.ModelAndView;
6 import org.springframework.web.servlet.mvc
7     .Controller;
8
9 public class LoginController implements Controller {
10    private String successView;
11    private String failView;
12    public ModelAndView handleRequest(
13        HttpServletRequest request,
14        HttpServletResponse response)
15    throws Exception {
16        String user = request.getParameter("user");
17        String passwd =request.getParameter("passwd");
18        if ("user".equals(user) &&
19            "123".equals(passwd)) {
20            return new ModelAndView(getSuccessView()
21                , "user", user);

```

練習 2 : Spring Web MVC

```

22         } else {
23             return new ModelAndView(getFailView());
24         }
25     }
26
27     public void setSuccessView(String view) {
28         this.successView = view;
29     }
30
31     public String getSuccessView() {
32         return successView;
33     }
34
35     public void setFailView(String view) {
36         this.failView = view;
37     }
38
39     public String getFailView() {
40         return failView;
41     }
42 }

```

2. 參考練習一，建立一個新的登入用網頁(index.jsp)，並在<body>...</body>中間加入下列程式碼：

```

<form action="login.do" method="POST">
    username:<input type="text" name="user" /> <br/>
    password:<input type="text" name="passwd" /> <br/>
    <input type="submit" />
</form>

```

3. 在WEB-INF下建立一個新目錄，名稱為「pages」。
 4. 在WEB-INF/pages下建立一個名稱為success.jsp的新網頁，並在<body>...</body>中間加入下列程式碼：

```
<H2>Login Success !</H2>
```

5. 在WEB-INF/pages下建立一個名稱為fail.jsp的新網頁，

練習 2 : Spring Web MVC

並在<body>...</body>中間加入下列程式碼：

```
<H2>Login Fail !</H2>
```

工作項目 3：編寫設定檔

- 開啟 lab6-2 下 WebContent/WEB-INF/web.xml，在<welcome-file-list>元素之下加入下列設定：

```
<servlet>
  <servlet-name>dispatcherServlet</servlet-name>
  <servlet-class>
    org.springframework.web.servlet
      .DispatcherServlet
  </servlet-class>
  <init-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>
      /WEB-INF/applicationContext.xml
    </param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
  <servlet-name>dispatcherServlet</servlet-name>
  <url-pattern>*.do</url-pattern>
</servlet-mapping>
```

- 在 WEB-INF 下建立一個新的 applicationContext.xml 檔案，內容如程式碼 6-2。

程式碼 6-2

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <beans
3    xmlns="http://www.springframework.org/schema/beans"
4    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

練習 2 : Spring Web MVC

```

5      xsi:schemaLocation=
6          "http://www.springframework.org/schema/beans
7              http://www.springframework.org/schema/beans
8                  /spring-beans-2.0.xsd">
9      <bean class="org.springframework.web.servlet
10         .handler.SimpleUrlHandlerMapping">
11         <property name="mappings">
12             <props>
13                 <prop key="/login.do">
14                     loginController
15                 </prop>
16             </props>
17         </property>
18     </bean>
19
20     <bean class="org.springframework.web.servlet
21         .view.InternalResourceViewResolver">
22         <property name="prefix">
23             <value>/WEB-INF/pages/</value>
24         </property>
25         <property name="suffix">
26             <value>.jsp</value>
27         </property>
28     </bean>
29
30     <bean id="loginController"
31         class="lab62.LoginController">
32         <property name="successView">
33             <value>success</value>
34         </property>
35         <property name="failView">
36             <value>fail</value>
37         </property>
38     </bean>
39 </beans>

```

3. 在 Project Explorer 中選取 index.jsp，按右鍵 → 「Run as...」

練習 3 : ActionSupport 與 WebApplicationContext

- 「Run on Server」，打開 Server 選單。
4. 在前一個工作項目中我們已經定義了一個新的 Server，所以請直接選取「Choose an existing server」，點選「Tomcat v5.5 Server @ localhost」，選「Next」。
 5. 在 Add and Remove Projects 選單中，選擇「Remove All」移除其它專案，並將「lab6-2」專案加入右方的「Configured projects」中，點選「Finish 完成設定」。
 6. 如果設定正確，接下來系統會在 Console View 中顯示 Tomcat 的執行情況，並自動開啟一個 Browser 視窗，您可以從 Browser 中看到執行結果，或是另行開啟一個 Brwoser ，輸入網址「<http://localhost:8080/lab6-2/index.jsp>」，將得到相同的結果。
 7. 點選 Console View 視窗中的「Terminate」選項將 Tomcat 關閉。

練習 3 : ActionSupport 與 WebApplicationContext

在這個練習中您將使用 Spring 所提供的 ActionSupport 與 WebApplicationContext，實際練習 Spring 和 Struts 的整合方式。

工作項目 1：匯入專案及 web.xml 設定

1. 將 Lab6-3 的練習用專案目錄拷貝到 c:\spring-framework 底下，使得其路徑為 c:\spring-framework\lab6-3。
2. 將 Lab6-3 專案檔匯入：在 Package View 空白處按右鍵，選擇「Import...」，在 Import 視窗點選「General」→「Existing Projects into Workspace」，選擇「Next」。
3. 點選「Select root directory:」，按下右方「Browse...」鍵選取 c:\spring-framework\lab6-3，按「OK」。
4. 按「Finish」匯入專案。
5. 開啟 lab6-3 下 WebContent/WEB-INF/web.xml，在 <welcome-file-list> 元素之下加入 struts 框架的設定：

```
<servlet>
<servlet-name>struts</servlet-name>
```

練習 3 : ActionSupport 與 WebApplicationContext

```

<servlet-class>
    org.apache.struts.action.ActionServlet
</servlet-class>
<load-on-startup>2</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>struts</servlet-name>
    <url-pattern>*.do</url-pattern>
</servlet-mapping>

```

6. 接下來在同一個檔案中繼續加入 ContextLoaderListener 的設定:

```

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

```

工作項目 2：編寫服務程式碼

在這個工作項目中，我們首先將在建立負責商務邏輯的介面及其實作類別，另外也將實作一個 Struts 的 Action 類別，在這個類別中我們將使用 Spring 所提供的機制來取得 Spring 元件的實體。

- 首先在 Java Resource :src 中建立一個 lab63.LoginService 介面，如下所示：

```

package lab63;

public interface LoginService {
    public boolean login(String user, String password);
}

```

- 接下來建立 lab63/LoginService 介面的實作名稱為 LoginServiceImpl，實作在同一個目錄下：

練習 3 : ActionSupport 與 WebApplicationContext

```
package lab63;

public class LoginServiceImpl implements LoginService {
    public boolean login(String user, String password) {
        if ("user".equals(user) &&
            "123".equals(password)) {
            return true;
        } else {
            return false;
        }
    }
}
```

3. 打開在 lab63/LoginAction 類別，在 execute(...)方法中加入下列程式碼：

