5.Spring Boot 自動配置原理

@EnableAutoConfiguration查看自動配置報告

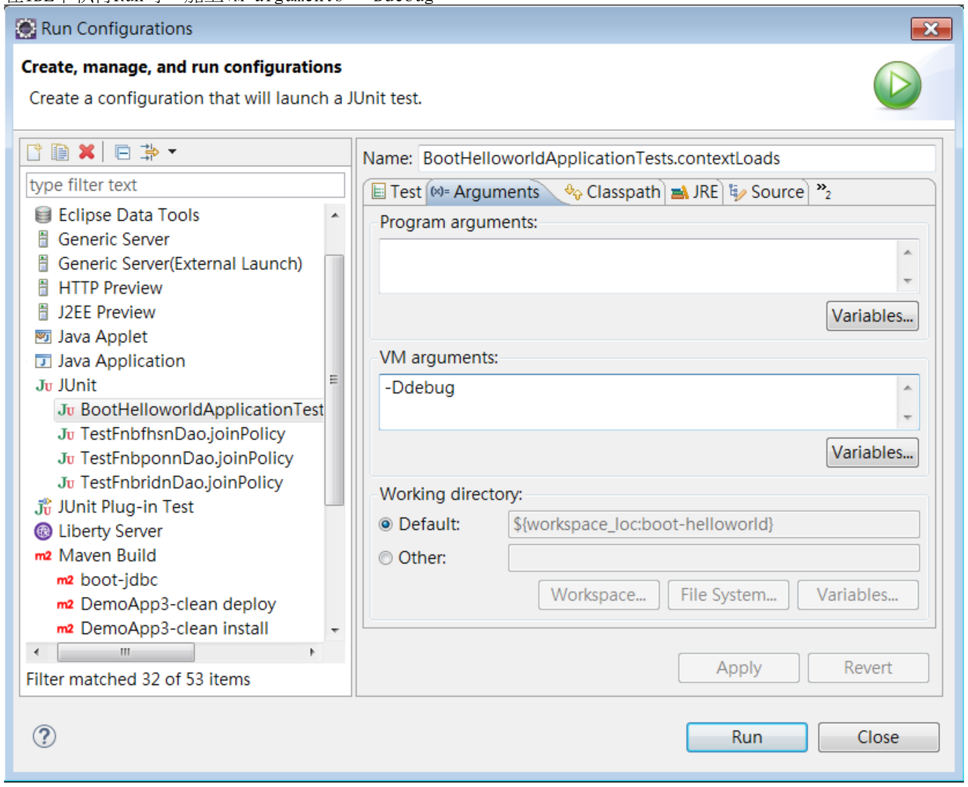
若想知道Spring Boot為當前專案做了哪些自動配置，可以用以下3種方法查看已啟用和未啟用的自動配置報告：

- 以java -jar xx.jar方式執行時， 增加 --debug參數：

```
java -jar
```

```
java -jar xx.jar --debug
```

- 在IDE中執行Run時，加上VM arguments：-Ddebug



Run Configurations

Create, manage, and run configurations

Create a configuration that will launch a JUnit test.

Name: BootHelloworldApplicationTests.contextLoads

Program arguments:

VM arguments: -Ddebug

Working directory:

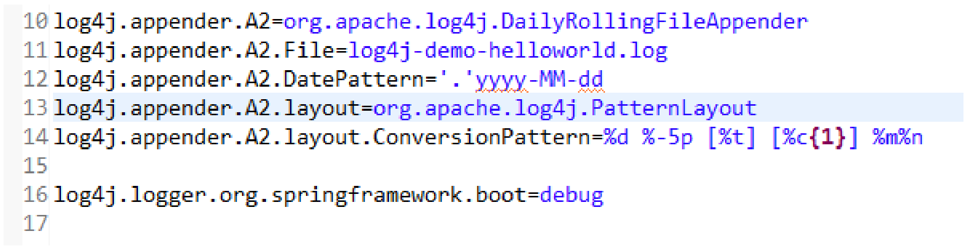
Default: ${workspace_loc:boot-helloworld}

Other:

Filter matched 32 of 53 items

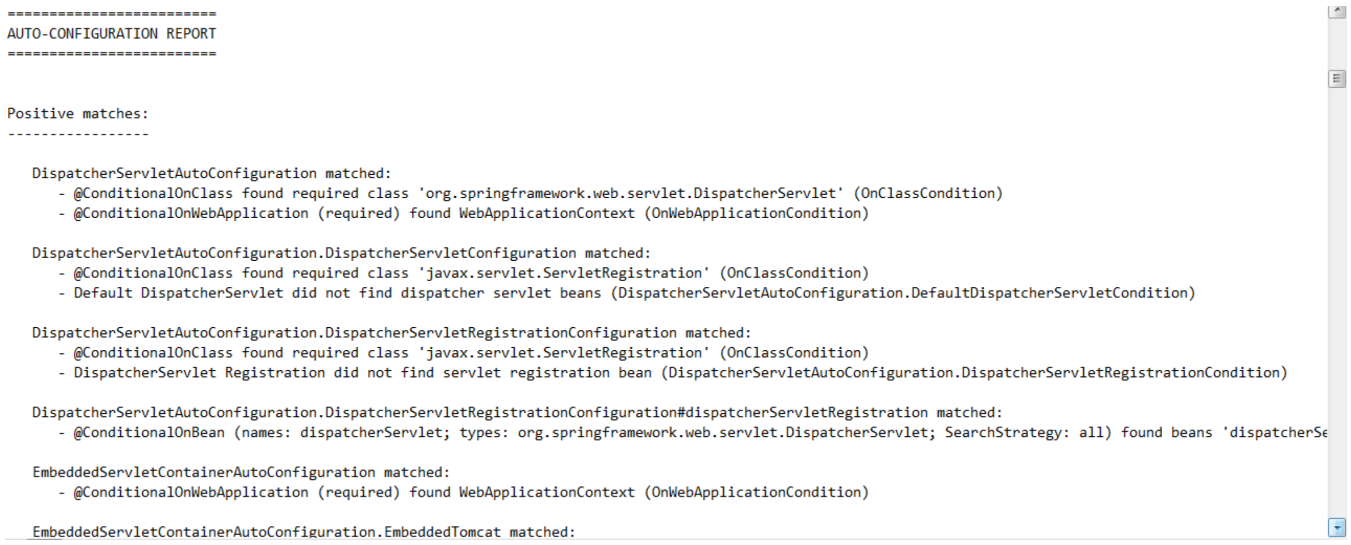
- 在 application.properties文件中增加debug=true

(這是針對預設的logback框架的配置，如果切換至其它log框架，debug=true會失效，需要在各自的log配置文件中指明，比如log4j需要添加log4j.logger.org.springframework.boot=debug)



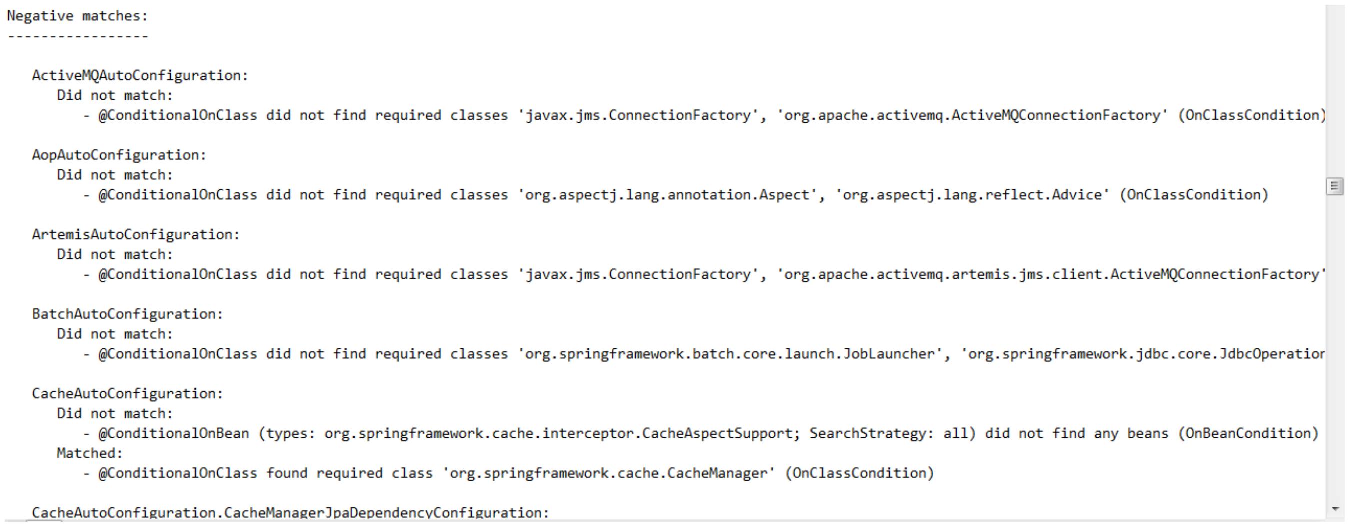
```
log4j.appender.A2=org.apache.log4j.DailyRollingFileAppender
log4j.appender.A2.File=log4j-demo-helloworld.log
log4j.appender.A2.DatePattern='.'yyyy-MM-dd
log4j.appender.A2.layout=org.apache.log4j.PatternLayout
log4j.appender.A2.layout.ConversionPattern=%d %-5p [%t] [%c{1}] %m%n

log4j.logger.org.springframework.boot=debug

