



SpringBoot의 원리

플랫폼서비스팀

김설한

2021.07.26

- 01 소개
- 02 의존성 관리
- 03 자동 설정
- 04 내장 WAS
- 05 정리

1. SpringBoot 소개



※ 개념

제품 수준의 Spring 기반 application을 빠르고 쉽게(독립적으로) 만들 수 있게 해주는 tool.

스프링 부트 자체로 가장 널리 쓰이며, 최선이라 판단되는 스프링 설정들과 third-party 라이브러리들을 제공해준다.

1. SpringBoot 소개

※ 목적

- ① 모든 Spring 개발을 할 때, 더 빠르고 폭 넓은 사용성을 제공해 준다.
- ② 일일이 설정하지 않아도 되는 컨벤션을 제공하며, 원한다면 변경도 쉽고 빠르게 할 수 있게해 준다.
- ③ 비즈니스 로직에 필요한 기능 뿐 아닌, non-functional 기능들도 제공해준다. (내장서버, security, 외부설정 등)
- ④ Xml 설정, code generation을 더 이상 사용하지 않는다. (쉬운 커스터마이징과 명확한 사용성을 위해)
- ⑤ Java8 이상, servlet 3.1 버전 이상 부터 사용 가능하다.

SpringBoot의 어떤 원리로 이를 가능하게 할까?

