Spring DI와 AOP

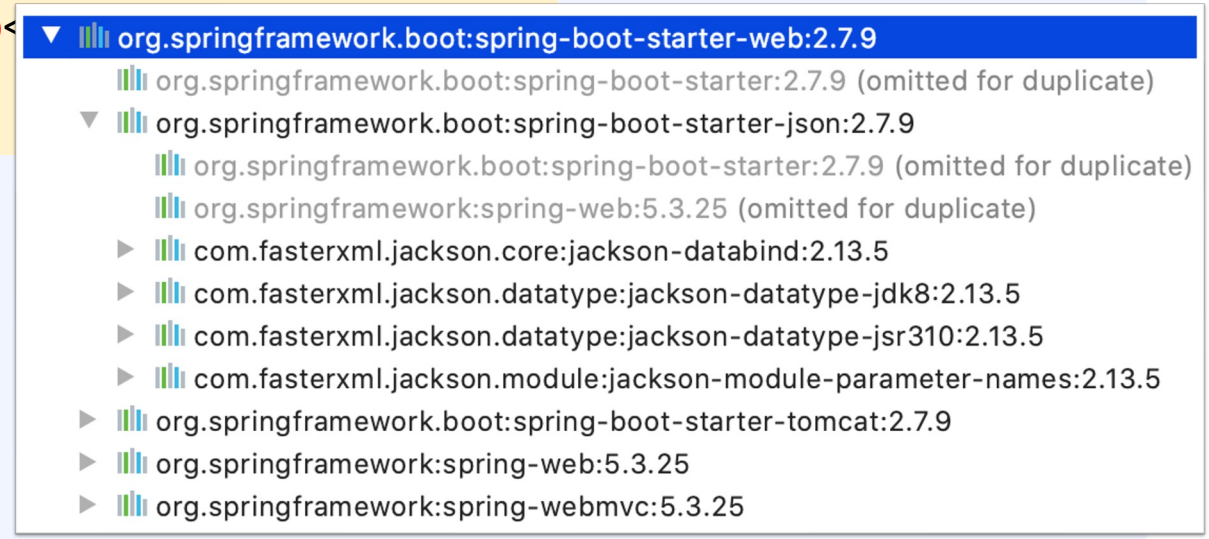
6 의존성 관리와 설정의 자동화

의존성 관리와 설정의 자동화
1. 스타터(starter)란?

여러 관련 라이브러리를 묶어서 패키지로 제공
starter만 pom.xml에 추가하면, 관련 라이브러리가 자동으로 추가됨.

[pom.xml]

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



의존성 관리와 설정의 자동화
2. pom.xml이란?

Maven기반 프로젝트의 설정 파일
프로젝트 기본 설정 정보, 의존 라이브러리, 설정 상속 정보를 지정.

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.7.9</version>
    <relativePath/> <!-- lookup p
</parent>
```

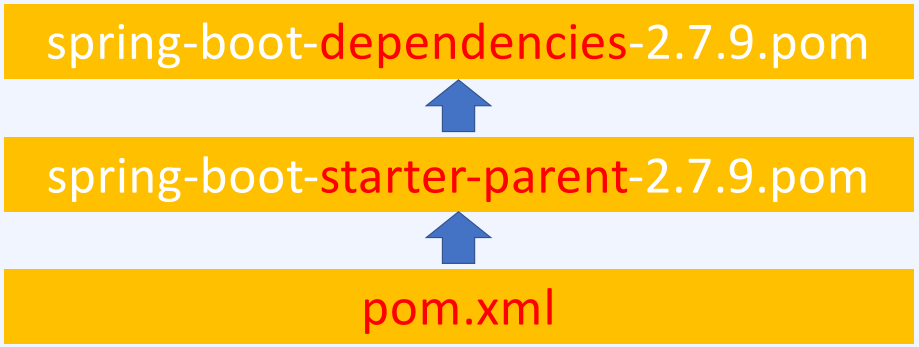
```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>${org.springframework-version}</version>
</dependency>
```

```
<properties>
    <java.version>1.8</java.version>
    <org.springframework-version>5.0.2.RELEASE</org.springframework-version>
    <org.aspectj-version>1.6.10</org.aspectj-version>
</properties>
```

의존성 관리와 설정의 자동화
3. pom파일의 상속

<parent>로 pom.xml파일 간의 상속이 가능하며, 자손에서 덮어쓰기 가능

```
<parent>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-parent</artifactId>
   <version>2.7.9</version>
   <relativePath/> <!-- lookup parent from repository -->
</parent>
```



의존성 관리와 설정의 자동화
4. 자동 의존성 변경하기 – 덮어쓰기와 제외하기

starter에 포함된 특정 라이브러리를 제외하는 것이 가능
특정 라이브러리의 버전을 변경하는 것도 가능

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-web</artifactId>
   <exclusions>
      <exclusion>
         <groupid>com.fasterxml.jackson.core</groupId>
      </exclusion>
   </exclusions>
</dependency>
```

```
<dependency>
   <groupId>com.fasterxml.jackson.core</groupId>
   <artifactId>jackson-databind</artifactId>
   <version>2.14.2</version>
</dependency>
```

의존성 관리와 설정의 자동화

5. @SpringBootApplication

Spring Boot Application의 시작 클래스에 붙인다.

