序

誠摯感謝 前金管會主委暨現任立法委員 曾銘宗委員、宏達電 謝昱帆副總經理、華碩電腦 郭海威處長、中華兩岸頤養促進會 詹清池理事長等對本書的肯定與推薦！隔了兩年多，繼「Java SE7/8 OCAJP 專業認證指南：擬真試題實戰」、「Java SE7/8 OCPJP 進階認證指南：擬真試題實戰」兩書後，本書「Java RWD Web 企業網站開發指南：使用 Spring MVC 與 Bootstrap」終於完成了！在此同時，前述兩本書也改版為「Java SE8 OCAJP 專業認證指南」與「Java SE8 OCPJP 進階認證指南」。

撰寫本書的過程中，除了持續在公司開發系統之外，也藉由勞動力發展署主持的「產業人才投資計畫」不間斷推展教學工作。在與同事及學員互動時，我經常思考什麼樣的 Java 書籍才是大家需要的？我嘗試把教學心得和開發經驗結合成本書，希望可以協助具備 Java SE 基礎，為準備進入 Java 企業網站開發領域的朋友們提供一個完整管道。

書中附有大量且完整的程式碼範例，以 Eclipse 匯入專案後調整設定即可執行，部分專案使用資料庫為 Derby 與 HSQL，定位為常備的工具書籍。

要開發企業級網站並不容易，需要具備的知識與技術很多；近年來因為 RWD(響應式網頁設計) 興起，也要考慮在行動裝置上呈現的樣貌，因此更需要一些開放原始碼 (open source) 的協助，即便在使用者體驗 (user experience) 方面。本書共分 4 個部分，分別是：

Part 1：前端網頁應用程式。包含 HTML、CSS、JavaScript、JQuery 與 Bootstrap，本書著重點不在美編設計或是前端程式設計，而是在運用已知的元件和函式庫。

Part 2：Sevlet、JSP 與網站伺服器。了解 Java 在網站容器裡的運行方式與觀念，有助於解決一些底層問題。

Part 3：Java 網站框架。架構企業網站如同興建大型建築，細節、效率、安全性、成本等都要考量，使用框架可以達成這些需求。

Part 4：綜合實作。結合前述 3 個部分的網站開發基礎，使用 Bootstrap 模板建構現代化 Java 企業網站。

限於篇幅，書中涉及 Java SE 的主題將請讀者參閱「Java SE8 OCPJP 進階認證指南」一書的指定篇章。

書中部分範例為求編排美觀，會將跨行的程式碼置右；惟實際範例程式碼無此考量，皆為正常編排且可執行。

這本書的出版，受到了很多親朋好友、公司同事、讀者與 Java 技術與認證交流平台 (https://www.facebook.com/groups/748837991920629/) 版友的鼓勵與督促，再次感謝各位。

18

Spring 框架導論

章節重點

18.1 Spring 的目的與策略
18.2 Spring 開發手法 DI 介紹
18.3 Spring 開發手法 AOP 介紹
18.4 Spring 開發手法 Template 介紹
18.5 Spring 的架構
18.6 Spring 的模組與商品組合

18.1 Spring 的目的與策略

18.1.1 Spring 簡介

Java 自 1995 年誕生以來，歷經 20 多個年頭的演變，逐漸分拆出 Java SE、Java EE、Java ME 三大版本，也擁有許多眾所皆知的功能模組，如 Applets(已不建議使用)、Enterprise JavaBeans(EJB)、Servlet 等。除此之外受惠於「開放原始碼 (open source)」的本質，有著眾多「社群 (community)」的支持，因此有著今日繁榮的生態圈與多元應用。「Spring」社群在這裡就扮演著重要角色。

在一開始的時候，針對 Java EE 裡「笨重」的 EJB 架構，Spring 使用了在 EJB 或企業板的 Java 規格裡看不到的「plain old Java objects (POJOs)」，提供了一個「輕量」級的解決方案，因使開始吸引了眾多開發者的目光。

