< 의존성 관리 >

<artifactId>spring-boot-starter-dependencies<artifactId>
- <artifactId>spring-boot-starter-parent</artifactId> 상위 프로젝트
- 가장 상위 프로젝트
- <dependencyManagement> 영역에서 의존성 관리(버젼 관리)
- 장점
    - 우리가 직접관리해야할 의존성이 줄어든다

스프링 부트 관리 2가지 방법
1) 자신의 프로젝트에 <parent>로 spring-boot-starter-parent를 선언하여 설정
2) <dependencyManagement> 엘리먼트를 사용하여 dependency 주입
    - 의존성 관리외 다른 여러 설정 적용 X(자바설정, 인코딩설정)

<자동설정>

@SpringBootApplication
= @SpringBootConfiguration + @ComponentScan + @EnableAutoConfiguration

스프링부트에서 Bean을 등록하는 단계
1단계 : @ComponentScan
- @Componet를 가진 클래스들을 스캔해서 빈으로 등록
- @Configuration / @Repository / @Service / @Controller / @RestController
- 하위 패키지 까지 모두 찾는다

2단계 :@EnableAutoConfiguration
- spring-boot-autoconfigure > META-INF > spring.factories 안에 org.springframework.boot.autoconfigure.EnableAutoConfiguration 라는 키값 아래 AutoConfiguration들이 정의되어있다 (설정 파일들)
- 정의되있는 설정들은 조건에 따라 설정이 된다

< 자동 설정 만들기>

- Xxx-Spring-Boot-Autoconfigure 모듈 : 자동 설정
- Xxx-Spring-Boot-Starter 모듈: 필요한 의존성 정의
- 그냥 하나로 만들고 싶을 때는? = Xxx-Spring-Boot-Starter

구현 방법

1) 의존성 추가

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-autoconfigure</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-autoconfigure-processor</artifactId>
        <optional>true</optional>
    </dependency>
</dependencies>

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

2) @Configuration 파일 작성
3) src/main/resource/META-INF에 spring.factories 파일 만들기
4) spring.factories 안에 자동 설정 파일 추가