@SpringBootApplication = @Configuration + @EnableAutoConfiguration + @ComponentScan

```java
@SpringBootApplication
public class Main {
    public static void main(String[] args) {
        ApplicationContext ac = SpringApplication.run(Main.class, args);
        String[] beanDefinitionNames = ac.getBeanDefinitionNames();
        Arrays.sort(beanDefinitionNames);

        Arrays.stream(beanDefinitionNames).forEach(System.out::println);
    }
}
```

의존성 관리와 설정의 자동화

5. @SpringBootApplication

- @SpringBootApplication
  - @SpringBootConfiguration -> @Configuration
  - @ComponentScan
  - @EnableAutoConfiguration
    - @AutoConfigurationPackage -> @Import({Registrar.class})
    - @Import({AutoConfigurationImportSelector.**class**})

Spring DI와 AOP

8 @Conditional과 @Import

의존성 관리와 설정의 자동화
6. Condition과 @Conditional

조건에 따라 빈의 등록 여부를 결정. @Bean, @Component와 같이 사용

Condition의 matches()를 구현한 클래스를 @Conditional로 지정

```
class FalseCondition implements Condition {
    public boolean matches(ConditionContext context, AnnotatedTypeMetadata metadata) {
        return false;
    }
}
class TrueCondition implements Condition {
    public boolean matches(ConditionContext context, AnnotatedTypeMetadata metadata) {
        return true;
    }
}
```

```
@Component
@Conditional(FalseCondition.class)
class Engine { public String toString() { return "Engine{}"; } }

@Component
@Conditional(TrueCondition.class)
class Door  { public String toString() { return "Door{}"; } }
```

의존성 관리와 설정의 자동화
6. Condition과 @Conditional

그 밖에 여러 조건부로 빈을 등록하는 여러 애너테이션을 제공

조건부 빈 등록 애너테이션	설 명
@ConditionalOnBean	지정한 빈들이 이미 등록되어 있을 때
@ConditionalOnClass	지정된 클래스들이 classpath에 있을 때
@ConditionalOnMissingBean	지정한 빈들이 아직 등록되어 있지 않을 때
@ConditionalOnMissingClass	지정된 클래스들이 classpath에 있지 않을 때
@ConditionalOnProperty	지정된 property가 지정된 value와 일치할 때
@ConditionalOnResource	지정된 리소스가 classpath에 있을 때
@ConditionalOnJava	지정된 JVM 버전에서 application이 실행 중 일 때
@ConditioanlOnWebApplication	웹 애플리케이션일 때
@ConditioanlOnNotWebApplication	웹 애플리케이션이 아닐 때

```
@EnableMyAutoConfiguration("test")
class MainConfig { @Bean Car car() { return new Car(); } }
class Config1 { @Bean Car sportsCar() { return new SportsCar(); } }
class Config2 { @Bean Car sportsCar() { return new SportsCar2(); } }
```

조건에 따라 다른 Configuration을 적용할 때 사용

ImportSelector를 구현하고 이를 @Import하는 애너테이션을 작성해서 적용

```
class MyImportSelector implements ImportSelector {
    public String[] selectImports(AnnotationMetadata importingClassMetadata) {
        AnnotationAttributes attr = AnnotationAttributes.fromMap(importingClassMetadata
            .getAnnotationAttributes(EnableMyAutoConfiguration.class.getName(), classValuesAsString: false));

        String value = attr.getString( attributeName: "value");
        if ("test".equals(value))
            return new String[]{Config1.class.getName()};
        else
            return new String[]{Config2.class.getName()};
    }
}
```

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)
@Import(MyImportSelector.class)
@interface EnableMyAutoConfiguration {
    String value() default "";
}
```

Spring DI와 AOP

9 외부 설정 사용하기

의존성 관리와 설정의 자동화
8. 외부 설정 파일 – application.properties

애플리케이션 속성의 기본 값을 바꿀 수 있는 설정 파일.
16개의 분류. 약 1700개의 속성을 제공

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

[src/main/resources/application.properties]

```
spring.main.web-application-type=none
spring.main.banner-mode=off
spring.main.allow-bean-definition-overriding=true

server.servlet.context-path=/ch2
server.servlet.encoding.enabled=true
server.servlet.encoding.charset=utf-8

# thymeleaf template 경로
spring.thymeleaf.prefix=classpath:/templates/
spring.thymeleaf.suffix=.html
```

의존성 관리와 설정의 자동화
9. 사용자 정의 속성 추가하기 - @ConfigurationProperties

```java
@ConfigurationProperties(prefix="mysite")
public class MyProperties {
    private String domain; // mysite.domain
    private String email;  // mysite.email

    public String getDomain() {
        return domain;
    }

    public void setDomain(String domain) {
        this.domain = domain;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }
}
```

[src/main/resources/application.properties]

```
mysite.domain=www.fastcampus.com
mysite.email=admin@fastcampus.com
```

```java
@EnableConfigurationProperties({MyProperties.class})
public class Main {
    public static void main(String[] args) {
        ApplicationContext ac = SpringApplication.run(Main.class, args);
        MyProperties prop = (MyProperties)ac.getBean(MyProperties.class);

        System.out.println("prop.getDomain() = " + prop.getDomain());
        System.out.println("prop.getEmail() = " + prop.getEmail());
    }
}
```

의존성 관리와 설정의 자동화

10. @Value와 @PropertySource

```
@Component
@PropertySource("setting.properties")
class SysInfo {
    @Value("#{systemProperties['user.timezone']}")
    String timeZone;

    @Value("#{systemEnvironment['PWD']}")
    String currDir;

    @Value("${autosaveDir}")
    String autosaveDir;

    @Value("${autosaveInterval}")
    int autosaveInterval;

    @Value("${autosave}")
    boolean autosave;
```

[src/main/resources/setting.properties]

```
autosaveDir=/autosave
autosave=true
autosaveInterval=30
```

```
Map<String, String> env = System.getenv();
System.out.println("System.getenv() = " + env);

Properties prop = System.getProperties();
System.out.println("System.getProperties() = " + prop);

System.out.println("ac.getBean(SysInfo.class) = " + ac.getBean(SysInfo.class));
```