逐漸，EJB 和 Java EE 也開始改革自己的架構，推出了類似 Spring 使用的 POJO 元件，並加入了「Dependency Injection (DI)」和「Aspect-Oriented Programming (AOP)」，其實也都源自 Spring 的啟發。

碁峯 www.gotop.com.tw

雖然 Java EE 已經逐漸在相關領域追上 Spring，但依然不及 Spring 在其他領域的進展步調。諸如：

- 手機程式開發 (Mobile Development)
- 社群 API 的整合 (Social API Integration)
- NoSQL 資料庫
- 雲端運算 (Cloud Computing)
- 大數據 (Big Data)

等都是 Spring 近年來持續耕耘的領域，早已領先 Java EE 規格更多，因此是帶動 Java 發展的領頭羊之一。

18.1.2 Spring 的目的與簡化開發的策略

Spring 由 Rod Johnson 在他的 2002 年著作「Expert One-on-One: J2EE Design and Development」中首次描述。Spring 框架的創立目的在「簡化複雜 Java 程式開發 (Spring simplifies Java development)」。Spring 的首戰瞄準當時常常令人卻步的複雜企業版程式開發，並使用簡單的 POJO 元件達成過往只有使用 EJB 架構才能達到的功能。除了簡化伺服器端的程式開發外，使用 Spring 框架也可以提升程式的「可測試性 (testability)」和「鬆耦合度 (loose coupling)」。

大部分的 Java 框架都是針對某一個主題宣稱可以簡化和快速開發，如網頁的 MVC、排程等等，只有 Spring 宣稱可以簡化 Java 程式的開發「Spring simplifies Java development」。 Spring 使用 4 個關鍵策略來達到這個目的：

1. 藉由使用 POJO 物件達到輕量化 (lightweight) 且最小侵入式 (minimally invasive) 開發。
2. 藉由「依賴注入 (dependency injection)」和 interface 導向的開發方式達到鬆耦合 (loose coupling) 的目的。
3. 使用「面向 (aspect)」的概念或物件達成宣告式開發目地。
4. 使用「面向 (aspect)」、「模板 (template)」的概念或物件讓一些照本宣科的程式碼 (boilerplate code) 可以重複使用。

18.1.3 Spring 使用 POJO 元件

POJO (Plain Old Java Object) 出 自 Martin Fowler, Rebecca Parsons 和 Josh MacKenzie 在西元 2000 年 9 月的一場演講裡。該場演講中強調將商業邏輯使用「一般 Java 物件」開發的好處，而非依照 EJB 的架構將商業邏輯放在 Entity Bean (Enterprise JavaBean 的一種，主要用於處理商業邏輯) 裡。

作者認為一般開發者會排斥這樣的做法的原因是因為「一般 Java 物件」沒有一個響亮或特別的名稱，因而缺乏被使用的誘因，故將之取名為 POJO。所以 POJO 是一個簡單的類別，它可以包含商業或持久化邏輯等，但不是 Java Bean、Entity Bean，不具有任何特殊角色，不繼承、不實作任何其他 Java 框架的 class 或 interface。

若您有過使用其他 Java 框架或函式庫協助開發的經驗，必然體驗過許多框架或函式庫會要求繼承特定 class 或實作特定 interface。一旦如此，程式碼必須依賴框架，稱為「侵入式 (invasive)」開發。這類經驗，發生在 EJB 2 或是早期的 Struts、WebWork、Tapestry 等數不清的框架內。

Spring 盡可能避免這種情況，因而大量使用 POJO 元件，或是使用一些 annotation，如本章範例程式。使用 Spring 框架的程式，由外觀上看不太出來是 Spring 程式。因為非侵入式開發，也因此必要時可以輕鬆移植到其他框架。

18.2 Spring 開發手法 DI 介紹

18.2.1 DI 簡介

在物件導向的系統裡，必須藉由物件和物件間的互動，來演示商業邏輯。此時，因為物件之間互有關聯 (dependency)，因此常見地一個物件會擁有其他物件的參照 (reference)，造成程式碼糾結一團 (highly coupled)，因而不好維護也不易測試，如以下範例：