```

例：已啟用的自動配置：



```
=========================
AUTO-CONFIGURATION REPORT
=========================


Positive matches:
-----------------

   DispatcherServletAutoConfiguration matched:
      - @ConditionalOnClass found required class 'org.springframework.web.servlet.DispatcherServlet' (OnClassCondition)
      - @ConditionalOnWebApplication (required) found WebApplicationContext (OnWebApplicationCondition)

   DispatcherServletAutoConfiguration.DispatcherServletConfiguration matched:
      - @ConditionalOnClass found required class 'javax.servlet.ServletRegistration' (OnClassCondition)
      - Default DispatcherServlet did not find dispatcher servlet beans (DispatcherServletAutoConfiguration.DefaultDispatcherServletCondition)

   DispatcherServletAutoConfiguration.DispatcherServletRegistrationConfiguration matched:
      - @ConditionalOnClass found required class 'javax.servlet.ServletRegistration' (OnClassCondition)
      - DispatcherServlet Registration did not find servlet registration bean (DispatcherServletAutoConfiguration.DispatcherServletRegistrationCondition)

   DispatcherServletAutoConfiguration.DispatcherServletRegistrationConfiguration#dispatcherServletRegistration matched:
      - @ConditionalOnBean (names: dispatcherServlet; types: org.springframework.web.servlet.DispatcherServlet; SearchStrategy: all) found beans 'dispatcherSe

   EmbeddedServletContainerAutoConfiguration matched:
      - @ConditionalOnWebApplication (required) found WebApplicationContext (OnWebApplicationCondition)

   EmbeddedServletContainerAutoConfiguration.EmbeddedTomcat matched:
```

