4.Spring Boot 核心

應用程式入口和@SpringBootApplication

Spring Boot 應用程式通常有一個名為xxxApplication的入口類別，搭配main Method 入口方法作為入口點。

此應用程式標注 @SpringBootApplication 表明是Spring Boot 專案。

在main方法中使用 SpringApplication.run{xxx.class， args}，啟動 Spring Boot 專案。

BootHelloworldApplication

```
package com.fubon.springdemo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@SpringBootApplication
@RestController
public class BootHelloworldApplication {

    public static void main(String[] args) {
        SpringApplication.run(BootHelloworldApplication.class, args);
    }

    @RequestMapping("/")
    String index() {
        return "Hello Spring Boot!";
    }
}
```

@SpringBootApplication 是Spring Boot 的核心注解， source code：

@SpringBootApplication

```
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Inherited
@SpringBootConfiguration
@EnableAutoConfiguration
@ComponentScan(excludeFilters = @Filter(type = FilterType.CUSTOM, classes = TypeExcludeFilter.class))
public @interface SpringBootApplication {
        Class<?>[] exclude() default {};
        String[] excludeName() default {};

        @AliasFor(annotation = ComponentScan.class, attribute = "basePackages")
        String[] scanBasePackages() default {};

        @AliasFor(annotation = ComponentScan.class, attribute = "basePackageClasses")
        Class<?>[] scanBasePackageClasses() default {};
}
```

說明：

- @SpringBootApplication事實上是一個組合注解， 主要組合了：@SpringBootConfiguration、@EnableAutoConfiguration、@ComponentScan。
- @SpringBootConfiguration 實際上就是@Configuration，說明這是一個JavaConfig
- @ EnableAutoConfiguration讓Spring Boot 會根據class path中依賴的jar 包，來為當前專案進行自動配置。例如，添加了spring-boot-starter-web ，就會自動添加 Tomcat和Spring MVC的依賴，因此Spring Boot會對Tomcat 和Spring MVC進行自動配置。又如添加了 spring-boot-starter-data-jpa ，Spring Boot會自動進行JPA相關配置。這些預設配置我們可以根據需要進行修改。

- @ComponentScan 啟用注解自動掃描，Spring Boot 預設會自動掃描@SpringBootApplication所在類的同級package，以及下級package裡的Bean(若為JPA專案還可以掃描標注@Entity的實體類)。因此建議入口類放置的位置是groupId(或+arctifactId組合)的package下。
- exclude 和 excludeName 用於關閉指定的自動配置，比如關閉DataSource相關的自動配置：

  exclude

  ```
  @SpringBootApplication(exclude={DataSourceAutoConfiguration.class})
  ```

- scanBasePackages和scanBasePackageClasses用於指定掃描的路徑，比如指定掃描的package路徑如下：

  scanBasePackages

  ```
  @SpringBootApplication(scanBasePackages = {"com.fubon.demo.controller"})
  ```

  則重啟後，只會掃描 com.fubon.demo.controller下的Bean了。

Spring Boot的配置文件

SpringBootproperties(application.properties)yaml(application.yml)

yaml ConfigFileTypes

SpringBoot會從src/main/resources目錄或classpath下的 /config目錄或者classpath的根目錄查找application.properties或application.yml

如果要修改SpringBoot自動配置中預設設定，可以在配置文件中設定相應的參數即可

比如，專案啟動時，tomcat預設的port是『8080』，context-path是『/』，修改如下：

- **application.properties**

  application.properties

  ```
  server.port=8081
  server.context-path=/demo
  ```

- **application.yml**

  application.yml

  ```
  server:
    port: 8081
    context-path: /demo
  ```

SpringBoot預設使用properites方式進行配置，但也支持其它多種方式指定，常用的方式如下幾種，按加載的優先序列出：

- 命令行參數：`java -jar xx.jar --server.port=8081 --server.context-path=/demo`
- 作業系统環境變數：有些OS不支持使用. 這種名字，如server.port，可以使用SERVER_PORT來配置。
- 專案中的配置文件：application.properties 或者 application.yml
- 專案依賴jar包中的配置文件：application.properties 或者 application.yml

Spring Boot 提倡不使用xml 配置，但是在實際專案開發中，若有些特殊需求必須使用 xml 配置，還是可以使用Spring 提供的 @ImportResource 來加載 xml 配置，例如：

@ImportResource

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

@Configuration
@ImportResource({"classpath:some-context.xml","classpath:another-context.xml"})
public class AppConfig {

}
```

Spring Boot 常用配置設定值的列表：

http://docs.spring.io/spring-boot/docs/current/reference/html/common-application-properties.html

一般屬性值注入

- Spring ：@PropertySource 指明 properties文件位置，然後使用@Value注入值，例如：

  @PropertySource