建立介面 Soldier：

範例：/spring01/src/course/c01/Soldier.java

```
1  public interface Soldier {
2      void destroyTarget();
3  }
```

建立類別 Gun：

範例：/spring01/src/course/c01/Gun.java

```
1  public class Gun {
2      public void attack() {
3          System.out.println("Gun shoot!!");
4      }
5  }
```

建立類別 MySoldierNG，並實作介面 Soldier：

範例：/spring01/src/course/c01/MySoldierNG.java

```
1  public class MySoldierNG implements Soldier {
2      Gun weapon;
3      public MySoldierNG() {
4          this.weapon = new Gun();
5      }
6      @Override
7      public void destroyTarget() {
8          this.weapon.attack();
9      }
10 }
```

因類別 MySoldierNG 的建構子內使用「this.weapon = new Gun()」建立自己的武器物件 Gun，這會導致 2 個類別緊密依賴，失去武器抽換的彈性。若要進行「單元測試 (unit test)」，因為建立 MySoldierNG 物件必然伴隨 Gun 物件的建立，因此測試結果就邏輯上而言無法切割。

建立類別 SoldierMainNG，以 main 方法執行程式：

範例：/spring01/src/course/c01/SoldierMainNG.java

```
1  public class SoldierMainNG {
2      public static void main(String[] args) {
3          new MySoldierNG().destroyTarget();
4      }
5  }
```

使用「關聯注入 (dependency Injection)」，簡稱「DI」可以解決此一問題。針對所有武器，統一建立介面 Weapon：

範例：/spring01/src/course/c01/Weapon.java

```
1  public interface Weapon {
2      void attack();
3  }
```

所有武器，包含 Gun，都實作該介面：

範例：/spring01/src/course/c01/Gun.java

```
1  public class Gun implements Weapon {
2      public void attack() {
3          System.out.println("Gun shoot!!");
4      }
5  }
```

類別 MySoldierOK 的建構子改允許武器由參數以 interface 的相依型態輸入，如此 Soldier 的實作類別不會綁定任何實體武器。可以視需要抽換，此為「關聯注入 (dependency injection)」。必須提醒的是注入的地方不限於建構子 (constructor)，一般方法 (method) 也可以：

範例：/spring01/src/course/c01/MySoldierOK.java

```
1  public class MySoldierOK implements Soldier {
2      Weapon weapon;
3      public MySoldierOK (Weapon w) {
4          this.weapon = w;
5      }
6      @Override
7      public void destroyTarget() {
8          this.weapon.attack();
9      }
10 }
```

使用類別 SoldierMainOK 執行：

範例：/spring01/src/course/c01/SoldierMainOK.java

```
1  public class SoldierMainOK {
2      public static void main(String[] args) {
3          Weapon w = new Gun();
4          new MySoldierOK(w).destroyTarget();
```

```
5        }
6    }
```

進行單元測試時，可以建立一個偽冒的 (mock) 武器物件傳入，因此實作 Soldier 和 Weapon 介面的 2 個類別就可以分開測試：

範例：/spring01/src/course/c01/MySoldierOkUnitTest.java

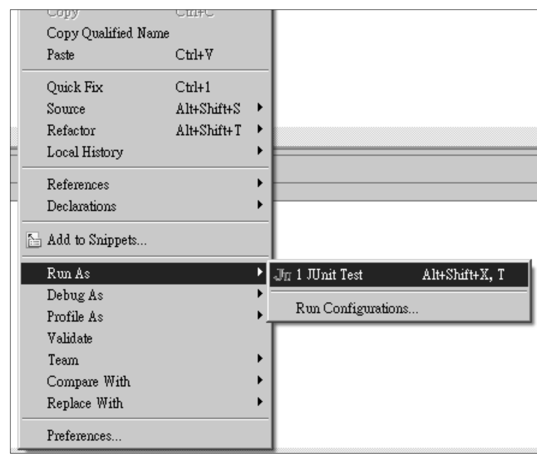
```
1    import static org.mockito.Mockito.*;
2    import org.junit.Test;
3    public class MySoldierOkUnitTest {
4        @Test
5        public void test() {
6            Weapon mockWeapon = mock(Weapon.class);
7            MySoldierOK soldier = new MySoldierOK(mockWeapon);
8            soldier.destroyTarget();
9            verify(mockWeapon, times(1)).attack();
10       }
11   }
```