```
WebApplicationContext ctx = getWebApplicationContext();
LoginService loginService =
    (LoginService) ctx.getBean("loginService");
String user = request.getParameter("user");
String password = request.getParameter("passwd");

if (loginService.login(user, password))
    return mapping.findForward("success");
else
    return mapping.findForward("fail");
```



動動腦—這裏的 LoginAction 是繼承 ActionSupport 類別，想一想，為什麼在這裏我們要繼承這個類別？直接繼承 Struts 的 Action 類別會造成什麼問題？

工作項目 3：編寫設定檔

1. 編寫 struts-config.xml，在<struts-config>...</struts-config>之間加入 action-mapping 元素，並設定登入的處理邏輯為 lab63.LoginAction，成功及失敗的網頁分別為 success.jsp

練習 4：Spring 與 Struts 的非侵入性整合

及 fail.jsp:

```
<action-mappings>
    <action path="/login"
        type="lab63.LoginAction"
        scope="session">
        <forward name="success"
            path="/WEB-INF/pages/success.jsp" />
        <forward name="fail"
            path="/WEB-INF/pages/fail.jsp" />
    </action>
</action-mappings>
```

2. 根據下列規格，在 applicationContext.xml 中設定一個 bean 元件：

Bean id: loginService

Bean class: lab63.LoginServiceImpl

3. 在 Project Explorer 中選取 index.jsp，按右鍵 → 「Run as...」 → 「Run on Server」，打開 Server 選單。
4. 選取「Choose an existing server」，點選「Tomcat v5.5 Server @ localhost」，選「Next」。
5. 在 Add and Remove Projects 選單中，選擇「Remove All」移除其它專案，並將「lab6-3」專案加入右方的「Configured projects」中，點選「Finish 完成設定」。
6. 如果設定正確，接下來系統會在 Console View 中顯示 Tomcat 的執行情況，並自動開啟一個 Browser 視窗，您可以從 Browser 中看到執行結果，或是另行開啟一個 Brwoser ，輸入網址「<http://localhost:8080/lab6-3/index.jsp>」，將得到相同的結果。
7. 點選 Console View 視窗中的「Terminate」選項將 Tomcat 關閉。

練習 4：Spring 與 Struts 的非侵入性整合

這個練習的應用程式和上個練習在功能上完全相同，不過在這

練習 4 : Spring 與 Struts 的非侵入性整合

個練習中您將學習如何透過置換 Struts 的 Controller，讓 Struts 完全感覺不到 Spring 的存在，達成 Spring 與 Struts 間非侵入性的整合，此外，LoginService 也將透過您熟悉的依賴注入機制直接注入 LoginAction 中，不需要再透過 WebApplicationContext 取得。

工作項目 1：匯入專案及 web.xml 設定

1. 將 Lab6-4 的練習用專案目錄拷貝到 c:\spring-framework 底下，使得其路徑為 c:\spring-framework\lab6-4。
2. 將 Lab6-4 專案檔匯入：在 Package View 空白處按右鍵，選擇「Import…」，在 Import 視窗點選「General」「Existing Projects into Workspace」，選擇「Next」。
3. 點選「Select root directory:」，按下右方「Browse…」鍵選取 c:\spring-framework\lab6-4，按「OK」。
4. 按「Finish」匯入專案。

工作項目 2：編寫服務程式碼

在這個工作項目中，我們將在建立負責商務邏輯的介面及其實作類別，另外將實作一個 Struts 的 Action 類別，這個類別和上一個練習最大的不同在於 LoginAction 是直接繼承 Struts 的 Action 類別。

1. 依據下列規格，新增一個類別：

類別名稱: lab64.LoginAction
父類別: org.apache.struts.action.Action;

2. 在上述類別中加入下列程式碼：

```
private LoginService loginService;

public void setLoginService(LoginService loginService)
{
    this.loginService = loginService;
}
```

練習 4 : Spring 與 Struts 的非侵入性整合

```

public ActionForward execute(
        ActionMapping mapping, ActionForm form,
        HttpServletRequest request,
        HttpServletResponse response)
{
    String user = request.getParameter("user");
    String password = request.getParameter("passwd");

    if (loginService.login(user, password))
        return mapping.findForward("success");
    else
        return mapping.findForward("fail");
}

```

工作項目 3：編寫設定檔

1. 編寫 `struts-config.xml`，在 `<struts-config>...</struts-config>` 之間加入 `action-mapping` 元素，並設定登入的處理邏輯為 `lab64.LoginAction`，成功及失敗的網頁分別為 `success.jsp` 及 `fail.jsp`:

```

<action-mappings>
    <action path="/login"
            type="lab64.LoginAction"
            scope="session">
        <forward name="success"
                path="/WEB-INF/pages/success.jsp" />
        <forward name="fail"
                path="/WEB-INF/pages/fail.jsp" />
    </action>
</action-mappings>

```

2. 在 `<action-mapping>` 下，加入下列設定，以便置換 Struts 的 RequestProcessor，並加入 Spring 提供的 Struts ContextLoaderPlugin:

練習 4 : Spring 與 Struts 的非侵入性整合

```
<controller>
    <set-property property="processorClass"
        value="org.springframework.web.struts
        .DelegatingRequestProcessor" />
</controller>
<plug-in className="org.springframework.web.struts
    .ContextLoaderPlugIn">
    <set-property property="contextConfigLocation"
        value="/WEB-INF/struts-servlet.xml" />
</plug-in>
```

3. 根據下列規格，在 applicationContext.xml 中設定一個 bean 元件：

Bean id: loginService

Bean class: lab64.LoginServiceImpl

4. 根據下列程式碼，在 struts-servlet.xml 中設定一個 Struts Action 的 bean 元件，並將 loginService 注入：

```
<bean name="/login" class="lab64.LoginAction">
    <property name="loginService">
        <ref bean="loginService" />
    </property>
</bean>
```

5. 在 Project Explorer 中選取 index.jsp，按右鍵 → 「Run as...」 → 「Run on Server」，打開 Server 選單。
6. 選取「Choose an existing server」，點選「Tomcat v5.5 Server @ localhost」，選「Next」。
7. 在 Add and Remove Projects 選單中，選擇「Remove All」移除其它專案，並將「lab6-4」專案加入右方的「Configured projects」中，點選「Finish 完成設定」。
8. 如果設定正確，接下來系統會在 Console View 中顯示 Tomcat 的執行情況，並自動開啟一個 Browser 視窗，您可以從 Browser 中看到執行結果，或是另行開啟一個 Browser ，輸入網址「http://localhost:8080/lab6-4/index.jsp」，將得到相同的

練習 4 : Spring 與 Struts 的非侵入性整合

結果。

9. 點選 Console View 視窗中的「Terminate」選項將 Tomcat 關閉。



Tian Feng (tim9958@gmail.com) has a non-transferable license to
use this Student Guide.

實驗七

Spring 與 JMS

實驗目標

當完成本實驗後，您將能學習到：

- 基本的 ActiveMQ 設定及操作。
- 透過 JMX Console 管理及監看 ActiveMQ 的訊息。
- 設定 ActiveMQ 中 Administered Objects 的 JNDI 名稱。
- 透過 Spring 提供的 JmsTemplate 發送及傳送訊息。
- 實作 MDP 及 MDP 容器。

相關資料

準備工作



本實驗假設您已經完成以下事項：

- 下傳或拷貝 ActiveMQ 檔案(apache-activemq-4.1.1.zip)
並解壓縮至 C:\apache-activemq-4.1.1\。

練習 1：安裝及設定 ActiveMQ

工作項目 1：設定 JAVA_HOME 環境變數

1. 桌面上「我的電腦」圖示 → 「進階」→ 「環境變數」找到下方「系統變數」，按「新增」。
2. 變數名稱請設定 JAVA_HOME，變數值請設定您的 JDK 安裝路徑(例如 C:\Program Files\Java\jdk1.5.0_10)

工作項目 2：啟動 Active MQ Server

1. 使用檔案總管，將目錄切換至「C:\apache-activemq-4.1.1\bin」下。
2. 執行 activemq.bat，看到如下畫面就代表已經成功地啟動 Active MQ Server。