  ```
  @PropertySource(“classpath:config.properties”)
  Public class DemoConfig{
    @Value(“${eip.url}”)
    private String url;

    //getter & setter
  }
  ```

- 在Spring Boot裡，可直接將自定屬性加到 **application.properties**，直接使用@Value注入即可。
  - application.properties：

    application.properties

    ```
    server.port=8081
    server.context-path=/demo

    guest.name=John
    ```

  - run：

BootHelloworldApplication

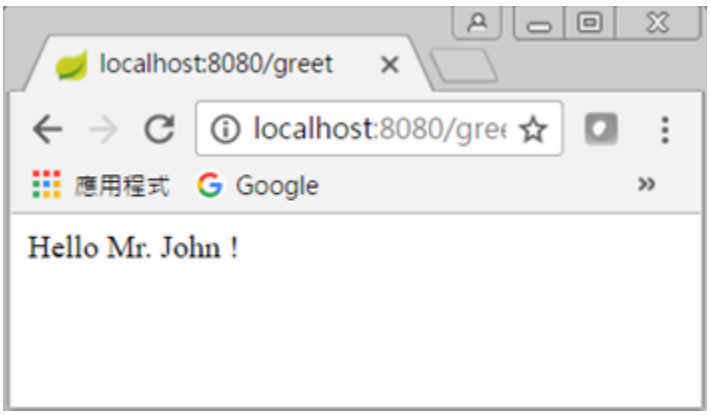
```
@SpringBootApplication
@RestController
public class BootHelloworldApplication {

	@Value("${guest.name}")
	private String guestName;

	@Value("${guest.title:Mr.}")  // Mr.
	private String guestTitle;

	public static void main(String[] args) {
		SpringApplication.run(BootHelloworldApplication.class, args);
	}

	@RequestMapping("/greet")
	String helloUser() {
		return String.format("Hello %s %s !", guestTitle, guestName);
	}
}
```



使用類型安全(Type-safe)的配置(基於properties)

若配置的設定很多，用@Value注入要很多次，若有多個Bean都會用到相同設定則顯得非常繁瑣。

Spring Boot提供了基於類於安全的配置方式，藉由@ConfigurationProperties注解，能將指定的properties file和Bean的屬性做關聯並自動注入，從而實現類型安全的配置。

- demo-config.properties (放在src/main/resources)：

  demo-config

  ```
  datasource.as400.url=jdbc:as400://10.42.16.36/LP_FBDB
  datasource.as400.username=a1234
  datasource.as400.password=q1W2E3r4
  datasource.as400.initialSize=20
  datasource.as400.maxActive=50
  datasource.as400.maxIdle=20
  datasource.as400.minIdle=10
  ```

- demo：

DataSourceConfig

```
@Component
@ConfigurationProperties(prefix = "datasource.as400" locations="classpath:demo-config.properties")
public class DataSourceConfig {

        private String url;
        private String username;
        private String password;

        //getter & seeter
}
```

- run：

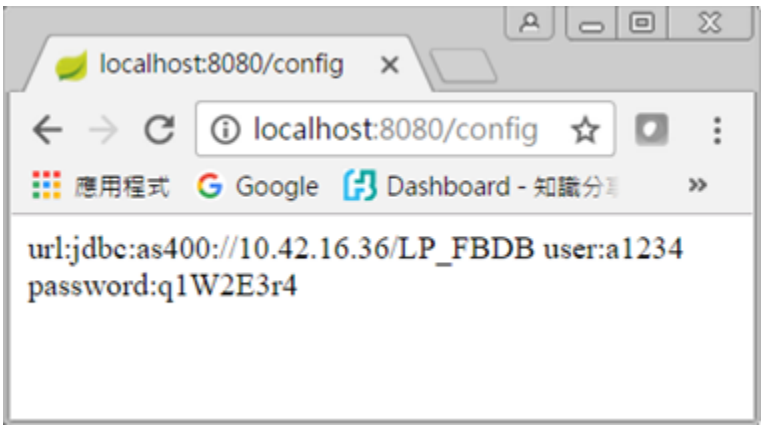
BootHelloworldApplication

```
@SpringBootApplication
@RestController
public class BootHelloworldApplication {

        @Autowired
        private DataSourceConfig dataSourceConfig;

        public static void main(String[] args) {
                SpringApplication.run(BootHelloworldApplication.class, args);
        }