@ConditionalOnMissingBean

- 덮어쓰기 방지

빈 재정의 수고 덜기

1) 의존성 추가

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-configuration-processor</artifactId>
    <optional>true</optional>
</dependency>
```

2) @ConfigurationProperties(“AAAA”)
3) @EnableConfigurationProperetiers(AAAA.class)
4) 프로퍼티 키값 자동 완성

< 내장 웹 서버>

- 스프링부트는 서버가 아님
    - 톰캣 객체 생성
    - 포트 설정
    - 톰캣에 컨텍스트 추가
    - 서블릿만들기
    - 톰캣에 서블릿 추가
    - 컨텍스트에 서블릿 맵핑
    - 톰캣 실행 및 대기

- 이 모든 과정 보다 상세하고 유연하게 설정하고 실행해주는 것이 자동설정
    - ServletWebServerFactoryAutoConfiguration(서블릿 웹 서버 생성)
        - TomcatServletWebServerFactoryCustomizer(서버 커스터마이징)
    - DispatcherServletAutoConfiguration
        - 서블릿 만들고 등록
    - 2개가 떨어져 등록돼 있음
        - 서블릿컨테이너들은 다 달라질 수 있지만 서블릿은 달리지지 않음

다른 서블릿 컨테이너로 변경
- <spring-boot-starter-web>에서 <spring-boot-starter-tomcat>을 exclusion하고 원하는 서버를 의존성 추가

웹서버 사용하지 않기
- 프로퍼티에 spring.main.web-application-type=none

포트
- 원하는 포트 설정 server.port = A
- 랜덤 포트 server.port=0
- ApplicationListner <ServletWebServerInitializedEvent>

<jar>
- 독립적으로 실행 가능
- mvn package를 하면 실행 가능한 JAR파일 하나가 생성 됨
- spring-maven-plugin이 해주는일 (패키징)
- 과거 “uber” jar 를 사용
    - 모든 클래스(의존성 및 애플리케이션)를 하나로 압축하는 방법
    - 무엇이 어디에서 온건지 알 수 없음
- 스프링 부트의 전략
    - 내장 JAR : 기본적으로 자바에는 내장JAR를 로딩하는 표준방법 X
    - 애플리케이션 클래스와 라이브러리 위치 구분
    - org.springframework.boot.loader.jar.JarFile을 사용해서 내장 JAR를 읽는다
    - org.springframework.boot.loader.Launcher를 사용해서 실행한다

<springApplication>
- 기본 로그 레벨은 INFO
- FailureAnalyzer
    - 오류 출력을 이쁘게
- 배너
    - 리소스 패키지 아래에 banner.txt | gif | jpg | png 파일을 생성
    - classpath 또는 spring.banner.location
    - ${spring-boot.version} 등의 변수를 사용할 수 있음.
    - Banner 클래스 구현하고 SpringApplication.setBanner()로 설정 가능.
    - SpringApplicationBuilder로 빌더 패턴 사용 가능
    - 배너 끄는 방법
        - app.setBannerMode(Banner.mode.OFF)
- ApplicationEvent 등록
    - ApplicationContext를 만들기 전에 사용하는 리스너는 빈으로 등록할 수 없다
        - SpringApplication.addListners()

- WebApplicationType 설정
    - SERVLET / REACTIVE / NONE
- 애플리케이션 아규먼트 사용하기
    - ApplicationArguments를 빈으로 등록해 주니까 가져다 쓰면 됨
    - -- 옵션
- 애플리케이션 실행한 뒤 뭔가 실행하고 싶을 때
    - ApplicationRunner (추천) 또는 CommandLineRunner
    - 순서 지정 가능 @Order
        - 숫자가 낮을 수록 우선순위가 높음

< 외부설정>
● properties
● YAML
● 환경 변수
● 커맨드 라인 아규먼트

프로퍼티 우선순위
1. 유저 홈 디렉토리에 있는 spring-boot-dev-tools.properties
2. 테스트에 있는 @TestPropertySource
3. @SpringBootTest 애노테이션의 properties 애트리뷰트
4. 커맨드 라인 아규먼트
5. SPRING_APPLICATION_JSON (환경 변수 또는 시스템 프로티) 에 들어있는 프로퍼티
6. ServletConfig 파라미터
7. ServletContext 파라미터
8. java:comp/env JNDI 애트리뷰트
9. System.getProperties() 자바 시스템 프로퍼티
10. OS 환경 변수
11. RandomValuePropertySource
12. JAR 밖에 있는 특정 프로파일용 application properties
13. JAR 안에 있는 특정 프로파일용 application properties
14. JAR 밖에 있는 application properties
15. JAR 안에 있는 application properties
16. @PropertySource
17. 기본 프로퍼티 (SpringApplication.setDefaultProperties)

application.properties 우선 순위
1. file:./config/
2. file:./
3. classpath:/config/
4. classpath:/

프로퍼티 랜던값
● ${random.*}

플레이스 홀더
● name = YoungSung
● fullName = ${name} Ko

#타입-세이프 프로퍼티 @ConfigurationProperties(“AAA”)

- 여러 프로퍼티를 묶어서 읽어올 수 있음
- 빈으로 등록해서 다른 빈에 주입할 수 있음
    - @EnableConfigurationProperties
    - @Component
    - @Bean
- 융통성 있는 바인딩
    - context-path (케밥)
    - context_path (언드스코어)
    - contextPath (캐멀)
    - CONTEXTPATH
- 프로퍼티 타입 컨버전
    - @DurationUnit
- 프로퍼티 값 검증
    - @Validated
    - JSR-303 (@NotNull, ...)
- 메타 정보 생성
- @Value