說明

4	方法前使用 @Test 表示該方法將被用於「單元測試 (unit test)」。
6	進行單元測試時，需要測試的是有商業邏輯內容實作的類別，interface 無需測試，也沒有內容可供測試。我們希望可以將所有類別獨立區隔，所以若測試失敗，方便釐清哪個類別的實作內容有問題，因此使用 mock(Weapon.class) 建立偽冒物件。該物件的參考型別宣告為 Weapon mockWeapon，因此程式碼可以通過編譯。我們的目的是要測試類別 MySoldierOK 是否功能正常，所以不需要關注 Weapon 的實作內容，用偽冒物件即可。
7	將偽冒物件 mockWeapon 注入 MySoldierOK 類別的建構子，如此可以建立物件。
8	呼叫 MySoldierOK 物件的 destroyTarget() 方法摧毀目標。我們要測試的是該方法是否可以正常轉呼叫 Weapon 的 attack() 方法，但不在乎 Weapon 的 attack() 方法內容實作為何 (因為是偽冒物件，即假物件，也無從在乎)。
9	測試目的不盡相同，本例驗證物件 mockWeapon 的 attack() 方法是否只被呼叫 1 次。

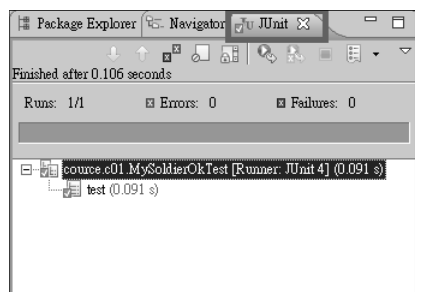
使用 Eclipse 執行單元測試的步驟為：

Step01 在程式碼區塊的任一地方，點擊滑鼠右鍵，點選「Run As」，再點選「JUnit Test」：



▲ 圖 18-1 使用 Eclipse 執行 JUnit Test

Step02 Eclipse 將自動出現 JUnit 的測試結果視窗，如下。本例顯示綠色燈號，因此通過單元測試：



▲ 圖 18-2 單元測試結果

18.2.2 使用 Spring 的關聯注入 (DI)

物件導向程式設計時，若類別間關係緊密，稱為「tightly coupling」，常不利於程式維護。類別間關係 (dependency) 的緊密度 (coupling) 有如雙面刃：

1. 過緊的關係將造成測試及程式碼重複使用困難，不易理解；且容易出現「打地鼠式 (whack-amole) 的臭蟲 (bug)」，亦即修復一個問題，又冒出另一個問題。

2. 類別間也不能完全沒有關聯，否則物件無法互動。

藉由第三方的函式庫或框架，如 Spring，所提供的「關聯注入 (DI)」，就扮演著物件間協調的角色，可以設定或縫合 (wire) 物件間的關聯性：

1. Spring 使用「設定」的方式提供關聯注入 (DI) 的功能，可分 2 種。以下範例將分別舉例說明：
   - XML-based：使用 XML 檔案設定類別間的關聯性。
   - Java-based：使用 Java 類別搭配 annotation 設定類別間的關聯性。
2. 無論何種，都必須建立一個實作 ApplicationContext 介面的 Spring 框架物件，等同於取得 Spring 執行時期環境；再由該物件取得其他的物件，通常稱為 bean 元件。