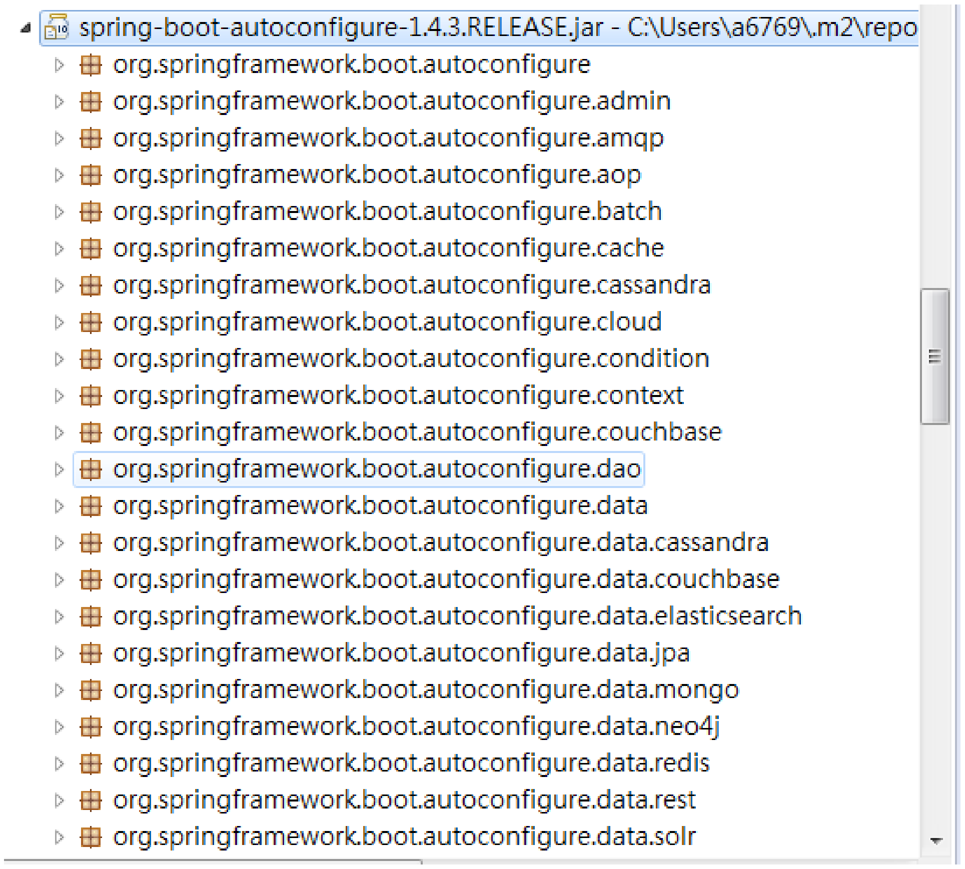
例：未啟用的自動配置：



```
Negative matches:
-----------------

   ActiveMQAutoConfiguration:
      Did not match:
         - @ConditionalOnClass did not find required classes 'javax.jms.ConnectionFactory', 'org.apache.activemq.ActiveMQConnectionFactory' (OnClassCondition)

   AopAutoConfiguration:
      Did not match:
         - @ConditionalOnClass did not find required classes 'org.aspectj.lang.annotation.Aspect', 'org.aspectj.lang.reflect.Advice' (OnClassCondition)

   ArtemisAutoConfiguration:
      Did not match:
         - @ConditionalOnClass did not find required classes 'javax.jms.ConnectionFactory', 'org.apache.activemq.artemis.jms.client.ActiveMQConnectionFactory'

   BatchAutoConfiguration:
      Did not match:
         - @ConditionalOnClass did not find required classes 'org.springframework.batch.core.launch.JobLauncher', 'org.springframework.jdbc.core.JdbcOperatior

   CacheAutoConfiguration:
      Did not match:
         - @ConditionalOnBean (types: org.springframework.cache.interceptor.CacheAspectSupport; SearchStrategy: all) did not find any beans (OnBeanCondition)
      Matched:
         - @ConditionalOnClass found required class 'org.springframework.cache.CacheManager' (OnClassCondition)

   CacheAutoConfiguration.CacheManagerJpaDependencyConfiguration:
```

查看自動配置原始碼

Spring Boot自動配置的source code是在 spring-boot-autoconfigure-<version>.jar內， 主要包含了如下圖所示， 如需知道詳細的項目可以查看這裡的source code：



spring-boot-autoconfigure-1.4.3.RELEASE.jar - C:\Users\a6769\.m2\repo
org.springframework.boot.autoconfigure
org.springframework.boot.autoconfigure.admin
org.springframework.boot.autoconfigure.amqp
org.springframework.boot.autoconfigure.aop
org.springframework.boot.autoconfigure.batch
org.springframework.boot.autoconfigure.cache
org.springframework.boot.autoconfigure.cassandra
org.springframework.boot.autoconfigure.cloud
org.springframework.boot.autoconfigure.condition
org.springframework.boot.autoconfigure.context
org.springframework.boot.autoconfigure.couchbase
org.springframework.boot.autoconfigure.dao
org.springframework.boot.autoconfigure.data
org.springframework.boot.autoconfigure.data.cassandra
org.springframework.boot.autoconfigure.data.couchbase
org.springframework.boot.autoconfigure.data.elasticsearch
org.springframework.boot.autoconfigure.data.jpa
org.springframework.boot.autoconfigure.data.mongo
org.springframework.boot.autoconfigure.data.neo4j
org.springframework.boot.autoconfigure.data.redis
org.springframework.boot.autoconfigure.data.rest
org.springframework.boot.autoconfigure.data.solr

@EnableAutoConfiguration 核心注解

在前面解釋@SpringBootApplication 注解時，有說明它其實組合了一個**@EnableAutoConfiguration** 注解，它就是自動配置的核心功能。