```

C:\WINDOWS\system32\cmd.exe
INFO ManagementContext - JMX consoles can connect to service:jmx:rmi://localhost/rmi://localhost:1099/jmrmxi
INFO JDBCPersistenceAdapter - Database driver recognized: [apache_derby_embedded_jdbc_driver]
INFO DefaultDatabaseLocker - Attempting to acquire the exclusive lock to become the Master broker
INFO DefaultDatabaseLocker - Becoming the master on dataSource: org.apache.derby.jdbc.EmbeddedDataSource@1ddda
INFO JournalPersistenceAdapter - Journal Recovery Started from: Active Journal: using 5 x 20.0 Megs at: C:\apache-activemq-4.1.1\activemq-data\journal
INFO JournalPersistenceAdapter - Journal Recovered: 0 message(s) in transactions recovered.
INFO TransportServerThreadSupport - Listening for connections at: tcp://hsb:61616
INFO TransportConnector - Connector openwire Started
INFO TransportServerThreadSupport - Listening for connections at: ssl://hsb:61617
INFO TransportConnector - Connector ssl Started
INFO TransportServerThreadSupport - Listening for connections at: stomp://hsb:61613
INFO TransportConnector - Connector stomp Started
INFO NetworkConnector - Network Connector default-nc Started
INFO BrokerService - ActiveMQ JMS Message Broker <localhost, ID:hsb-1174-1178240301578-1:0> started

```

圖 7-1

工作項目 3：使用 jconsole 監控及操作 ActiveMQ

1. 「開始」→ 「執行」→ 輸入「cmd」。
2. 使用 cd 指令將目錄切換至 JDK 安裝目錄下的 bin 目錄。(例如 cd c:\program files\java\jdk1.5.0_10\bin)

練習 1：安裝及設定 ActiveMQ

3. 執行 jconsole.exe 開啟 JMX console。
4. 在 Jconsole : Connect to Agent 視窗，切換到 **Advanced** Tab，在 JMX URL 中輸入「service:jmx:rmi://jndi/rmi://localhost:1099/jmxrmi」，按下「Connect」連接 ActiveMQ(如圖 7-2)。

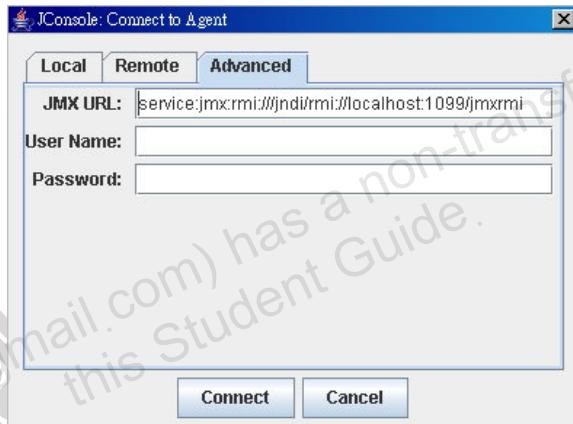


圖 7-2

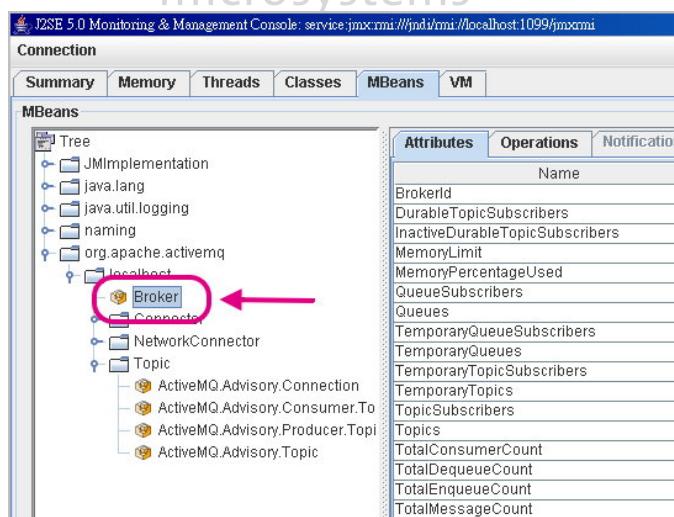


圖 7-3

5. 成功連接 ActiveMQ 之後，切換到 MBeans Tab，在左方樹狀圖中尋找「org.apache.activemq / localhost / Broker」，

練習 1：安裝及設定 ActiveMQ

雙擊「Broker」圖示(如圖 7-3)。

6. 點選 Broker 後，在右方出現 Broker 的相關資訊，點選「Operation」Tab，會出現 Broker 相關的指令。

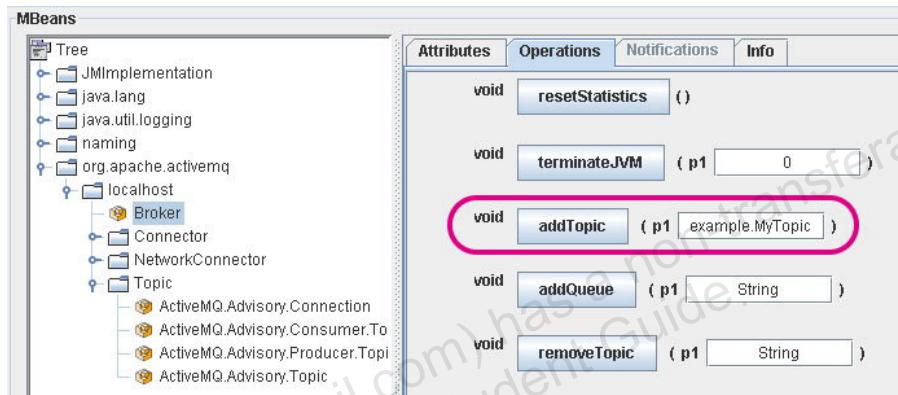


圖 7-4

7. 參考圖 7-4，找到「addTopic」指令鍵，在文字欄中填入「example.MyTopic」，按下「addTopic」指令鍵。此時會出現一個 Method successfully invoked 的指示視窗，按「確定」關閉。這時在視窗右方可看到剛才所新增的 example.MyTopic。
8. 在左方樹狀圖點選「org.apache.activemq / localhost / Topic / example.MyTopic」的圖示，右方會出現 example.MyTopic 的相關資訊。
9. 找到右方的「sendTextMessage」指令鍵，在文字欄位中輸入「Hello World!」後按下「sendTextMessage」指令鍵送出訊息。
10. 按下「browse」指令鍵，觀察所收到的訊息內容。

練習 2：發送 JMS 訊息

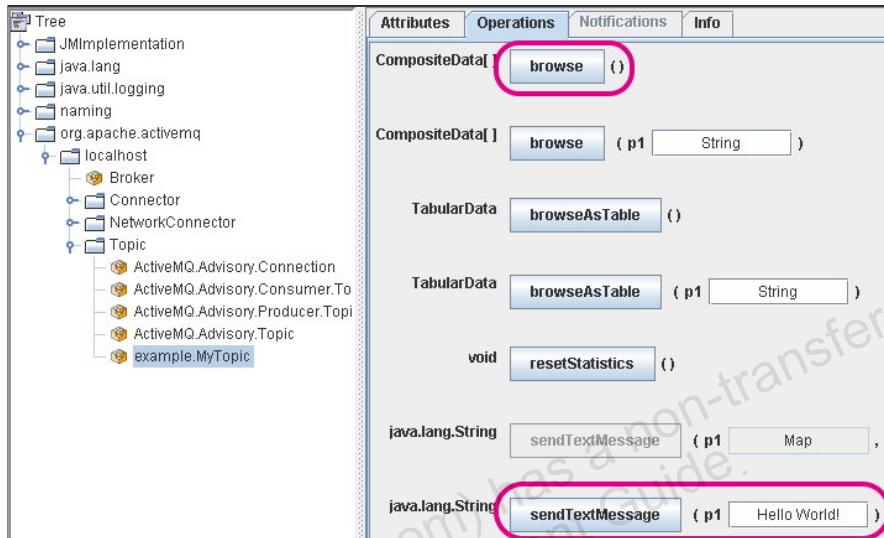


圖 7-5

練習 2：發送 JMS 訊息

工作項目 1：匯入練習專案

1. 將 Lab7 的練習用專案目錄拷貝到 c:\spring-framework 底下，使得其路徑為 c:\spring-framework\Lab7。
2. 將 Lab7 專案檔匯入：在 Package View 空白處按右鍵，選擇「Import...」，在 Import 視窗點選「General」→「Existing Projects into Workspace」，選擇「Next」。
3. 點選「Select root directory:」，按下右方「Browse...」鍵選取 c:\spring-framework\Lab7，按「OK」。
4. 按「Finish」匯入專案。

工作項目 2：建立 Administered Objects

在這個工作項目中，我們將建立 ConnectionFactory 及一些 Destinations，並設定其 JNDI 名稱，以利後續在程式中透過 JNDI API 來存取。

練習 2：發送 JMS 訊息

- 在 src 目錄下按下右鍵 → 「New」→ 「File」，建立一個空白的 jndi.properties 檔。
- 在 jndi.properties 中設定 Administered Object 的各項屬性：