使用 XML 設定檔

Spring 的 XML 設定檔：

範例：/spring01/src/course/c01/di/xmlBased/applicationContext.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <beans xmlns="http://www.springframework.org/schema/beans"
3      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4      xsi:schemaLocation="http://www.springframework.org/schema/beans
5      http://www.springframework.org/schema/beans/spring-beans.xsd">
6      <bean id="weapon" class="course.c01.Gun" />
7      <bean id="mySoldier" class="course.c01.MySoldierOK">
8          <constructor-arg ref="weapon" />
9      </bean>
10 </beans>
```

說明

1	XML 檔案宣告。
2-5	檔案內的標籤宣告。
6	Spring 將自動建立 course.c01.Gun 物件，稱 bean 元件，id 為 weapon。
7-8	Spring 將自動建立 course.c01.MySoldierOK 物件，id 為 mySoldier。因建構子需要實作 Weapon 介面的物件，Spring 將由其控管的 bean 元件中找合適的自動關聯注入，即為前述 id 為 weapon 的 bean 元件。

使用 XML 設定檔啟動 Spring 框架：

範例：/spring01/src/course/c01/di/xmlBased/SoldierMain.java

```
1  public class SoldierMain {
2      public static void main(String[] args) {
3          ApplicationContext context = new ClassPathXmlApplicationContext(
4                  "course/c01/di/xmlBased/applicationContext.xml");
5          Soldier soldier = context.getBean(Soldier.class);
6          soldier.destroyTarget();
7      }
8  }
```

說明

3-4	使用類別 ClassPathXmlApplicationContext，並指定 classpath 上的 xml 檔案 course/c01/di/xmlBased/applicationContext.xml 建立 Spring 的執行環境，物件型態屬於 ApplicationContext。
5	類似工廠方法的概念，利用 class 指定物件型態，自 Spring 的執行環境取出 Soldier 元件。
6	執行 Soldier 元件的 destroyTarget() 方法。

使用 Java 設定類別

類似的概念，以 Spring 的設定類別取代 XML 設定檔：

範例：/spring01/src/course/c01/di/javaBased/SoldierConfig.java

```
1  @Configuration
2  public class SoldierConfig {
3      @Bean
4      public Weapon getWeapon() {
5          return new Gun();
6      }
7      @Bean
8      public Soldier getSoldier() {
9          return new MySoldierOK(getWeapon());
10     }
11 }
```

說明

1	使用 @Configuration 宣告為 Spring 的設定類別。
3	在方法上使用 @Bean 宣告，表示 Spring 啟動時將執行該方法，並將產出的物件納管為 bean 元件。

9	建立 MySoldierOK 物件需要建構子注入 Weapon 介面的實作物件。

使用設定類別啟動 Spring 框架：

範例：/spring01/src/course/c01/di/javaBased/SoldierMain.java

```
1  public class SoldierMain {
2      public static void main(String[] args) {
3          ApplicationContext context =
4              new AnnotationConfigApplicationContext (SoldierConfig.class);
5          Soldier soldier = context.getBean(Soldier.class);
6          soldier.destroyTarget();
7      }
8  }
```

說明

3-4	使用類別 AnnotationConfigApplicationContext，並指定設定類別 SoldierConfig 建立 Spring 的執行環境，物件型態屬於 ApplicationContext。

結果