        @RequestMapping("/config")
        String showConfig() {
                return String.format("url%s user%s password%s" dataSourceConfig.getUrl()
dataSourceConfig.getUsername() dataSourceConfig.getPassword());
        }
}
```



使用 Profile 配置

不同的環境可以使用不同的配置文件，使用 **application-{profile}.properties** 的命名方式，比如

- 開發環境：application-dev.properties
- 測試環境：application-test.properties
- 正式環境：application-prod.properties

搭配在 application.properties 或使用命令列參數，設定 **spring.profiles.active=[dev|test|prod]** 來指定使用哪一Profile。

(例： spring.profiles.active=prod 指明使用正式環境的設定)

- demo：在src/main/resources 目錄下分別建立 2個環境的 properties file
  - application-dev.properties：

    application-dev.properties

    ```
    server.port=8081
    ```

  - application-prod.properties：

    application-prod.properties

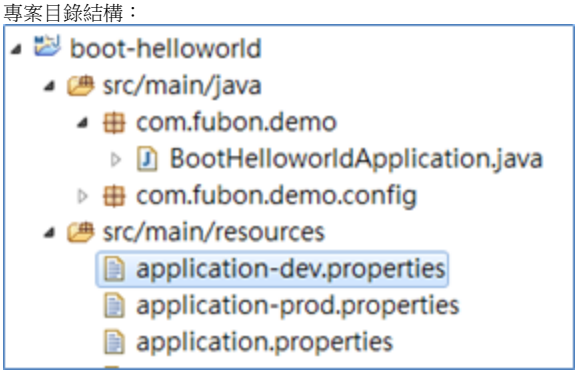
    ```
    server.port=8082
    ```

- run：
  - application.properties 指定使用profile：

    application.properties

    ```
    spring.profiles.active=dev
    ```

  - 專案目錄結構：

  - 執行log：

    ```
    Registering beans for JMX exposure on startup
    Tomcat started on port(s): 8081 (http)
    Started BootHelloworldApplication in 4.224 seconds (JVM running for 4.654)
    ```

log 配置

Spring Boot支持使用Java Util Logging、Log4J、Log4J2、Logback等作為日誌的框架，無論使用哪種框架，Spring Boot都已為當前專案的日誌框架的控制台輸出及文件輸出做好了配置。

Spring Boot 預設使用 Logback作為日誌框架。

- Logback相關配置：
  - 可以直接在application.properties中進行如下配置：

    logback

    ```
    logging.file=logback-boot-jdbc.log
    logging.level.root=INFO
    logging.level.com.fubon.demo.dao=DEBUG

    logging.pattern.file=%d{HHmmss.SSS} [%thread] %-5level %logger{35} - %msg %n
    ```

  - 也可以直接將自己慣用的logback.xml (Spring Boot建議改叫 logback-spring.xml)配置放到專案根目錄下，例如：

logback-spring.xml

```
<?xmlversion="1.0"encoding="UTF-8"?>
<configuration>
  <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
    <encoder>
      <pattern>%-4relative [%thread] %-5level %logger{35} - %msg %n</pattern>
    </encoder>
  </appender>

  <appender name="FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">
    <file>test.log</file>

    <rollingPolicy class="ch.qos.logback.core.rolling.FixedWindowRollingPolicy">
      <fileNamePattern>tests.%i.log.zip</fileNamePattern>
      <minIndex>1</minIndex>
      <maxIndex>3</maxIndex>
    </rollingPolicy>

    <triggeringPolicy class="ch.qos.logback.core.rolling.SizeBasedTriggeringPolicy">
      <maxFileSize>5MB</maxFileSize>
    </triggeringPolicy>
    <encoder>
      <pattern>%-4relative [%thread] %-5level %logger{35} - %msg%n</pattern>
    </encoder>
  </appender>

  <root level="DEBUG">
    <appender-ref ref="FILE" />
        <appender-ref ref="STDOUT" />
  </root>
</configuration>
```

- 修改為 log4j框架：
  - pom中排除對spring-boot-starter-logging的依賴，並加入spring-boot-starter-log4j的依賴

pom.xml

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter</artifactId>
    <exclusions>
        <exclusion>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-logging</artifactId>
        </exclusion>
    </exclusions>
</dependency>

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-log4j</artifactId>
    <version>1.3.8.RELEASE</version>
</dependency>
```

  - 在 src/main/resources 下放置log4j.properties

log4j.properties

```
log4j.rootLogger=INFO,A1,A2

# A1 is set to be a ConsoleAppender which outputs to System.out.
log4j.appender.A1=org.apache.log4j.ConsoleAppender
log4j.appender.A1.layout=org.apache.log4j.PatternLayout
log4j.appender.A1.layout.ConversionPattern=%d %-5p [%t] [%c{1}] %m%n


# Appender A2 writes to the file
log4j.appender.A2=org.apache.log4j.DailyRollingFileAppender
log4j.appender.A2.File=log4j-demo-helloworld.log
log4j.appender.A2.DatePattern='.'yyyy-MM-dd
log4j.appender.A2.layout=org.apache.log4j.PatternLayout
log4j.appender.A2.layout.ConversionPattern=%d %-5p [%t] [%c{1}] %m%n
```