2. 의존성 관리

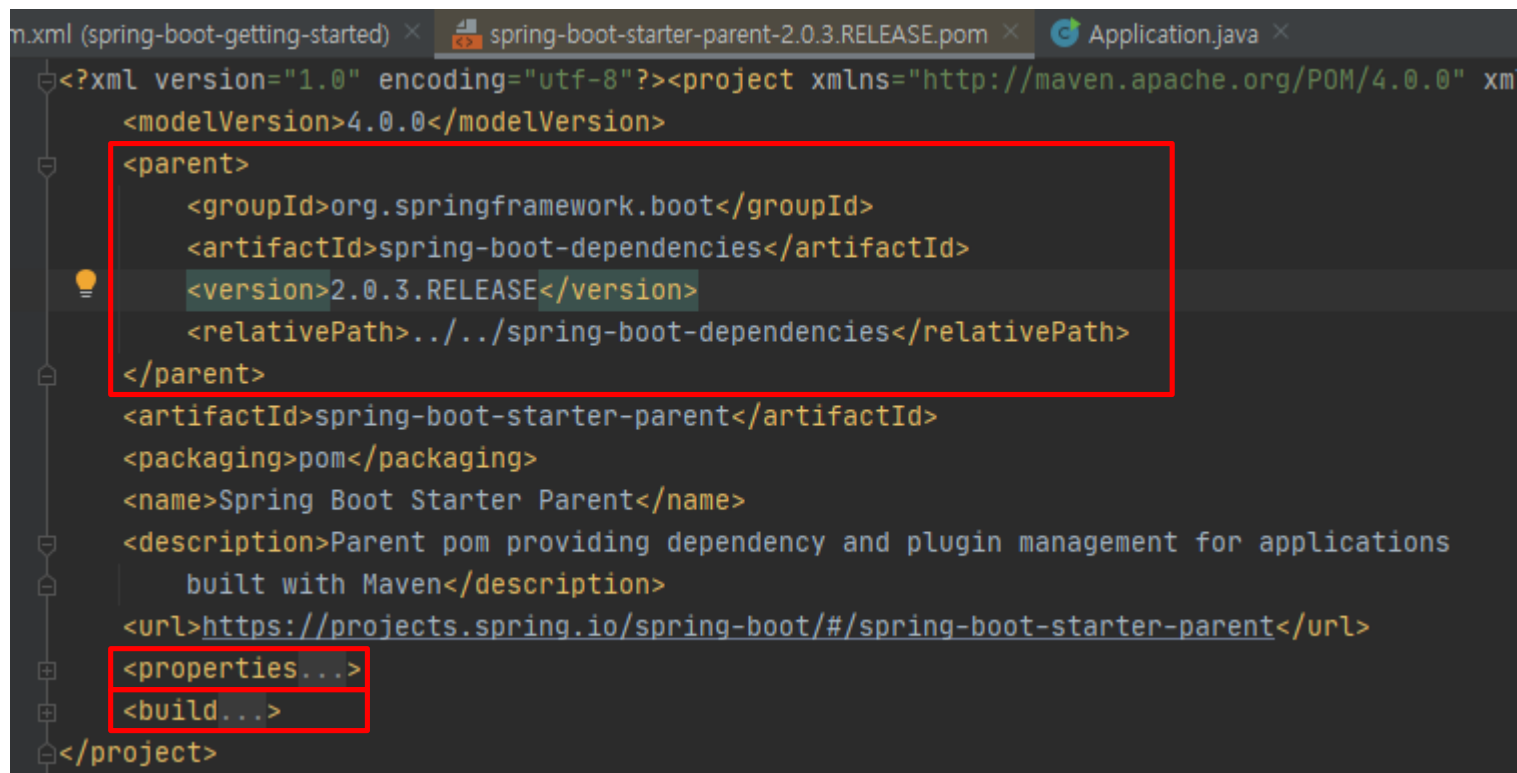
1) 원리

- SpringBoot는 여러 dependency들을 직접 등록하거나, 버전을 기입하지 않아도 자동으로 dependency들을 import된다.
- 이는, pom.xml에 parent로 등록한 spring-boot-starter-parent덕에 가능하다.

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>2.0.3.RELEASE</version>
</parent>
```

2. 의존성 관리

- pom.xml에 등록한 parent는 해당 pom.xml의 부모 역할을 하는데, spring-boot-starter-parent의 pom.xml을 살펴보면 아래와 같다.



```
<?xml version="1.0" encoding="utf-8"?><project xmlns="http://maven.apache.org/POM/4.0.0" xm
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-dependencies</artifactId>
    <version>2.0.3.RELEASE</version>
    <relativePath>../../spring-boot-dependencies</relativePath>
  </parent>
  <artifactId>spring-boot-starter-parent</artifactId>
  <packaging>pom</packaging>
  <name>Spring Boot Starter Parent</name>
  <description>Parent pom providing dependency and plugin management for applications
    built with Maven</description>
  <url>https://projects.spring.io/spring-boot/#/spring-boot-starter-parent</url>
  <properties...>
  <build...>
</project>
```

2. 의존성 관리

encoding, java version 등 설정

```
<properties>
  <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
  <java.version>1.8</java.version>
  <resource.delimiter>@</resource.delimiter>
  <maven.compiler.source>${java.version}</maven.compiler.source>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <maven.compiler.target>${java.version}</maven.compiler.target>
</properties>
```

spring-boot-starter-parent-2.0.3.RELEASE.pom

```
<?xml version="1.0" encoding="utf-8"?><project xmlns="http://maven.apache.org/POM/4.0.0"
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-dependencies</artifactId>
    <version>2.0.3.RELEASE</version>
    <relativePath>../../spring-boot-dependencies</relativePath>
  </parent>
  <artifactId>spring-boot-starter-parent</artifactId>
  <packaging>pom</packaging>
  <name>Spring Boot Starter Parent</name>
  <description>Parent pom providing dependency and plugin management for applications
    built with Maven</description>
  <url>https://projects.spring.io/spring-boot/#/spring-boot-starter-parent</url>
  <properties...>
  <build...>
</project>
```

Spring-boot-starter-parent의 parent

resource 필터

```
<resources>
  <resource>
    <filtering>true</filtering>
    <directory>${basedir}/src/main/resources</directory>
    <includes>
      <include>**/application*.yml</include>
      <include>**/application*.yaml</include>
      <include>**/application*.properties</include>
    </includes>
  </resource>
  <resource>
    <directory>${basedir}/src/main/resources</directory>
    <excludes>
      <exclude>**/application*.yml</exclude>
      <exclude>**/application*.yaml</exclude>
      <exclude>**/application*.properties</exclude>
    </excludes>
  </resource>
</resources>
```

plugin

```
<pluginManagement>
  <plugins>
    <plugin...>
    <plugin...>
    <plugin...>
    <plugin...>
    <plugin...>
    <plugin...>
    <plugin...>
    <plugin...>
    <plugin...>
    <plugin...>
  </plugins>
</pluginManagement>
```

=> SpringBoot가 정의한 Convention.

2. 의존성 관리

- spring-boot-starter-parent의 parent인 spring-boot-dependencies가 최상위 parent 인데, 이 것이 실질적으로 의존성 관리를 해준다.