```
Gun shoot!!
```

18.3 Spring 開發手法 AOP 介紹

18.3.1 AOP 簡介

AOP 是「面向導向程式開發 (aspect-oriented programming)」的縮寫。這和常見的「OOP，物件導向程式開發 (object-oriented programming)」並沒有衝突。

Spring 使用 DI 讓程式碼可以鬆耦合 (loose coupling)，並促進重複使用 (reusable)；AOP 則是建構在 DI 的基礎上，且具有一樣的效果！

除此之外，AOP 經常扮演執行物件導向設計原則裡的「關注分離 (separation of concerns)」的角色。簡單來說，程式碼是為了提供商業邏輯的服務而存在，而每個商業邏輯的服務裡，通常會存在一些和商業邏輯無關的程式碼，如：

- 日誌記錄 (logging) 功能
- 交易控制 (transaction) 功能
- 安全檢核 (security) 功能

以物件導向的設計原則來看：

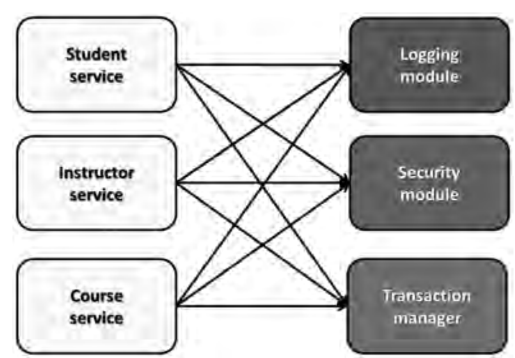
- 關注分離：商業邏輯相關的程式碼裡不應該看到以上功能性的程式碼。
- 重複使用：以上功能性程式碼基本上不會改變，應該要可以共用。

因此 Spring 推出 AOP 模組進行支援！

18.3.2 未使用 AOP 時的程式碼亂象

在使用 Spring AOP 前，程式碼有幾種常見現象：

1. 不同功能性的程式碼將混雜在商業邏輯的程式碼裡，並可能造成大量重複的程式碼，明顯違反 SRP、DRY 等 OO 法則。雖然可以事先將不同功能性的程式碼予以模組化，最終只以一行程式碼出現在商業邏輯程式碼中，如使用 log4j 函式庫，但商業邏輯程式碼中畢竟還是出現了如 logger.info("…") 的程式碼。
2. 不同功能性的程式碼間可能還會互相混雜，如安全性檢核的程式碼又夾雜了日誌記錄的程式碼，造成程式碼的維護和測試更加困難！如：



▲ 圖 18-3 使用 AOP 前程式間錯綜複雜的關係

使用以下範例示範：

1. 建立告警類別 Alert，在士兵類別使用武器介面摧毀目標的前後，都應該示警：

範例：/spring01/src/course/c01/aop/Alert.java

```
1  public class Alert {
2      public void beforeAttack() {
3          System.out.println("Before Attack....");
```

碁峯 www.gotop.com.tw

```
4     }
5     public void afterAttack() {
6         System.out.println("After Attack....");
7     }
8 }
```

建立不良示範的士兵類別。比照使用 DI 將武器的實作注入建構子的方式，一併注入 Alert 物件。如此，原先士兵的 destroyTarget() 方法只是單純將實作內容「委派 (delegate)」或「轉交 (forward)」給武器的 attack() 方法 (可參考「Java SE8 OCPJP 進階認證指南」一書的「5.3 使用複合」)，如今卻必須在方法前後呼叫告警類別的示警方法，這和在商業邏輯的實作中加入日誌記錄 (log) 的需求相似。如以下行 10 和 12：

範例：/spring01/src/course/c01/aop/MySoldierNG.java

```
1  public class MySoldierNG implements Soldier {
2      Weapon weapon;
3      Alert alert;
4      public MySoldierNG (Weapon w, Alert a) {
5          this.weapon = w;
6          this.alert = a;
7      }
8      @Override
9      public void destroyTarget() {
10         this.alert.beforeAttack();
11         this.weapon.attack();
12         this.alert.afterAttack();
13     }
14 }
```

將武器和告警物件注入不良示範的士兵物件中，並執行程式：

範例：/spring01/src/course/c01/aop/SoldierMain.java

```
1  public class SoldierMain {
2      public static void main(String[] args) {
3          Weapon w = new Gun();
4          Alert a = new Alert();
5          new MySoldierNG(w, a).destroyTarget();
6      }
7  }
```

結果

