

1.Spring 基礎

Spring 概述

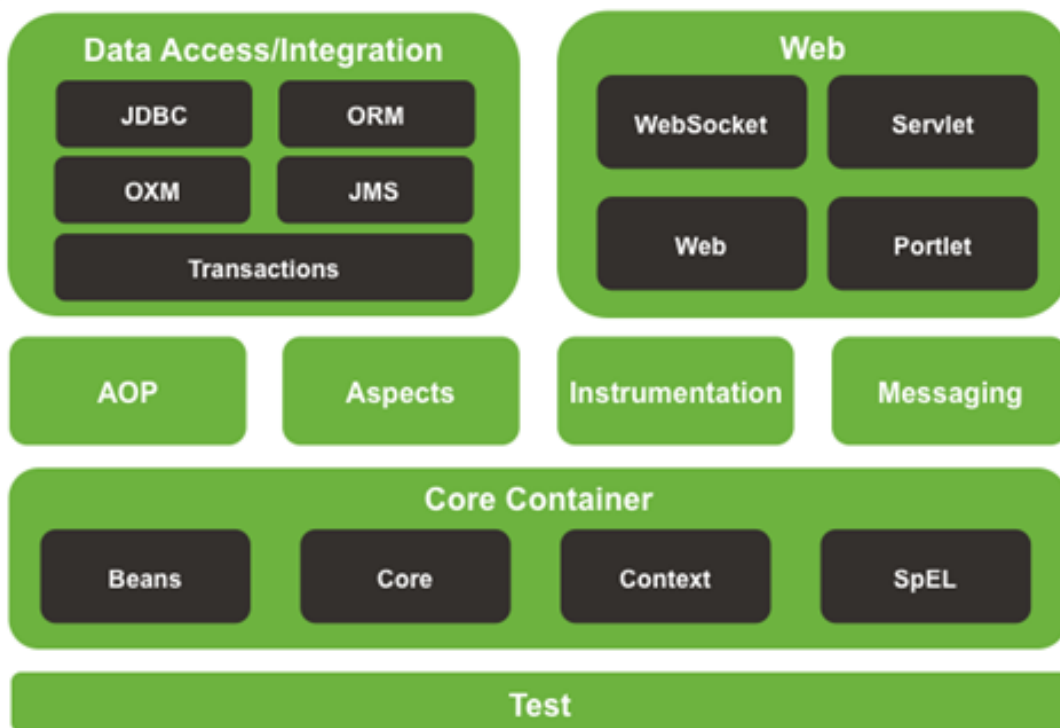
- Spring framework 輕量級企業級開發的一體式解決方案，意即可基於Spring 提供的技術框架與libraries 解決絕大部份 Java EE開發上的需求。
- Spring framework 主要提供了IoC容器、AOP、資料存取、Web開發、Message、測試等相關技術的支持。
- Spring framework使用簡單的POJO(Plain Old Java Object，即無任何限制的普通Java Object)來進行企業級開發。每一個被Spring 管理的Java Object都稱之為Bean；而Spring 提供了一個IoC(Inversion of Control)容器用來初始化物件，解決物件間的依賴管理(Dependency Injection)和物件的使用。

Spring 模組

- Spring 是模組化的，意指可隨專案需求而只使用需要的Spring 模組。



Spring Framework Runtime



上圖中的每一個最小單元，Spring 都至少有一個對應的jar檔。

- 核心容器(Core Container)
 - Spring-Core：核心工具類，Spring 其它模組大量使用Spring-Core
 - Spring-Beans：Spring 定義Bean的支持
 - Spring-Context：runtime 時Spring 容器
 - Spring-Context-Support：Spring容器對第三方開發的jar的整合support
 - Spring-Expression：使用表達式語言在運行時查詢和操作物件
- AOP
 - AOP：基於代理的AOP支持
 - Aspects：基於AspectJ的AOP支持
- Messaging
 - Messaging：對message架構和協定的支持
- Web
 - Web：提供基礎的Web整合的功能，在Web專案中提供Spring的容器
 - Webmvc：提供基於Servlet 的Spring MVC
 - WebSocket：提供WebSocket功能
 - Portlet：提供Portlet環境支持
- Data Access/Integration
 - JDBC：提供以JDBC存取資料庫的支持