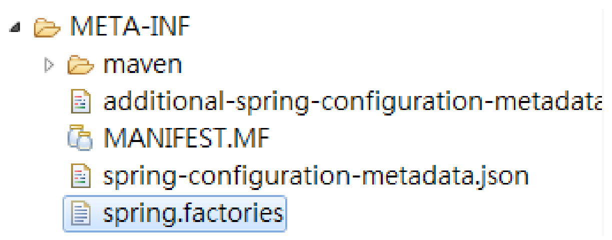
Source code如下：

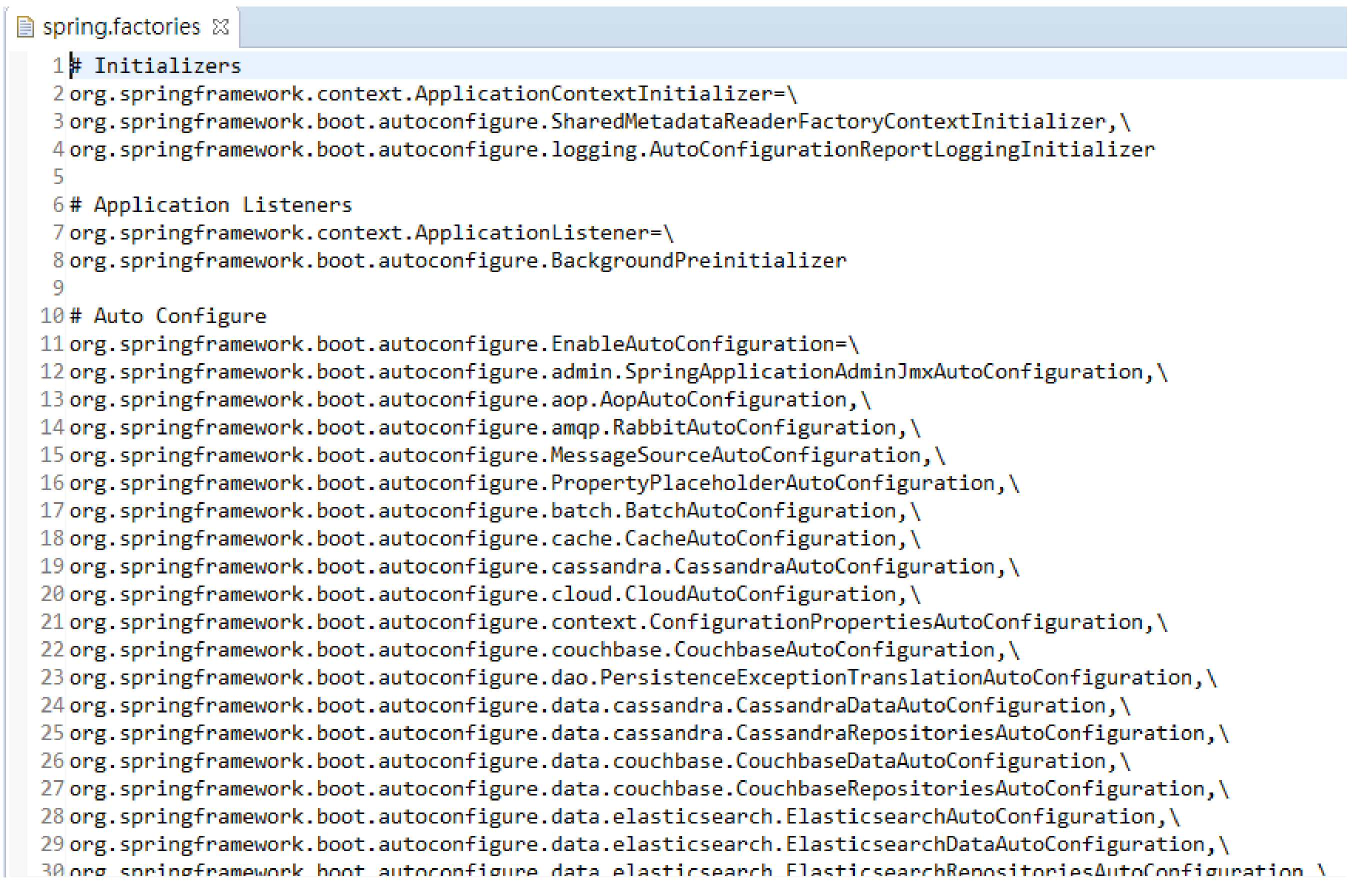
EnableAutoConfiguration

```
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Inherited
@AutoConfigurationPackage
@Import(EnableAutoConfigurationImportSelector.class)
public @interface EnableAutoConfiguration {

        String ENABLED_OVERRIDE_PROPERTY = "spring.boot.enableautoconfiguration";

        Class<?>[] exclude() default {};
        String[] excludeName() default {};

}
```