```
-boot-starter-parent-2.0.3.RELEASE.pom x spring-boot-dependencies-2.0.3.RELEASE.pom x
<?xml version="1.0" encoding="utf-8"?><project xmlns="http://maven.apache.org/POM/4.0.0"
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-dependencies</artifactId>
  <version>2.0.3.RELEASE</version>
  <packaging>pom</packaging>
  <name>Spring Boot Dependencies</name>
  <description>Spring Boot Dependencies</description>
  <url>https://projects.spring.io/spring-boot/#</url>
  <licenses>
    <license>
      <name>Apache License, Version 2.0</name>
      <url>http://www.apache.org/licenses/LICENSE-2.0</url>
    </license>
  </licenses>
  <developers>
    <developer>
      <name>Pivotal</name>
      <email>info@pivotal.io</email>
      <organization>Pivotal Software, Inc.</organization>
      <organizationUrl>http://www.spring.io</organizationUrl>
    </developer>
  </developers>
  <scm>
    <url>https://github.com/spring-projects/spring-boot</url>
  </scm>
  <properties>
  </properties>
  <dependencyManagement>
  </dependencyManagement>
  <build>
  </build>
</project>
```

properties에는 버전 정보, build에는 build 정보들이...
dependencyManagement에 dependency들이 정의되어있음.

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot</artifactId>
      <version>2.0.3.RELEASE</version>
    </dependency>
  </dependencies>
</dependencyManagement>
```

⋮

```
<dependency>
  <groupId>xml-apis</groupId>
  <artifactId>xml-apis</artifactId>
  <version>${xml-apis.version}</version>
</dependency>
</dependencies>
</dependencyManagement>
```


2. 의존성 관리

∴ 내 프로젝트 pom.xml에 등록된 최상단 parent pom.xml에 의해 의존성이 관리 됨.

- parent영역을 등록하는 방법이 아닌 dependencyManagement를 직접 등록해 의존성 관리를 할 수도 있는데, 이는 spring-boot-starter-parent에 등록된 build나 plugin 관리 기능을 사용할 수 없게 된다 ∴ **parent를 사용하는 것이 권장된다.**

2. 의존성 관리

2) 응용

① springboot가 관리해주는 dependency

- 기본 사용 : version 명시하지 않고, dependency만 기입. (intelliJ의 경우 왼쪽에 관리여부)
- 버전 변경 : properties영역에 버전 정보 기입. (Java.version 등 속성 변경도 동일)

```
<properties>  
    <java.version>1.7</java.version>  
    <spring.version>5.0.6.RELEASE</spring.version>  
</properties>
```

②springboot가 관리하지 않는 dependency

- dependency를 추가할 때, version정보도 함께 기재. (Default가 있으나, 명시해주는것이 좋음)