- TX：提供分散式交易管理的支持
- ORM：提供對物件/資料表關係映射技術的支持
- OXM：提供對物件/xml映射技術的支持
- JMS：提供對JMS的支持

Spring 生態

- Spring 發展到現在已不僅僅是Spring 框架本身的內容，Spring 目前提供了大量基於Spring framework 的專案，可以用來更深入地降低應用系統開發難度，提高開發效率。
- 主要專案：
 - Spring Boot：使用預設優於配置來實現快速開發
 - Spring XD：用來簡化大數據應用開發
 - Spring Cloud：為分散式系統開發提供工具集
 - Spring Data：為關聯式資料庫和NoSQL資料庫存取的支持
 - Spring Integration：通過messaging機制對企業整合模式的支持
 - Spring Batch：簡化及優化大量數據的批次處理操作
 - Spring Security：通過認證和授權保護的支持
 - Spring Social：與社交網絡API(如：Facebook、Linkedin、Twitter)的整合
 - Spring Mobile：提供對手機設備檢測的功能，給不同設備返回不同頁面的支持
 - Spring for Android：主要提供在Android上消費RESTful API的功能
 - Spring Web Flow：基於Spring MVC，提供基於流程控制的Web 應用程式開發
 - Spring LDAP：簡化使用LDAP開發
 - Spring Session：提供API來實現管理Session

Spring 配置方式

- xml 配置

在 Spring 1.x 時代，使用 xml 設定方式來配置 Bean，隨著專案擴大，常需要把xml 設定放置在不同的配置文件裡。

- Annotation 配置

在Spring 2.x時代，隨著 JDK 1.5 對annotation的支持，Spring 提供了聲明Bean的注解(如：@Component、@Service)並搭配 @Autowired 大大減少了配置量。然而因應大型專案需要，通常會混用xml 配置和 annotation配置，比如會把應用系統的基本配置(如資料庫配置)使用xml，而業務類的配置則使用annotation。