```
java.naming.factory.initial = org.apache.activemq
.jndi.ActiveMQInitialContextFactory

# Configure the default connector
java.naming.provider.url = tcp://localhost:61616

# Setup a ConnectionFactory
connectionFactoryNames = jms/topicConnectionFactory

# register some topics in JNDI using the form
# topic.[jndiName] = [physicalName]
topic.jms/MyTopic = lab7.MyTopic
topic.jms/MyTopic2 = lab7.MyTopic2
```



重點提示—請留意 Topic/Queue 的設定語法為

Topic.[JNDI 名稱]=[在 ActiveMQ 中的名稱]**或**
Queue.[JNDI 名稱]=[在 ActiveMQ 中的名稱]

例如

topic.jms/MyTopic = lab7.MyTopic

代表 JNDI 名稱為 jms/MyTopic，ActiveMQ 名稱為 lab7.MyTopic，ActiveMQ 名稱就是在 JMXConsole 中可看到的名稱。

在此我們指定了 Active MQ 的 initial context factory 為 ActiveMQInitialContextFactory，且設定了與 Server 符合的 URL 及 TCP port。此外，也建立了一個 JNDI 名稱為「 jms/topicConnectionFactory 」的 TopicConnectionFactory，及二個 Topics，JNDI 名稱分別為「 jms/MyTopic 」及「 jms/MyTopic2 」。

練習 2：發送 JMS 訊息

工作項目 3：寫作一個簡單的訊息發送器

- 依照下列規格，建立一個新類別。

Package : lab7.ex2

Class Name : TestClient1

在 Which method stubs would you like to create 中勾選
public static void main(String[] args)自動產生 main()。

- 接著我們將在 main() 函式開始寫作程式碼，首先透過 InitialContext 取得一個 TopicConnectionFactory，請記得 JNDI 名稱要和在 jndi.properties 中的設定相同：

```
Context ctx = new InitialContext();
ConnectionFactory cfactory =
    (ConnectionFactory) ctx
        .lookup("jms/topicConnectionFactory");
```

- 接著我們要透過 context 取得二個剛才設定的 Topics：

```
Destination myTopic = (Destination)
    ctx.lookup("jms/MyTopic");
Destination myTopic2 = (Destination)
    ctx.lookup("jms/MyTopic2");
```

- 再來利用 Spring 提供的 Jmstemplate 傳送一些測試訊息：

```
JmsTemplate jmst = new JmsTemplate(cfactory);
jmst.convertAndSend(myTopic, "Hi!");
jmst.convertAndSend(myTopic2, "Hi2!");
```

- 在 Package View 的 Task1Client.java 上按右鍵，選取「Run as」→「Java Application」來執行程式並觀察輸出結果。
- 開啟練習一所介紹的 JMX Console，找到 lab7.MyTopic 及 lab7.MyTopic2，透過 browse 指令觀察訊息是否已確實收到。

工作項目 4：透過 Spring 依賴注入設定 JMS

在這個工作項目中，您將以 Spring 所提供的依賴注入機制來設定 JMS，這種做法可以大幅減少設定的程式碼。

- 打開 lab7/ex2/beans-config.xml 檔案，首先使用 Spring 提供的 JndiObjectFactoryBean 設定二個 Administered Objects：

```
<bean id="jmsFactory"
      class="org.springframework
            .jndi.JndiObjectFactoryBean">
    <property name="jndiName"
              value="jms/topicConnectionFactory" />
</bean>

<bean id="myTopic"
      class="org.springframework
            .jndi.JndiObjectFactoryBean">
    <property name="jndiName" value="jms/MyTopic" />
</bean>
```

- 接下來將上述二個 Administered Objects 注入 JmsTemplate 中