其關鍵處在於 **@Import注解指明要載入EnableAutoConfigurationImportSelector.class 配置**，它大致的功能就是說要掃描jar包裡的**META-INF/spring.factories**文件；並在該文件中尋找 **@EnableAutoConfiguration**的全路徑名稱**org.springframework.boot.autoconfigure.EnableAutoConfiguration這個**key，該key對應的value就是用於聲明都需要啟用哪些自動配置類。

在 spring-boot-autoconfigure-<version>.jar 裏就包含有一個 spring.factories 文件


```
META-INF
  maven
  additional-spring-configuration-metadata
  MANIFEST.MF
  spring-configuration-metadata.json
  spring.factories
```



```
# Initializers
org.springframework.context.ApplicationContextInitializer=\
org.springframework.boot.autoconfigure.SharedMetadataReaderFactoryContextInitializer,\
org.springframework.boot.autoconfigure.logging.AutoConfigurationReportLoggingInitializer

# Application Listeners
org.springframework.context.ApplicationListener=\
org.springframework.boot.autoconfigure.BackgroundPreinitializer

# Auto Configure
org.springframework.boot.autoconfigure.EnableAutoConfiguration=\
org.springframework.boot.autoconfigure.admin.SpringApplicationAdminJmxAutoConfiguration,\
org.springframework.boot.autoconfigure.aop.AopAutoConfiguration,\
org.springframework.boot.autoconfigure.amqp.RabbitAutoConfiguration,\
org.springframework.boot.autoconfigure.MessageSourceAutoConfiguration,\
org.springframework.boot.autoconfigure.PropertyPlaceholderAutoConfiguration,\
org.springframework.boot.autoconfigure.batch.BatchAutoConfiguration,\
org.springframework.boot.autoconfigure.cache.CacheAutoConfiguration,\
org.springframework.boot.autoconfigure.cassandra.CassandraAutoConfiguration,\
org.springframework.boot.autoconfigure.cloud.CloudAutoConfiguration,\
org.springframework.boot.autoconfigure.context.ConfigurationPropertiesAutoConfiguration,\
org.springframework.boot.autoconfigure.couchbase.CouchbaseAutoConfiguration,\
org.springframework.boot.autoconfigure.dao.PersistenceExceptionTranslationAutoConfiguration,\
org.springframework.boot.autoconfigure.data.cassandra.CassandraDataAutoConfiguration,\
org.springframework.boot.autoconfigure.data.cassandra.CassandraRepositoriesAutoConfiguration,\
org.springframework.boot.autoconfigure.data.couchbase.CouchbaseDataAutoConfiguration,\
org.springframework.boot.autoconfigure.data.couchbase.CouchbaseRepositoriesAutoConfiguration,\
org.springframework.boot.autoconfigure.data.elasticsearch.ElasticsearchAutoConfiguration,\
org.springframework.boot.autoconfigure.data.elasticsearch.ElasticsearchDataAutoConfiguration,\
org.springframework.boot.autoconfigure.data.elasticsearch.ElasticsearchRepositoriesAutoConfiguration,\
```

(有多個 configuration要啟用，以','區隔， 行尾的'\'表示接續下一行的意思)

在這文件裏我們可以得知 Spring Boot 預設會啟用哪些自動配置，如果想關閉某些自動配置，可以在@SpringBootApplication 宣告中加上 exclude屬性，如：

exclude

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

甚至要完全關閉Spring Boot 自動配置功能，可在 application.properties 中加上 spring.boot.enableautoconfiguration=false (但就失去Spring Boot 的意義了)

實例說明

在一般的web project中要設定http編碼，通常會在web.xml裡配置一個filter， 如：

filter

```
<filter>
        <filter-name>encodingFilter</filter-name>
        <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
        <init-param>
                <param-name>encoding</param-name>
                <param-value>UTF-8</param-value>
        </init-param>
        <init-param>
                <param-name>forceEncoding</param-name>
                <param-value>true</param-value>
        </init-param>
</filter>
```