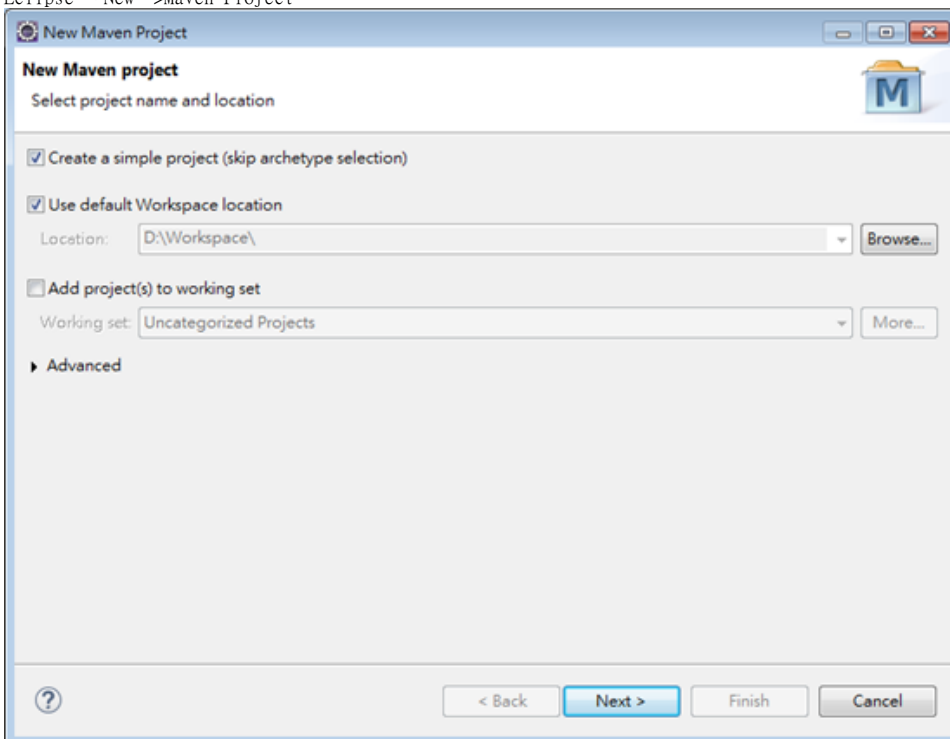
- Java 配置

從Spring 3.x到現在，Spring 又再提供了Java配置的方式；使用Java 配置可以讓人更理解配置的Bean(因為有直覺的 new Object的操作方式)。