- run：

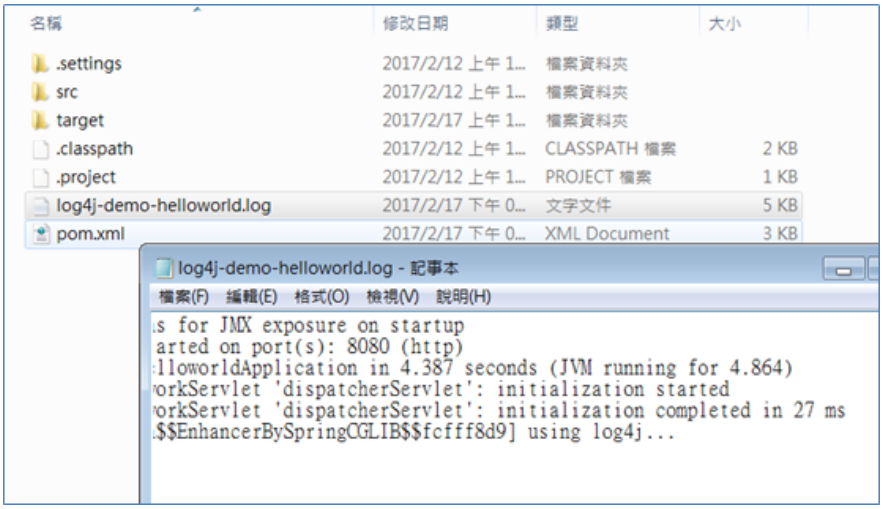
BootHelloworldApplication

```
@SpringBootApplication
@RestController
public class BootHelloworldApplication {
	private final Logger log = LoggerFactory.getLogger(this.getClass());

	public static void main(String[] args) {
		SpringApplication.run(BootHelloworldApplication.class, args);
	}

	@RequestMapping("/")
	String index() {
		log.info("using log4j...");

		return "Hello Spring Boot!";
	}
}
```



單元測試

Spring Boot的單元測試和Spring MVC的測試類似。如果使用start.spring.io 或Spring Boot CLI方式建立 Spring Boot專案，則預設都會自動加上spring-boot-starter-test 的依賴，但若是手動建立maven專案方式都必須自己加上.

此外，Spring Boot還會建立一個預設的測試類， 位於src/test/java 下。

- pom.xml：

pom

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
</dependency>
```

- Junit4 單元測試：

@SpringBootTest

```
@RunWith(SpringJUnit4ClassRunner.class)  //@RunWith(SpringRunner.class)
@SpringBootTest(classes = {BootHelloworldApplication.class}, webEnvironment=WebEnvironment.RANDOM_PORT)
public class BootHelloworldApplicationTests {
        private final Logger log = LoggerFactory.getLogger(this.getClass());

        @Autowired
        private TestRestTemplate  restTemplate;

        @Value("${guest.name:}")
        private String guestName;

        @Autowired
        private DataSourceConfig dataSourceConfig;

        @Test
        public void contextLoads() {
                log.info("guestName:" + guestName);
                log.info("url:" + dataSourceConfig.getUrl());
        }

}
```

- 說明：
    - @RunWith(SpringJUnit4ClassRunner.class) 指明啟用Spring 整合 Junit 測試. (也可使用SpringRunner.class， 它是新名稱)
    - @SpringBootTest的意思是“帶有Spring Boot支持的引導程序”（例如，加載應用程序、屬性）
      本例指明要將 BootHelloworldApplication.class 聲明的 config 配置一起載入，並且配置tomcat web容器

```
2017-02-27 12:13:58,922 INFO  [main] [TomcatEmbeddedServletContainer] Tomcat started on port(s): 54596 (http)
2017-02-27 12:13:58,929 INFO  [main] [BootHelloworldApplicationTests] Started BootHelloworldApplicationTests in 4.724 seconds (JVM running for 5.537)
2017-02-27 12:13:58,960 INFO  [main] [BootHelloworldApplicationTests] guestName:John
2017-02-27 12:13:58,960 INFO  [main] [BootHelloworldApplicationTests] url:jdbc:as400://10.42.16.36/LP_FBDB
2017-02-27 12:13:58,971 INFO  [Thread-2] [AnnotationConfigEmbeddedWebApplicationContext] Closing org.springframework.boot.context.embedded.AnnotationConfigEmb
```