```
<bean id="jmst"
      class="org.springframework.jms.core
            .JmsTemplate">
    <property name="connectionFactory"
              ref="jmsFactory" />
    <property name="defaultDestination"
              ref="myTopic" />
</bean>
```

- 依照下列規格，建立一個新類別。

Package : lab7.ex2

練習 3：接收 JMS 訊息**Class Name :TestClient2**

在 Which method stubs would you like to create 中勾選
public static void main(String[] args)自動產生 main()。

4. 在 main()中寫作程式碼，透過 ApplicationContext 取得 jmsTemplate 的實體，並發送一個測試訊息。

提示

```
ApplicationContext context =
    new ClassPathXmlApplicationContext(
        "lab7/ex3/beans-config.xml");
JmsTemplate jmst = (JmsTemplate)
    context.getBean("jmst");
jmst.convertAndSend("Hello World!");
```

5. 選取「Run as」→「Java Application」來執行程式並觀察輸出結果。
6. 開啟練習一所介紹的 JMX Console，找到 lab7.MyTopic，透過 browse 指令觀察訊息是否已確實收到。

練習 3：接收 JMS 訊息**工作項目 1：寫作 MessageConsumer**

1. 依照下列規格，建立一個新類別。

Package : lab7.ex3**Class Name :MessageConsumer**

在 Which method stubs would you like to create 中勾選
public static void main(String[] args)自動產生 main()。

2. 在 main()中寫作程式碼，透過 ApplicationContext 取得 jmsTemplate 的實體，使用 receiveAndConvert 方法接收

練習 4 : Message-Driven POJO

一個測試訊息並將它列印至 Console。

提示

```
ApplicationContext context =
    new ClassPathXmlApplicationContext(
        "lab7/ex3/beans-config.xml");
JmsTemplate jmst = (JmsTemplate)
    context.getBean("jmst");
System.out.println(jmst.receiveAndConvert());
```

3. 選取「Run as」→「Java Application」來執行程式並觀察輸出結果。
4. 開啟 JMX Console，找到 lab7.MyTopic，透過 browse 指令觀察訊息是否已確實收到。

練習 4 : Message-Driven POJO**工作項目 1：寫作 Message-Driven POJO 元件**

1. 依照下列規格，建立一個新類別。

Package : lab7.ex4**Class Name :** MyMessageDrivenPojo**Implements :** javax.jms.MessageListener

2. 接下實作 MDP 的 onMessage 方法，為了測試方便，我們只要將所收到的訊息列印出來即可。

提示

```
System.out.println("Message Received: " + msg);
```

練習 4 : Message-Driven POJO

工作項目 2：寫作 MDP Container 元件

- 打開 lab7/ex4/beans-config.xml，加入下列設定工作項目 1 中所寫作的 MDP 元件注入 MDP 容器中，在此我們使用 Spring 所提供的 DefaultMessageListenerContainer 使為 MDP 元件容器。

```
<bean id="myMDP"
      class="lab7.ex4.MyMessageDrivenPojo" />

<bean id="mdpContainer"
      class="org.springframework.jms.listener
              .DefaultMessageListenerContainer">
    <property name="concurrentConsumers" value="1" />
    <property name="connectionFactory"
              ref="jmsFactory" />
    <property name="destination" ref="myTopic" />
    <property name="messageListener" ref="myMDP" />
</bean>
```

- 依照下列規格，建立一個新類別。

Package : lab7.ex4

Class Name : MdpContainer

在 Which method stubs would you like to create 中勾選 **public static void main(String[] args)** 自動產生 main()。

- 在 main() 中寫作程式碼，透過 ApplicationContext 取得 mdpContainer 的實體，使用 start 方法啟動 Container。
- 在 lab7/ex4/MessageProvider 類別圖示上選取「Run as」→ 「Java Application」來執行程式送出一測試訊息。
- 觀察 MdpContainer 是否已接收到訊息並將內容列印出來。

實驗八

Spring 與排程管理

實驗目標

當完成本實驗後，您將能學習到：

- 如何使用 JDK 內建的 TimerTask 排程方式。
- 透過 Spring 建構 POJO-based TimerTask。
- 透過 Spring 的整合機制，使用 Quartz 進行排程。
- 使用 CronTrigger 進行排程。
- 透過 Spring 建構 POJO-based 的 Quartz 排程。

相關資料

準備工作



本實驗不需要任何準備工作。

練習 1 : JDK TimerTask

工作項目 1：匯入練習專案

1. 將 Lab8 的練習用專案目錄拷貝到 c:\spring-framework 底下，使得其路徑為 c:\spring-framework\lab8。
2. 將 Lab8 專案檔匯入：在 Package View 空白處按右鍵，選擇「Import...」，在 Import 視窗點選「General」→「Existing Projects into Workspace」，選擇「Next」。
3. 點選「Select root directory:」，按下右方「Browse...」鍵選取 c:\spring-framework\lab8，按「OK」。
4. 按「Finish」匯入專案。

工作項目 2：建立一個簡單的 TimerTask

1. 根據下列規格，建立 HelloTimerTask 類別：

Package : lab8.ex1

類別名稱 : HelloTimerTask

Extends : TimerTask

類別方法：

方法名稱	參數	回傳值
run	無	無

2. 實作 run 方法，將目前的時間印出。

提示

```
public void run() {
    System.out.println("Hello World! at time:" +
        new Date());
}
```

3. 在 lab8/ex1/beans-config.xml 中依據下列資料登錄 HelloTimerTask 為 Bean 元件。

練習 1 : JDK TimerTask

Bean 元件 ID: helloTask**類別名稱 :** lab8.ex1.HelloTimerTask**提示**

```
<bean id="helloTask" class="lab8.ex1.HelloTimerTask" />
```

4. 在 beans-config.xml 中依下列資訊登錄一個 ScheduledTimerTask 元件，並將 helloTask 注入其 timerTask 屬性。

Bean 元件 ID: helloTask**Bean 類別 :** org.springframework.scheduling.timer
.ScheduledTimerTask**屬性 :**

delay=1000 (一開始 delay1 秒)
period=3000 (之後每 3 秒執行一次)
timerTask=helloTask

提示 - 參考其它 Bean 元件時記得使用 ref 屬性

```
<property name="timerTask" ref="helloTask" />
```

5. 在 beans-config.xml 中依下列資訊登錄一個 timer 元件，並將上面的 ScheduledTimerTask 注入其 scheduledTimerTasks 屬性。

Bean 元件 ID: timer**Bean 類別 :** org.springframework.scheduling
.timer.TimerFactoryBean**屬性 :**

scheduledTimerTasks = {scheduledTask}
(scheduledTimerTasks 屬性是 List 型別)

提示 - 請留意<list>的設定方法

```
<property name="scheduledTimerTasks">
  <list>
    <ref bean="scheduledTask" />
  </list>
</property>
```

練習 1 : JDK TimerTask

```
</property>
```

6. 在 lab8/ex1/TestClient 類別圖示上選取「Run as」→「Java Application」，並觀察執行結果。

工作項目 3 : POJO-based TimerTask

在這個工作項目中，首先會讓您實做一個 POJO-based 的 TimerTask，接著透過 Spring 提供的 MethodInvokingTimerTaskFactoryBean 將該 POJO 注入原來的 ScheduledTimerTask 中。

1. 根據下列規格，建立 HelloPojoTask 類別：

Package : lab8.ex1

類別名稱 : HelloPojoTask

類別方法 :

方法名稱	參數	回傳值
hello	無	無

Hello 方法的內容 :