3. 자동 설정

원리

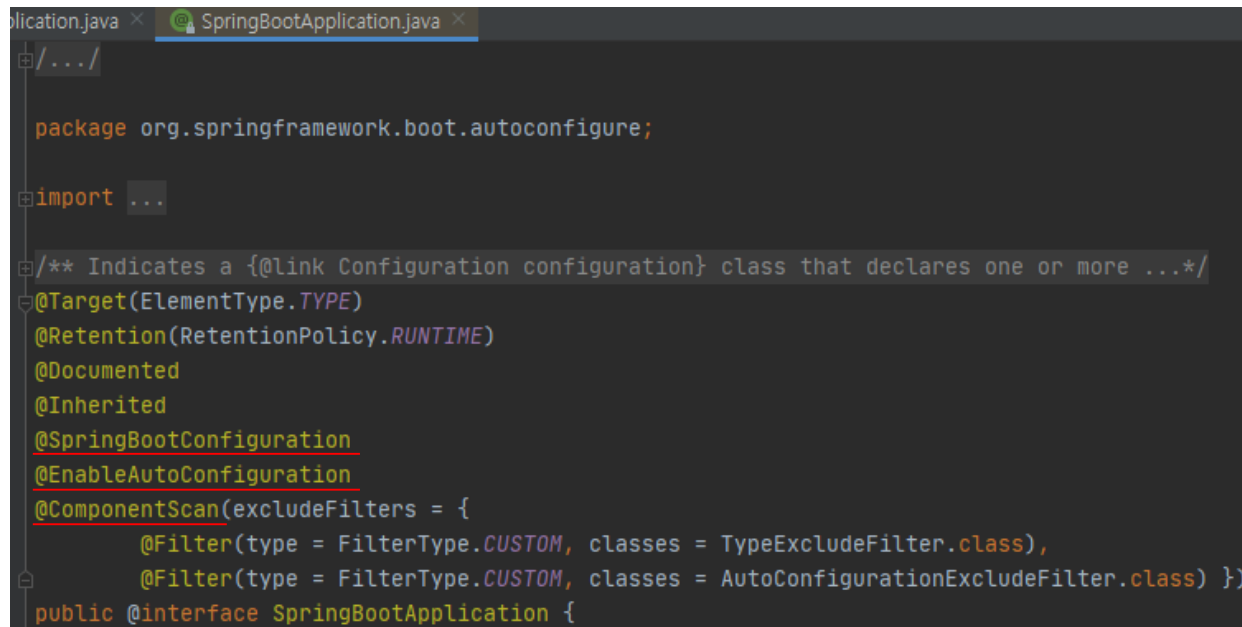
- Springboot는 annotation 기반 자동 bean 등록, 자동 설정 기능을 제공하는데 이는 Application에 등록하는 @SpringBootApplication 으로 가능케된다.

```
Application.java x
1  package me.whiteship;
2
3  import org.springframework.boot.SpringApplication;
4  import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6  @SpringBootApplication
7  public class Application {
8
9      public static void main(String[] args) { SpringApplication.run(Application.class, args); }
12 }
```

3. 자동 설정

@SpringBootApplication

= @SpringBootConfiguration + @EnableAutoConfiguration + @ComponentScan



```
package org.springframework.boot.autoconfigure;

import ...

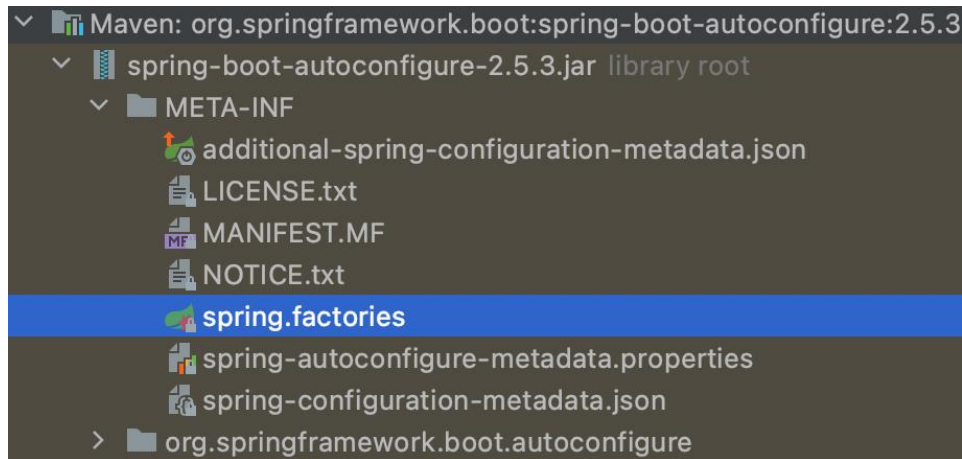
/** Indicates a {@link Configuration configuration} class that declares one or more ...*/
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Inherited
@SpringBootConfiguration
@EnableAutoConfiguration
@ComponentScan(excludeFilters = {
    @Filter(type = FilterType.CUSTOM, classes = TypeExcludeFilter.class),
    @Filter(type = FilterType.CUSTOM, classes = AutoConfigurationExcludeFilter.class) })
public @interface SpringBootApplication {
```

Springboot는 bean을 2번 등록하는데,

- @ComponentScan : @Component가 붙은 class들을 bean으로 등록.
- @EnableAutoConfiguration : jar파일에 정의된 configuration들을 bean으로 등록

3. 자동 설정

@EnableAutoConfiguration은 spring meta file을 읽어 config정보를 등록한다.