但在spring-boot-starter-web中是由Spring Boot自動配置完成， 比如spring-boot-autoconfigure-1.4.3.RELEASE.jar中的 spring.factories 就聲明要加載HttpEncodingAutoConfiguration這個自動配置類。該類的souce code 如下：

HttpEncodingProperties

```
@ConfigurationProperties(prefix = "spring.http.encoding")
public class HttpEncodingProperties {

        public static final Charset DEFAULT_CHARSET = Charset.forName("UTF-8");

        private Charset charset = DEFAULT_CHARSET;

        private Boolean force;

//...............
}
```

HttpEncodingAutoConfiguration

```
@Configuration
@EnableConfigurationProperties(HttpEncodingProperties.class)  //1
@ConditionalOnWebApplication  //2
@ConditionalOnClass(CharacterEncodingFilter.class)  //3
@ConditionalOnProperty(prefix = "spring.http.encoding", value = "enabled", matchIfMissing = true) //4
public class HttpEncodingAutoConfiguration {

        private final HttpEncodingProperties properties;  //1

        public HttpEncodingAutoConfiguration(HttpEncodingProperties properties) {
                this.properties = properties;  //1
        }

        @Bean
        @ConditionalOnMissingBean(CharacterEncodingFilter.class)  //5
        public CharacterEncodingFilter characterEncodingFilter() {
                CharacterEncodingFilter filter = new OrderedCharacterEncodingFilter();
                filter.setEncoding(this.properties.getCharset().name());
                filter.setForceRequestEncoding(this.properties.shouldForce(Type.REQUEST));
                filter.setForceResponseEncoding(this.properties.shouldForce(Type.RESPONSE));
                return filter;
        }

    //...............

}
```

說明：

- @EnableConfigurationProperties：啟用屬性類注入(屬性類是一種注解了@ConfigurationProperties的JavaBean)。本例表示要將HttpEncodingProperties 建立為JavaBean， 注入到本類中的properties。

- @ConditionalOnWebApplication：指當前專案是web project的條件下才加載當前配置類
- @ConditionalOnClass：當classpath 路徑下有指定的類的條件下才加載當前配置類。本例指當CharacterEncodingFilter在classpath 的條件下
- @ConditionalOnProperty：當指定的屬性等於指定的值的情況下加載當前配置類。本例指當設定spring.http.encoding=enabled的條件下，如果沒有設定則預設為true，即符合條件
- @ConditionalOnMissingBean：指當前容器裡沒有指定的Bean的情況下。本例指當容器中沒有CharacterEncodingFilter這個Bean的時候，就建立Bean。

補充說明：

- Spring Boot 使用Conditional Annotation: @ConditionalOnXXX 的方式，來達到動態依據當前專案的環境配置來判斷是否啟動某類的自動配置
- @ConditionalOn注解除了上面介紹的4種類形外，還有其它幾種形式：
    - @ConditionalOnExpression：基於SpEL表達式作為條件判斷
    - @ConditionalOnJava：基於JAVA版本作為條件判斷
    - @ConditionalOnJndi：在JNDI存在的條件下查找指定的位置
    - @ConditionalOnMissingClass：當前classpath下沒有指定的類的條件下
    - @ConditionalOnNotWebApplication：當前專案不是web project的條件下
    - @ConditionalOnResource：classpath下是否有指定的值
    - @ConditionalOnSingleCandidate：當指定的Bean在容器中只有一個的情況下

總結

總結一下SpringBoot是如何做到自動配置的：

- @SpringBootApplication 注解本身包含了@Configuration 和@EnableAutoConfiguration，聲明它本身是一個JavaConfig 類且要啟動自動配置
- @EnableAutoConfiguration 會掃描jar包中的META-INF/spring.factories 文件，加載要初始化的各種AutoConfiguration類
- 各種AutoConfiguration類 都是加注了 @Configuration 的JavaConfig， 透過各種 @ConditionalOn 注解決定哪些Bean可以被容器初始化
- 如果希望自己建立一個自動配置類(AutoConfiguration)，只需要在我們自己建立的JavaConfig中加上@ConditionalOn注解，並且在classpath 下建立 META-INF/spring.factories，加入參數： org.springframework.boot.autoconfigure.EnableAutoConfiguration=xxxxAutoConfiguration
- 如果希望進一步了解SpringBoot的自動配置，建議查看每一個AutoConfiguration類的source code