```
public void hello() {
    System.out.println("Hello World from POJO!
        at time:" + new Date());
}
```

2. 在 beans-config.xml 中，依下列資訊登錄一個 helloPojo 元件。

Bean 元件 ID: helloPojo

Bean 類別 : org.springframework.scheduling.timer
.MethodInvokingTimerTaskFactoryBean

屬性 :

targetObject= lab8.ex1.HelloPojoTask

targetMethod= hello

3. 修改 lab8/ex1/beans-config.xml 的內容，將 Bean ID 為 Spring 2.0 技術開發

練習 2 : Spring 與 Quartz 的整合運用

scheduledTask 的元件的 timerTask 屬性值由「helloTask」改為「**helloPojo**」。

- 在 lab8/ex1/TestClient 類別圖示上選取「Run as」→「Java Application」，並觀察執行結果。

練習 2 : Spring 與 Quartz 的整合運用

在這個工作項目中，我們將透過 Spring 所提供的整合機制來操作 Quartz Scheduling Framework。整個建構過程如圖 8-1 所示，首先我們會先建構 MyQuartzJobBean，這個類別的功能是告訴 Quartz Framework 當時間到要執行的指令內容，而 MyMessage 則是模擬 MyQuartzJobBean 在執行時會參考到的資訊，這二個類別會透過依賴注入進入 JobDetail 中。之後我們會依序將 JobDetail 注入 Trigger，將 Trigger 注入 Scheduler，透過啟動 Scheduler，就可以開始執行整個計時的處理工作。

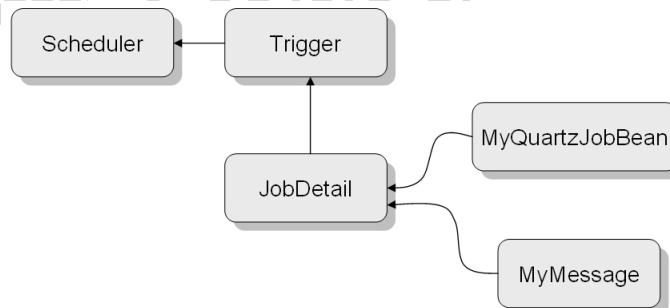


圖 8-1

工作項目 1：建立一個簡單的 Quartz 應用程式

- 根據下列規格，建立一個新的類別：

Package : lab8.ex2

類別名稱 : MyMessage

類別屬性 :

屬性名稱	型別	保護等級
------	----	------

練習 2 : Spring 與 Quartz 的整合運用

content	String	private
---------	--------	---------

類別方法 (content 屬性的 getter 及 setter)

方法名稱	參數	回傳值
setContent	String content	無
getContent	無	String

2. 根據下列規格，建立一個新的類別：

Package : lab8.ex2

類別名稱：MyQuartzJobBean

Extends : org.springframework.scheduling
.quartz.QuartzJobBean

類別屬性：

屬性名稱	型別	保護等級
myValue	int	private
myMessage	MyMessage	private

類別方法 (含屬性的 getter 及 setter)

方法名稱	參數	回傳值
setMyValue	String content	無
getMyValue	無	int
setMyMessage	Int value	無
getMyMessage	無	MyMessage
executeInternal	JobExecutionContext ctx	無 丟出 JobExecutionException

3. 實作 myValue 及 myMessage 的 getter 及 setter。
 4. 實作 executeInternal 方法，印出 MyMessage 的 content 屬性及 myValue 的內容。

提示

練習 2 : Spring 與 Quartz 的整合運用

```

public class MyQuartzJobBean extends QuartzJobBean
{
    private MyMessage message;
    private int myValue;

    ... (二個屬性的 getter 及 setter 方法)

    protected void executeInternal(
        JobExecutionContext ctx)
        throws JobExecutionException
    {
        System.out.println(message.getContent() + ","
            + myValue);
    }
}

```

-
5. 在 beans-config.xml 中，依下列資訊登錄一個 MyMessage 元件。

Bean 元件 ID: myMessage
Bean 類別 : lab8.ex2.MyMessage

屬性：
content= Hello Quartz!

6. 在 beans-config.xml 中，依下列資訊登錄一個 jobDetail 元件。

Bean 元件 ID: jobDetail
Bean 類別 : org.springframework.scheduling.quartz.JobDetailBean

屬性：
jobClass= lab8.ex2.MyQuartzJobBean
jobDataAsMap={key=myMessage,value=myMessage},
{key=myValue,value=3 }

提示 - 請留意 map 的寫法

```
<bean name="jobDetail" class="...">
    <property name="jobClass" value="..." />
    <property name="jobDataAsMap">
        <map>
            <entry key="myMessage"
                value-ref="myMessage" />
            <entry key="myValue" value="3" />
        </map>
    </property>
</bean>
```

7. 在 beans-config.xml 中，依下列資訊登錄一個 simpleTrigger 元件。

Bean 元件 ID: simpleTrigger

Bean 類別 : org.springframework.scheduling
.quartz.SimpleTriggerBean

屬性：

jobDetail=jobDetail
startDelay=1000
repeatInterval=3000

提示 - 請留意 map 的寫法

```
<bean id="simpleTrigger"
    class="org.springframework.scheduling
    .quartz.SimpleTriggerBean">
    <property name="jobDetail" ref="jobDetail" />
    <property name="startDelay" value="1000" />
    <property name="repeatInterval" value="3000" />
</bean>
```

練習 2 : Spring 與 Quartz 的整合運用

-
8. 在 beans-config.xml 中，依下列資訊登錄一個 scheduler 元件。

Bean 元件 ID: scheduler

Bean 類別 : org.springframework.scheduling.quartz
.SchedulerFactoryBean

屬性：

triggers(List)= {simpleTrigger}

提示 - 請您留意 list 的寫法

```
<bean id="scheduler"
      class="org.springframework.scheduling
      .quartz.SchedulerFactoryBean">
    <property name="triggers">
      <list>
        <ref bean="simpleTrigger" />
      </list>
    </property>
</bean>
```

9. 在 lab8/ex2/TestClient 類別圖示上選取「Run as」→「Java Application」，並觀察執行結果。

工作項目 2 : CronTrigger

1. 在 beans-config.xml 中，依下列資訊登錄一個 cronTrigger 元件。

Bean 元件 ID: cronTrigger

Bean 類別 : org.springframework.scheduling.

練習 2 : Spring 與 Quartz 的整合運用

quartz.CronTriggerBean

屬性 :