각 configuration파일들은
@configuration 애노테이션을
달고 있어, bean으로 등록 되는
원리.



```
org.springframework.boot.autoconfigure.EnableAutoConfiguration=\norg.springframework.boot.autoconfigure.admin.SpringApplicationAdminJmxAutoConfiguration,\norg.springframework.boot.autoconfigure.aop.AopAutoConfiguration,\norg.springframework.boot.autoconfigure.amqp.RabbitAutoConfiguration,\norg.springframework.boot.autoconfigure.batch.BatchAutoConfiguration,\norg.springframework.boot.autoconfigure.cache.CacheAutoConfiguration,\norg.springframework.boot.autoconfigure.cassandra.CassandraAutoConfiguration,\norg.springframework.boot.autoconfigure.cloud.CloudAutoConfiguration,\norg.springframework.boot.autoconfigure.context.ConfigurationPropertiesAutoConfiguration,\norg.springframework.boot.autoconfigure.context.MessageSourceAutoConfiguration,\norg.springframework.boot.autoconfigure.context.PropertyPlaceholderAutoConfiguration,\norg.springframework.boot.autoconfigure.couchbase.CouchbaseAutoConfiguration,\norg.springframework.boot.autoconfigure.dao.PersistenceExceptionTranslationAutoConfiguration,\norg.springframework.boot.autoconfigure.data.cassandra.CassandraDataAutoConfiguration,\norg.springframework.boot.autoconfigure.data.cassandra.CassandraReactiveDataAutoConfiguration,\norg.springframework.boot.autoconfigure.data.cassandra.CassandraReactiveRepositoriesAutoConfiguration,\n
```

:

```
org.springframework.boot.autoconfigure.web.reactive.error.ErrorWebFluxAutoConfiguration,\norg.springframework.boot.autoconfigure.web.reactive.function.client.WebClientAutoConfiguration,\norg.springframework.boot.autoconfigure.web.servlet.DispatcherServletAutoConfiguration,\norg.springframework.boot.autoconfigure.web.servlet.ServletWebServerFactoryAutoConfiguration,\norg.springframework.boot.autoconfigure.web.servlet.error.ErrorMvcAutoConfiguration,\norg.springframework.boot.autoconfigure.web.servlet.HttpEncodingAutoConfiguration,\norg.springframework.boot.autoconfigure.web.servlet.MultipartAutoConfiguration,\norg.springframework.boot.autoconfigure.web.servlet.WebMvcAutoConfiguration,\norg.springframework.boot.autoconfigure.websocket.reactive.WebSocketReactiveAutoConfiguration,\norg.springframework.boot.autoconfigure.websocket.servlet.WebSocketServletAutoConfiguration,\norg.springframework.boot.autoconfigure.websocket.servlet.WebSocketMessagingAutoConfiguration,\norg.springframework.boot.autoconfigure.webservices.WebServicesAutoConfiguration
```

4. 내장 WAS

Springboot를 사용하면, 내장 WAS가 import된다.

```
org.springframework.boot.autoconfigure.web.servlet.ServletWebServerFactoryAutoConfiguration,
```

```
> Maven: org.apache.tomcat.embed:tomcat-embed-core:9.0.50
```

```
> Maven: org.apache.tomcat.embed:tomcat-embed-el:9.0.50
```

```
> Maven: org.apache.tomcat.embed:tomcat-embed-websocket:9.0.50
```

- 내장 WAS 자동 설정은 앞서 설명했던, 자동 설정(@EnableAutoconfiguration)의 일부.
- Servlet container는 spring.factories에 기입된,
ServletWebServerFactoryAutoConfiguration에 의해 등록된다.

```
@Configuration(  
    proxyBeanMethods = false  
)  
@AutoConfigureOrder(-2147483648)  
@ConditionalOnClass({ServletRequest.class})  
@ConditionalOnWebApplication(  
    type = Type.SERVLET  
)  
@EnableConfigurationProperties({ServerProperties.class})  
@Import({ServletWebServerFactoryAutoConfiguration.BeanPostProcessorsRegistrar.class, EmbeddedTomcat.class, EmbeddedJetty.class, EmbeddedUndertow.class})  
public class ServletWebServerFactoryAutoConfiguration {
```

4. 내장 WAS

- 서블릿을 만들고 등록하는 역할은 DispatcherServletAutoConfiguration에 의해 이루어진다.