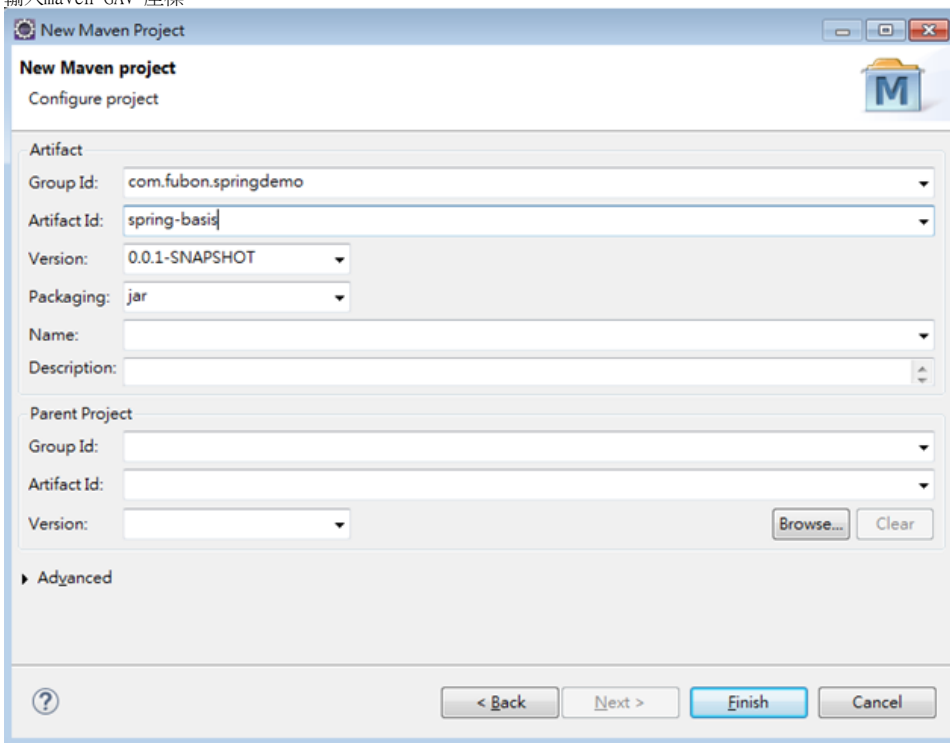
Spring 4.x 和Spring Boot 都建議使用Java 配置方式。

Spring 專案建立

- Eclipse ->New ->Maven Project

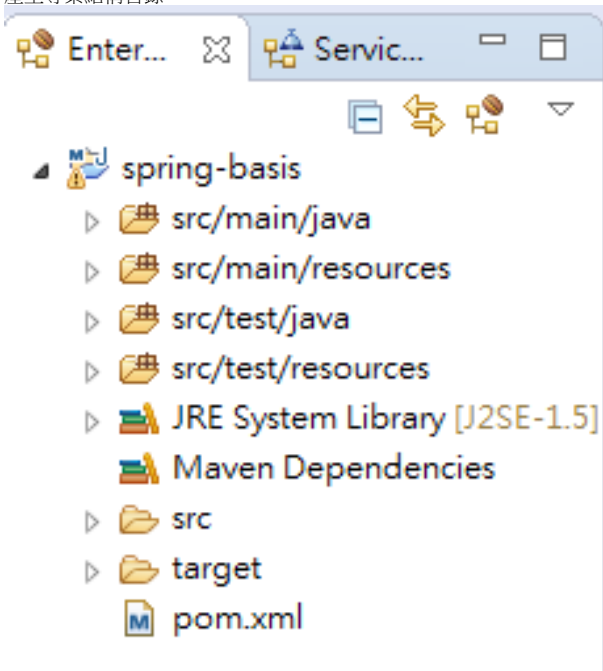


- 輸入maven GAV 座標



The 'New Maven Project' dialog box is shown. It has a title bar with standard window controls. Below the title bar, there's a section 'New Maven project' with a 'Configure project' link and a Maven logo. The 'Artifact' section contains fields for 'Group Id' (com.fubon.springdemo), 'Artifact Id' (spring-basis), 'Version' (0.0.1-SNAPSHOT), and 'Packaging' (jar). There are also empty fields for 'Name' and 'Description'. The 'Parent Project' section has empty fields for 'Group Id', 'Artifact Id', and 'Version', along with 'Browse...' and 'Clear' buttons. At the bottom, there's an 'Advanced' section (collapsed), a help icon, and navigation buttons: '< Back', 'Next >', 'Finish', and 'Cancel'.

- 產生專案結構目錄



- 修改maven pom.xml

pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi
schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.fubon.springdemo</groupId>
  <artifactId>spring-basis</artifactId>
  <version>0.0.1-SNAPSHOT</version>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <spring.version>4.2.5.RELEASE</spring.version>
```

```

        <java.version>1.8</java.version>
    </properties>

    <dependencies>

        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>${spring.version}</version>
        </dependency>

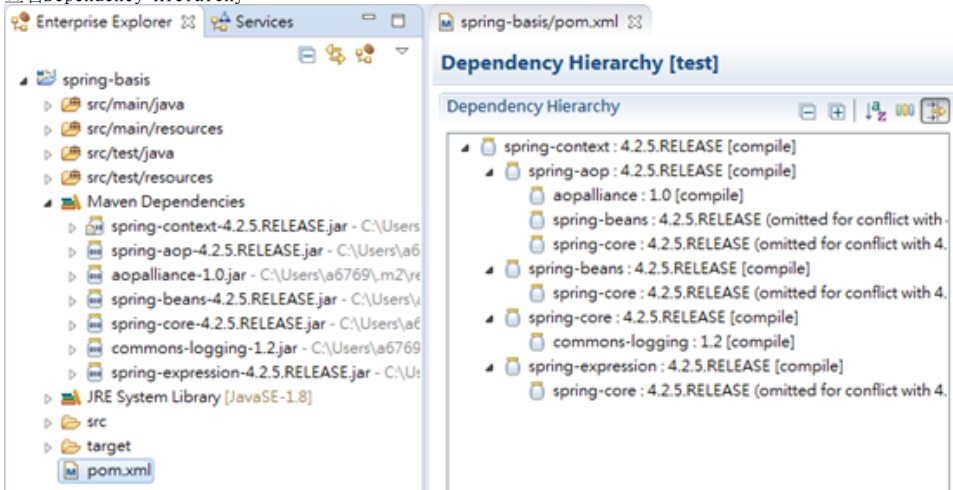
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <artifactId>maven-compiler-plugin</artifactId>
                <configuration>
                    <source>${java.version}</source>
                    <target>${java.version}</target>
                </configuration>
            </plugin>
        </plugins>
    </build>