```
jobDetail=jobDetail
cronExpression=0/3 * * * *
```



動動腦 – 查閱 Quartz 的 javaDoc，確定您了解這一段 cronExpression 的意義。

2. 修改 scheduler 的設定，simpleTrigger 置換為 cronTrigger。
3. 在 lab8/ex2/TestClient 類別圖示上選取「Run as」→「Java Application」，並觀察執行結果。

工作項目 3：使用 POJO 定義 QuartzJob

1. 在 beans-config.xml 中，依下列資訊登錄一個 pojoJobDetail 元件。

Bean 元件 ID: pojoJobDetail

Bean 類別 : org.springframework.scheduling.quartz
.MethodInvokingJobDetailFactoryBean

屬性 :

```
targetObject=lab8.ex2.HelloPojoTask
targetMethod=hello
```

提示 -

```
<bean id="pojoJobDetail" class="...>
  <property name="targetObject">
    <bean class="lab8.ex2.HelloPojoTask" />
  </property>
  <property name="targetMethod" value="hello" />
</bean>
```

練習 2 : Spring 與 Quartz 的整合運用

2. 修改 cronTrigger 的設定， 將 jobDetail 置換為 pojoJobDetail。
3. 在 lab8/ex2/TestClient 類別圖示上選取「Run as」→「Java Application」，並觀察執行結果。

實驗九

Spring Framework 整合應用

實驗目標

當完成本實驗後，您將能學習到：

- 安裝與佈署範例應用程式。
- 透過 Spring 整合資料存取層。
- 透過 Spring 整合商務邏輯層及展現層。

我們在這個實驗單元中帶領您佈署及安裝 JPetStore 範例應用程式，接下來由最底層的資料存取開始，練習如何透過 Spring 將 Web 的三層式架構結合在一起。建議您在操作本實驗時，可與教材單元 9 交互對照參考，隨時了解目前的整合對象及方法。

相關資料

準備工作



- 本實驗假設您已經完成下列事項:
- 1. 已經完成實驗一，並已加裝 JST(J2EE Standard Tools)。
- 2. 已經完成實驗三，並已安裝、設定 MySQL4.1.22 及 MySQL Control Center。
- 3. 已經完成實驗六，並已依指示安裝並設定好 Tomcat 5.5 與 Eclipse 的整合。

若您未完成上述實驗，請您先參考這些實驗的操作步驟完成相關安裝。



練習 1：安裝及佈署 JPetStore 應用程式

工作項目 1：匯入練習用資料檔

1. 從 Windows 的「開始」→「執行」，打開執行視窗，輸入「cmd」開啟命令列視窗。
2. 將目錄切換至「C:\MySQL」。 (使用 CD 指令)
3. 拷貝專案目錄下的「/database/jpetstore.sql」到「C:\MySQL\bin 中」。
4. 在命令列視窗鍵入指令「mysql < jpetstore.sql -u root -p」。
5. 接下來系統會詢問密碼，請輸入「springframework」。
6. 在命令列視窗鍵入指令「mysql-u root -p」，並輸入密碼，登入 MySQL 的命令列控制介面。
7. 鍵入「use jpetstore;」。 (記得最後面要加分號)
8. 鍵入「show tables;」，觀察輸出結果。
9. 鍵入「select * from product;」，觀察輸出結果。

工作項目 2：匯入練習專案

1. 將 Lab9-1 的練習用專案目錄拷貝到 c:\spring-framework 底下，使得其路徑為 c:\spring-framework\lab9-1。
2. 將 Lab9-1 專案檔匯入：在 Package View 空白處按右鍵，選擇「Import...」，在 Import 視窗點選「General」→「Existing Projects into Workspace」，選擇「Next」。
3. 點選「Select root directory:」，按下右方「Browse...」鍵選取 c:\spring-framework\Lab9-1，按「OK」。
4. 按「Finish」匯入專案。

工作項目 3：測試 Web 應用程式

1. 如果您尚未在 Eclipse 環境中定義 Tomcat 5.5 Server 實體，請您參考實驗 6 的練習一先行設定完成。若您已經完成，請在 Eclipse 視窗下方找到 Server View，在 Tomcat 5.5 的圖示上按右鍵，在 Add and Remove Projects 選單中，確認「lab9-1」專案已加入右方的「Configured projects」

練習 2：資料存取層的建構及設定

中，點選「Finish 完成設定」。

2. 如果設定正確，接下來在 Tomcat 5.5 的圖示上按右鍵，選擇「Start」開始執行系統，系統會在 Console View 中顯示 Tomcat 的執行情況，此時請您自行開啟一個 Brwoser，輸入網址「<http://localhost:8080/lab9-1/>」，觀察 JPetStore 歡迎畫面是否已正確顯示(如圖 9-1)。

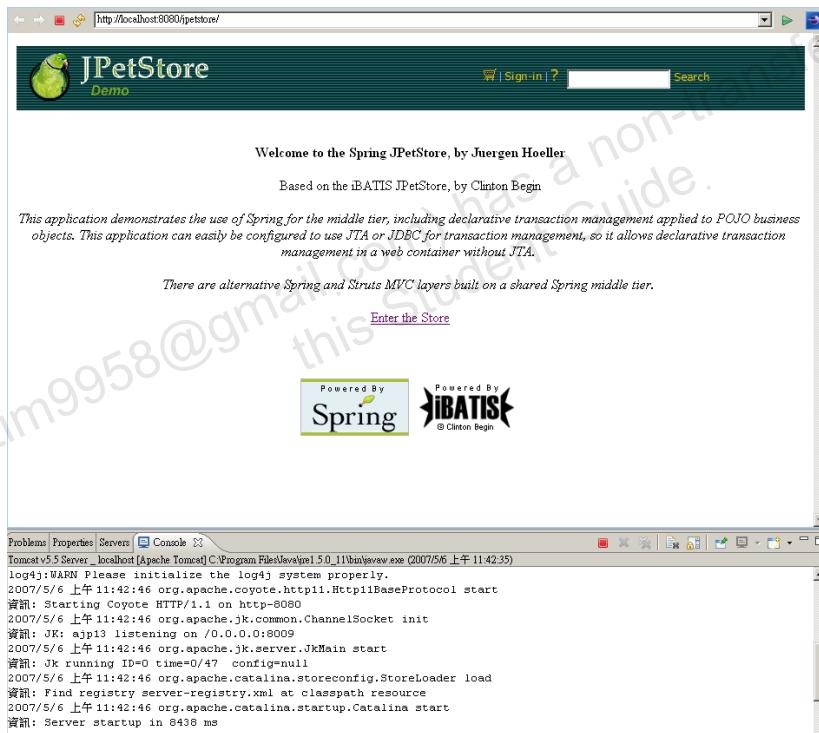


圖 9-1

3. 參閱第 9 單元的教材中的圖 9-4，實際瀏覽一次各個應用程式的執行流程及次序。

練習 2：資料存取層的建構及設定**工作項目 1：匯入練習專案**

練習 2：資料存取層的建構及設定

-
1. 將 Lab9-2 的練習用專案目錄拷貝到 c:\spring-framework 底下，使得其路徑為 c:\spring-framework\lab9-2。
 2. 將 Lab9-2 專案檔匯入：在 Package View 空白處按右鍵，選擇「Import...」，在 Import 視窗點選「General」→「Existing Projects into Workspace」，選擇「Next」。
 3. 點選「Select root directory:」，按下右方「Browse...」鍵選取 c:\spring-framework\Lab9-2，按「OK」。
 4. 按「Finish」匯入專案。

工作項目 2：以 JPA Annotations 設定 ORM

1. 從 Windows 的「開始」→「執行」，打開執行視窗，輸入「cmd」開啟命令列視窗。
2. 將目錄切換至「C:\MySQL」。 (使用 CD 指令)
3. 在命令列視窗鍵入指令「mysql -u root -p」，並輸入密碼，登入 MySQL 的命令列控制介面。
4. 鍵入「use jpetstore;」。 (記得最後面要加分號)
5. 鍵入「describe product;」，觀察 product 這個 table 的 schema。
6. 在 Eclipse 中，打開 jpetstore.domain.Product 類別，使用 JPA Annotations，並參考剛才看到的 product table schema，設定它們之間的對映。

提示