    - SpEL 을 사용할 수 있지만 위에 있는 기능들은 전부 사용 X

<프로파일>

- 프로퍼티에 활성화할 프로퍼티 추가
    - spring.profiles.active=aaa
- 프로파일 추가
    - spring.profiles.include
- 프로파일용 프로퍼티
    - application-{profile}.properties

<스프링 부터 2.1>

의존성 변경

- 스프링 프레임워크 5.0 -> 5.1
    - 로거 설정 개선 spring-jcl
    - 컴포넌트 인덱스
        - 컴포넌트 스캐닝 성능 개선이 가능
    - 함수형 프로그래밍 스타일 지원
    - 코틀린 지원
    - 리액티브 프로글매이 모델 지원
    - JUnit5
- JUnit4->5
    - Jupiter
    - extension모델
    - 람다 지원
- 톰캣 8.5 -> 9
    - BIO 커넥터 -> NIO 커넥터

        - 블로킹 IO / 넌블로킹 IO
    - HTTP/2지원
    - 웹소켓2
    - 서블릿4.0/JSP2.4
- 하이버네이트 5.2->5.3
    - JPA2.2 지원
    - JAVA 8 Date/Time API

<로거>
- Logging Facade
    - SLF4J (현재 거의 이거 씀)
    - JCL
- 로거
    - JUL
    - Log4j2 (main)
    - Log4j
    - Logback (main)

문제
1) 기존에 이미 다른 로깅 퍼사드나 로걸르 사용중인 프로젝트
    - SLF4J로 통하는 다리(Bridge)를 놓는다
        - ex) JCL-over-SLFJ4
              Log4j-to-SLF4J
2) SLF4J가 사용할 로거는 어떻게 정하는가
    - Binder 사용
        - ex) logback-classic

Spring-jcl
- JCL-over-SLF4J 대체제
- 클래스패스에 Log4J 2가 있다면 JCL을 사용한 코드가 Log4j 2를 사용
- 클래스패스에 SLF4J가 있다면 JCL을 사용한 코드가 SLF4j를 사용한다
- Log4J 2 를 사용할 때는 별다른 브릿지나 바인더가 필요없다
- SLF4J를 사용할 때에도 JCL을 굳이 exclusion하거나 JCL용 브릿지를 추가할 필요 없다

설정하기
- <spring-boot-starter>(<spring-boot-starter-logging>을 exclusion) 의존성 추가
- <spring-boot-starter-log4j2> 의존성 추가

<빈 오버라이딩>
1) 애플리케이션에 정의한 빈 등록
2) 자동설정이 제공하는 빈 등록
- 이때 1번에서 정의한 빈을 2번과정에 등록하는 빈이 오버라이딩 할 수도 있었는데 2.1 이후로는 허용하지 않는다 -> 오류 발생
- 프로퍼티를 변경해서 빈 오버라이딩을 허용 할 수도 있다
    - spring.main.allow-bean-definition-overriding=true
- 오버라이딩이 일어나지 않도록 자동설정 제공하는 쪽에 @Condition* 애노테이션 활용

< 자동설정 지원>
스프링 부트 2.1부터 지원하는 자동설정

1) 태스크 실행
    - @EnableAsync 사용 시 자동설정(TaskExecutionAutoConfiguration) 적용
    - spring.task.execution 프로퍼티로 제공
    - TaskExecutorBuilder 제공
2) 태스크 스케줄링
    - @EnableScheduling 사용 시 자동설정(TaskSchedulingAutoConfiguration)
    - 주기적으로 오퍼레이션 실행
    - spring.task.scheduling 프로퍼티 제공
    - TaskSchedulerBuilder 제공
3) 스프링 데이터 JDBC
    - spring-boot-starter-data-jdbc 의존성 추가시 지원
4) 기타
    - 카프카 스트림 지원
    - JMS ConnectionFactory 지원
    - 엘라스틱 서치 REST 클라이언트 지원 등

<프로퍼티 변경>

스프링 데이터 JPA 부트스트랩 모드 지원
- 애플리케이션 구동 시간을 줄이기 위해 스프링 데이터 JPA 리파지토리 생성을 지연시키는 설정
- spring.data.jpa.repositories.bootstrap-mode= ?
    1) DEFERRED : 애플리케이션 구동 이후에 리파지토리 인스턴스를 만들어 주입
    2) LAZY : 구동 이후에도 만들지 않다가 처음 사용하는 시점에 만들어 주입

프로퍼티 마이그레이션
- spring-boot-properties-migrator 의존성 추가
- 프로퍼티를 마이그레이션 하지 않더라도 기존 프로퍼티로 애플리케이션 구동이 가능하며 프로퍼티가 어떻게 바뀌었는지 알려주는 툴

<JUnit>

- 2.1부터 지원하지만 기본적으로 들어오는건 2.2부터
- @SpringBootTest 위에 @ExtensdWith 메타 어노테이션 -> @Runwith() 생략
- 모든 API가 Jupiter 에 포함

<DataSize>
- org.springframework.util.unit.DataSize
    - 스프링 부트가 아니라 스프링 프레임워크가 5.1부터 지원하는 타입
    - 지원하는 타입 : B ,KB, MB, GB, TB
- 스프링 부트는 컨버터를 지원
    - StringToDataSizeConverter
    - NumberToDataSizeconverter
- application.properties에서 데이터 사이즈를 손쉽게 바인딩 받기 가능

<로그 그룹>

- 같은 로그 레벨을 적용할 패키지 묶음을 만들 수 있는 기능으로 여러 패키지의 로그 레벨을 손쉽게 변경할 수 있다
- 로그 그룹 정희나는 방법
    - logging.group.{그룹이름}={패키지},{패키지},{패키지}...
    - logging.level.{그룹이름}={로그레벨}
- 스프링 부트가 미리 정의해둔 로그 그룹
    - web = 스프링 웹 MVC 관련 패키지 그룹
    - sql = 스프링 JDBC 와 하이버네이트 SQL을 묶어둔 로그 그룹

<Actuator>

- spring-boot-starter-actuator 의존성 추가
- /info와 /health 엔드포인트가 스프링 시큐리티를 추가하더라도 기본적으로 '공개' 하도록 변경됨
    - 스프링부트 2.0.* 에서는 스프링 시큐리티를 추가하면 모든 엔드포인트가 인증을 거쳐야 했음
        - 스프링 시큐리티 설정 추가해서 컨트롤 가능
- /info 엔드포인트에 정보 추가하는 방법
    1) Info 키값에 들어있는 모든 프로퍼티
    2) git.properties에 들어있는 프로퍼티
    3) META-INF/build-info.properties에 들어있는 프로퍼티