</project>

```

- 更新專案。點擊專案右鍵→Maven→Update Project→OK
- 查看Dependency Hierarchy



IoC and DI

- IoC是什麼
 - IoC—Inversion of Control，即「控制反轉」，不是什麼技術，而是一種設計思想。在Java開發中，IoC意味著將你設計好的對象交給容器控制，而不是傳統的在你的對象內部直接控制。
 - 誰控制誰，控制什麼：傳統Java SE程序設計，我們直接在物件內部通過 new 進行創建物件，是程序主動去創建依賴物件；而IoC是有專門一個容器來創建這些物件，即由IoC容器來控制物件的創建；誰控制誰？當然是IoC 容器控制了對象；控制什麼？那就是主要控制了外部資源獲取（不只是對象包括比如文件等）。
 - 為何是反轉，哪些方面反轉了：有反轉就有正轉，傳統應用程式是由我們自己在對象中主動控制去直接獲取依賴對象，也就是正轉；而反轉則是由容器來幫忙創建及注入依賴對象；為何是反轉？因為由容器幫我們查找及注入依賴對象，對象只是被動的接受依賴對象，所以是反轉。
- IoC能做什麼
 - 它能使得我們設計出鬆散耦合、更優良的程序。傳統應用程式都是由我們在類內部主動創建依賴對象，從而導致類與類之間高耦合，難於測試；有了IoC容器後，把創建和查找依賴對象的控制權交給了容器，由容器進行注入組合對象，所以對象與對象之間是鬆散耦合，這樣也方便測試，利於功能重複使用，更重要的是使得程序的整個體系結構變得非常靈活。
- Dependency injection-DI
 - 其實IoC 和DI是同一個概念的不同角度的描述，Spring 是通過依賴注入來實現IoC概念，組件之間依賴關係由容器在運行期決定，形象的說，即由容器動態的將某個依賴關係注入到組件之中。依賴注入的目的並非為軟體系統帶來更多功能，而是為了提升組件重用的頻率，並為系統搭建一個靈活、可擴展的平台。通過依賴注入機制，我們只需要通過簡單的配置，而無需任何代碼就可指定目標需要的資源，完成自身的業務邏輯，而不需要關心具體的資源來自何處，由誰實現。

（原文網址：<https://read01.com/RKmmDe.html>）

- 由Spring IoC容器(ApplicationContext)負責建立物件和維持物件間的依賴關係，而不是由開發人員於程式碼中自行去建立物件和解決物件依賴。
- 對於想要交由Spring 容器來管理依賴關係的物件，必須宣告為Bean，由Spring IoC容器負責建立Bean，再將此功能類Bean注入到需要此Bean的地方。

Spring dependency injection 宣告方式

- 聲明Bean 的注解
 - @Component：一般組件，沒有特定角色
 - @Service：在業務邏輯層(service 層)使用
 - @Repository：在資料庫存取層(dao層)使用
 - @Controller：在層現層(MVC → Spring MVC)使用
- 注入Bean 的注解
 - @Autowired：Spring 提供的依賴注入注解
 - @Inject：JSR-330提供的注解
 - @Resource：JSR-250提供的注解
 - 可注解在 set 方法或屬性上，建議注解在屬性上，較為易讀

ANNOTATION	PACKAGE	SOURCE
@Resource	javax.annotation	Java
@Inject	javax.inject	Java
@Qualifier	javax.inject	Java
@Autowired	org.springframework.bean.factory	Spring

@Autowired and @Inject

1. Matches by Type
2. Restricts by Qualifiers
3. Matches by Name

@Resource

1. Matches by Name
2. Matches by Type
3. Restricts by Qualifiers (ignored if match is found by name)

(原本網址：<http://blogs.sourceallies.com/2011/08/spring-injection-with-resource-and-autowired/>)

示例

- 設計功能類的Bean
 - DayGreeter

```

DayGreeter

package com.fubon.demo.component;

import org.springframework.stereotype.Component;

@Component //1
public class DayGreeter {

    public String sayHello(String name){
        return "Good morning " + name + " !";
    }

}

```

- NightGreeter

NightGreeter

```
package com.fubon.demo.component;

import org.springframework.stereotype.Component;

@Component //1
public class NightGreeter {

    public String sayHello(String name){
        return "Good night " + name + " !";
    }

}
```

- 說明：

使用@Component 注解，聲明當前DayGreeter和NightGreeter classes是Spring 管理的Bean。其中，使用@Component、@Service、@Repository、@Controller是等效的，可根據class 的角色來選用。

- 設計業務邏輯類的Bean
 - GreetService

GreetService