```
Before Attack....
Gun shoot!!
After Attack....
```

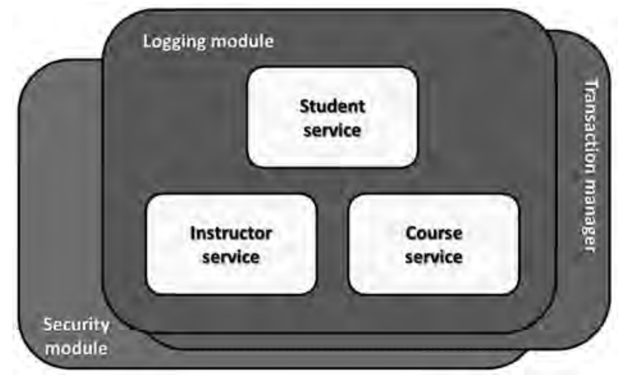
結論：

1. 類別 Alert 詮釋功能性模組，目的在提供日誌記錄。
2. 類別 MySoldierNG 詮釋商業邏輯。為了在使用 Weapon 類別前後進行記錄，必須使用「侵入式」的開發方式。導致 destroyTarget() 方法混雜了功能性模組和商業邏輯。
3. 不建議的開發方式。

18.3.3 導入 AOP 的概念

AOP 開發概念類似繪製地圖時的「套圖」概念。若一張地圖上要同時包含行政區、等高線、山川、交通、人口分布等資訊，則該圖必然是密密麻麻，不僅難製作，也難維護。一般會先繪製不同的圖層，有需要時再將圖層重疊，如行政區圖疊上人口分布資訊等。

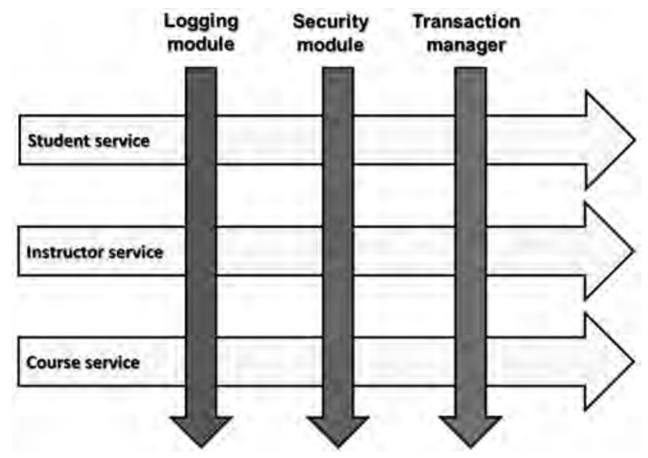
所以，程式設計師可以專注開發 Student、Instructor、Course 等商業邏輯服務，及 Logging、Transaction、Security 等共用功能模組。在有需要時，便將功能模組「套疊」在商業邏輯服務上！只需明確定義套疊的「位置」即可。如：



▲ 圖 18-4 使用 AOP 後程式間關係可重構為套圖的概念

換個角度看，套圖其實有切面的概念。共用功能模組將於適當位置切入商業邏輯服務：

碁峰 www.gotop.com.tw



▲ 圖 18-5 共用功能模組切入商業邏輯服務

使用 AOP 前，幾個名詞必須知道：

1. Aspect：

   稱為面向或切面。實際的程式內容稱為「Advice」，如先前舉例的日誌記錄或交易控制等功能性模組。

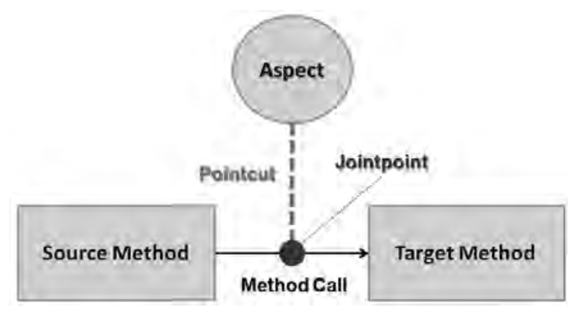
2. Joinpoint：

   Advice 在商業邏輯程式中被執行的時候。可能是某個商業方法被呼叫之前 (before) 或之後 (after)，或其他。

3. Pointcut：

   描述某個 Advice 在哪些 Joinpoint 時被套用至商業邏輯程式之上，常以文字敘述。

示意圖如下：



▲ 圖 18-6 Aspect + Pointcut + Joinpoint