`org.springframework.boot.autoconfigure.web.servlet.DispatcherServletAutoConfiguration`

- dispatcherServlet : HttpServlet을 상속해서 만든 Spring MVC의 클래스.

```
@Bean(  
    name = {"dispatcherServlet"}  
)  
public DispatcherServlet dispatcherServlet(WebMvcProperties webMvcProperties) {  
    DispatcherServlet dispatcherServlet = new DispatcherServlet();  
    dispatcherServlet.setDispatchOptionsRequest(webMvcProperties.isDispatchOptionsRequest());  
    dispatcherServlet.setDispatchTraceRequest(webMvcProperties.isDispatchTraceRequest());  
    dispatcherServlet.setThrowExceptionIfNoHandlerFound(webMvcProperties.isThrowExceptionIfNoHandlerFound());  
    dispatcherServlet.setPublishEvents(webMvcProperties.isPublishRequestHandledEvents());  
    dispatcherServlet.setEnableLoggingRequestDetails(webMvcProperties.isLogRequestDetails());  
    return dispatcherServlet;  
}
```

이처럼, 해당 configuration class안에서 dispatcherServlet을 생성하고,
Servlet 컨테이너에 등록하는 일련의 작업들이 이루어진다.

WAS와 sevlet configuration이 분리되어있는 이유

: servlet컨테이너는 변경될 수 있지만, servlet은 변하지 않기 때문

4. 내장 WAS

2) 응용

– WAS 변경

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

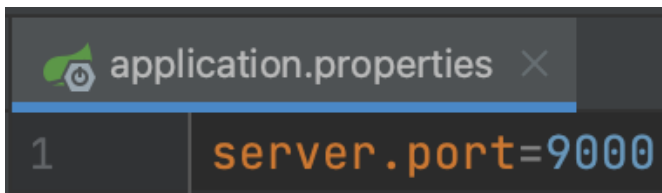
```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-jetty</artifactId>
</dependency>
```


4. 내장 웹 서버

2) 응용

- 포트 변경

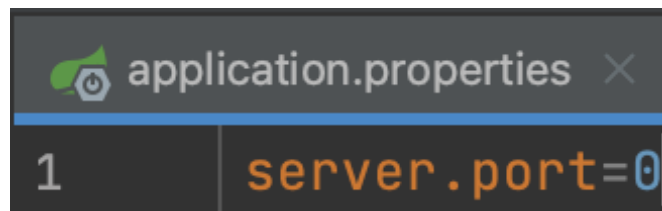
application.properties 파일에 변경할 port번호를 입력해준다.



```
application.properties ×  
1 server.port=9000
```

```
Tomcat started on port(s): 9000 (http) with context path ''
```

(+) server port를 0으로 설정 : 사용할 수 있는 포트번호를 자동으로 찾아 띄워주는 random port.



```
application.properties ×  
1 server.port=0
```

```
Tomcat started on port(s): 50062 (http) with context path ''
```

5. 정리

- 1) 의존성 관리 : spring boot start에 의해 가능. parent로 등록된 spring-boot-starter, 그 parent인 spring-boot-dependencies에 의해 주요 라이브러리들과 그 버전들이 관리 됨.
- 2) 자동 설정 : @SpringBootApplication의 @ComponentScan, @EnableAutoConfiguration에 의해 동작. componentScan으로 등록된 bean들을 기반으로 condition을 판단해, autoConfig.
- 3) 내장 WAS : spring boot의 주요 goal 중 하나인 stand alone(독립실행) application 제공. 이 또한 자동 설정으로 제공되는 기능인데, 필요에 따른 커스텀이 가능하다.



Thank you