```
package com.fubon.demo.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import com.fubon.demo.component.DayGreeter;
import com.fubon.demo.component.NightGreeter;

@Service //1
public class GreetService {

    @Autowired //2
    DayGreeter dayGreeter;

    @Autowired //2
    NightGreeter nightGreeter;

    public String greet(int hour String name) {
        if (hour <= 18){
            return dayGreeter.sayHello(name);
        } else {
            return nightGreeter.sayHello(name);
        }
    }

}
```

- 說明：

使用@Service注解，聲明當前GreetService class是Spring 管理的Bean。

使用@Autowired注解，請求Spring 將 DayGreeter和NightGreeter 的實體Bean 注入到 GreetService中，讓此service可以操作這些物件實例的功能。

- 使用Java 配置方式設計應用程式式配置類
 - GreeterConfig

GreeterConfig

```
package com.fubon.demo.config;

import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
```

```

@Configuration //1
@ComponentScan("com.fubon.demo") //2
public class GreeterConfig {
    //nothing
}

```

◦ 說明：

使用@Configuration注解，聲明當前class是一個配置類，取代Spring 早前使用的 configuration xml。

使用@ComponentScan注解，自動掃描參數表明的 package 下(本例含component和service2個下級package)的所有使用@Component、@Service、@Repository、@Controller的classes，並向Spring 容器註冊為Bean。

- 應用程式式運行

- DemoApplication

DemoApplication

```

package com.fubon.demo.application;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;

import com.fubon.demo.config.GreeterConfig;
import com.fubon.demo.service.GreetService;

public class DemoApplication {

    public static void main(String[] args){
        AnnotationConfigApplicationContext context = new AnnotationConfigApplicationContext
(GreeterConfig.class); //1

        GreetService greetService = context.getBean(GreetService.class); //2
        System.out.println(greetService.greet(19 "John"));

        context.close();
    }
}

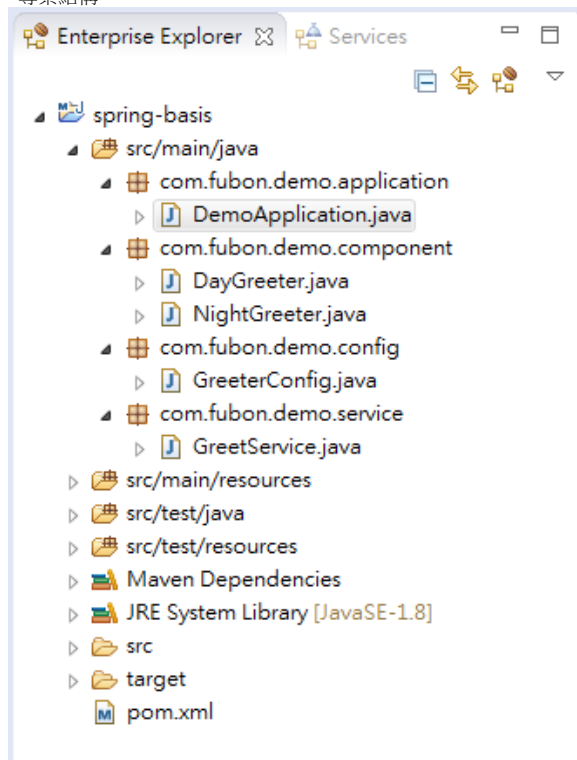
```

◦ 說明：

使用AnnotationConfigApplicationContext 作為Spring 容器，接受輸入配置類作為參數，並依配置類內容進行初始化配置。

向Spring 容器取得GreetService 類的Bean

專案結構：



執行結果：

```
二月 11, 2017 3:24:03 下午 org.springframework.con
資訊: Refreshing org.springframework.context.an
Good night, John !
二月 11, 2017 3:24:03 下午 org.springframework.con
資訊: Closing org.springframework.context.annot
```