```

@Entity
@Table(name = "product")
public class Product {
    @Id
    @Column(name = "productid")
    private String productId;

    @Column(name = "category")
    private String categoryId;

    @Column(name = "name")
    private String name;

```

練習 2：資料存取層的建構及設定

```

    @Column(name = "descn")
    private String description;
    ...
}

```

7. 重覆上述 4-6，為 Category 類別加上 JPA Annotation。

工作項目 3：資料存取層的 Bean 設定

- 在 WebContent/WEB-INF/ 建立一個 dataAccessContext.xml 的 Bean 設定檔案，在 namespace 宣告中引入 tx 命名空間。

提示

```

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/
    schema/beans"
    xmlns:xsi="http://www.w3.org/2001/
        XMLSchema-instance"
    xmlns:tx="http://www.springframework.org/
        schema/tx"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        (空格)
        http://www.springframework.org/schema/beans
        /spring-beans-2.0.xsd
        http://www.springframework.org/schema/tx
        (空格)
        http://www.springframework.org/schema/tx
        /spring-tx-2.0.xsd">
    ...
    </beans>

```

- 在 Bean 設定檔中設定一個名為「dataSource」的 Bean，設定如下：

Bean id:dataSource

練習 2：資料存取層的建構及設定

Bean class: org.apache.commons.dbcp.

BasicDataSource

Bean class 的 destroy-method: close

(下面為 dataSource 的屬性)

driverClassName: com.mysql.jdbc.Driver**url:** jdbc:mysql://localhost:3306/springdb**username:** root**password:** springframework

提示

```
<bean id="..." class="..." destroy-method="...">
    <property name="..." value="..." />
    ... (其它 properties) ...
</bean>
```

3. 接著加入一個名為「sessionFactory」的 Bean，設定如下：

Bean id: sessionFactory**Bean class:** org.springframework.orm.hibernate3

.annotation.AnnotationSessionFactoryBean

Bean class 的 destroy-method: close

(下面為屬性設定)

dataSource: dataSource (ref)**annotatedClasses (List):**

(加入所有在 jpetstore.domain 中的類別名稱)

hibernateProperties:

(1) hibernate.dialect=org.hibernate.dialect.MySQLDialect

(2) hibernate.show_sql=true

提示

```
<bean id="sessionFactory"
    class="..." destroy-method="...">
    <property name="dataSource" ref="..." />
    <property name="annotatedClasses">
        <list>
            <value>...</value>
        </list>
    </property>
```

練習 3：商務邏輯層及展現層的設定

```
<property name="hibernateProperties">
    ...
</property>
</bean>
```

4. 接著宣告一個名稱為 productDao 的 Bean，其相關資訊如下：

Bean id: productDao

Bean class: jpetstore.dao.hibernate

.ProductDaoHibernateImpl

屬性: sessionFactory=sessionFactory(透過 ref 指向上一步所設定的 sessionFactory)

提示

```
<bean id="productDao" class="...">
    <property name="sessionFactory"
        ref="sessionFactory" />
</bean>
```

5. 依據相同的方式，設定 accountDao、itemDao、categoryDao 及 orderDao。

6. 接下來加入下列交易設定，並注入 sessionFactory:

```
<tx:annotation-driven
    transaction-manager="txManager" />
```

```
<bean id="txManager"
    class="org.springframework.orm.hibernate3
        .HibernateTransactionManager">
    <property name="sessionFactory"
        ref="sessionFactory" />
</bean>
```

練習 3：商務邏輯層及展現層的設定**工作項目 1：商務邏輯層的 Bean 設定**

練習 3：商務邏輯層及展現層的設定

1. 在 WebContent/WEB-INF/ 建立一個 applicationContext.xml 的 Bean 設定檔案。
2. 接著宣告一個名稱為 petStore 的 Bean，其相關資訊如下：

Bean id: petStore**Bean class:** jpetstore.domain.logic.PetStoreImpl**屬性:**

accountDao=accountDao
 categoryDao=categoryDao
 productDao=productDao
 itemDao=itemDao
 orderDao=orderDao

提示

```
<bean id="petStore" class="...">
  <property name="accountDao">
    <ref bean="accountDao" />
  </property>
  <property name="categoryDao">
    <ref bean="categoryDao" />
  </property>
  ...
</bean>
```

3. 開啟 jpetstore.domain.logic.PetStoreImpl 類別，在 insertOrder 方法上面加入交易屬性的宣告：

```
@Transactional(readOnly = false,
               propagation = Propagation.REQUIRED,
               isolation = Isolation.DEFAULT)
public void insertOrder(Order order)
{
    this.orderDao.insertOrder(order);
    this.itemDao.updateQuantity(order);
}
```

工作項目 2：展現層的設定

練習 3：商務邏輯層及展現層的設定

1. 開啟 WebContent/WEB-INF/web.xml 檔案。
2. 加入 Spring Bean 設定檔的路徑設定：透過 <context-param>加入三個 Spring 設定檔的路徑資訊，參數名稱為「contextConfigLocation」。

提示

```
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>
        /WEB-INF/struts-servlet.xml
        /WEB-INF/dataAccessContext.xml
        /WEB-INF/applicationContext.xml
    </param-value>
</context-param>
```

3. 在 web.xml 中加入 Spring 提供的 ContextLoaderListener。

提示

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

工作項目 3：測試 Web 應用程式

1. 在 Eclipse 視窗下方找到 Server View，在 Tomcat 5.5 的圖示上按右鍵，在 Add and Remove Projects 選單中，確認「lab9-1」專案已加入右方的「Configured projects」中，點選「Finish 完成設定」。
2. 如果設定正確，接下來在 Tomcat 5.5 的圖示上按右鍵，選擇「Start」開始執行系統，系統會在 Console View 中顯示

練習 3：商務邏輯層及展現層的設定

Tomcat 的執行情況，此時請您自行開啟一個 Brwoser，輸入網址「<http://localhost:8080/lab9-2/>」，觀察 JPetStore 歡迎畫面是否已正確顯示。

3. 參閱第 9 單元的教材中的圖 9-4，實際測試一次各個應用程式的執行流程及次序。