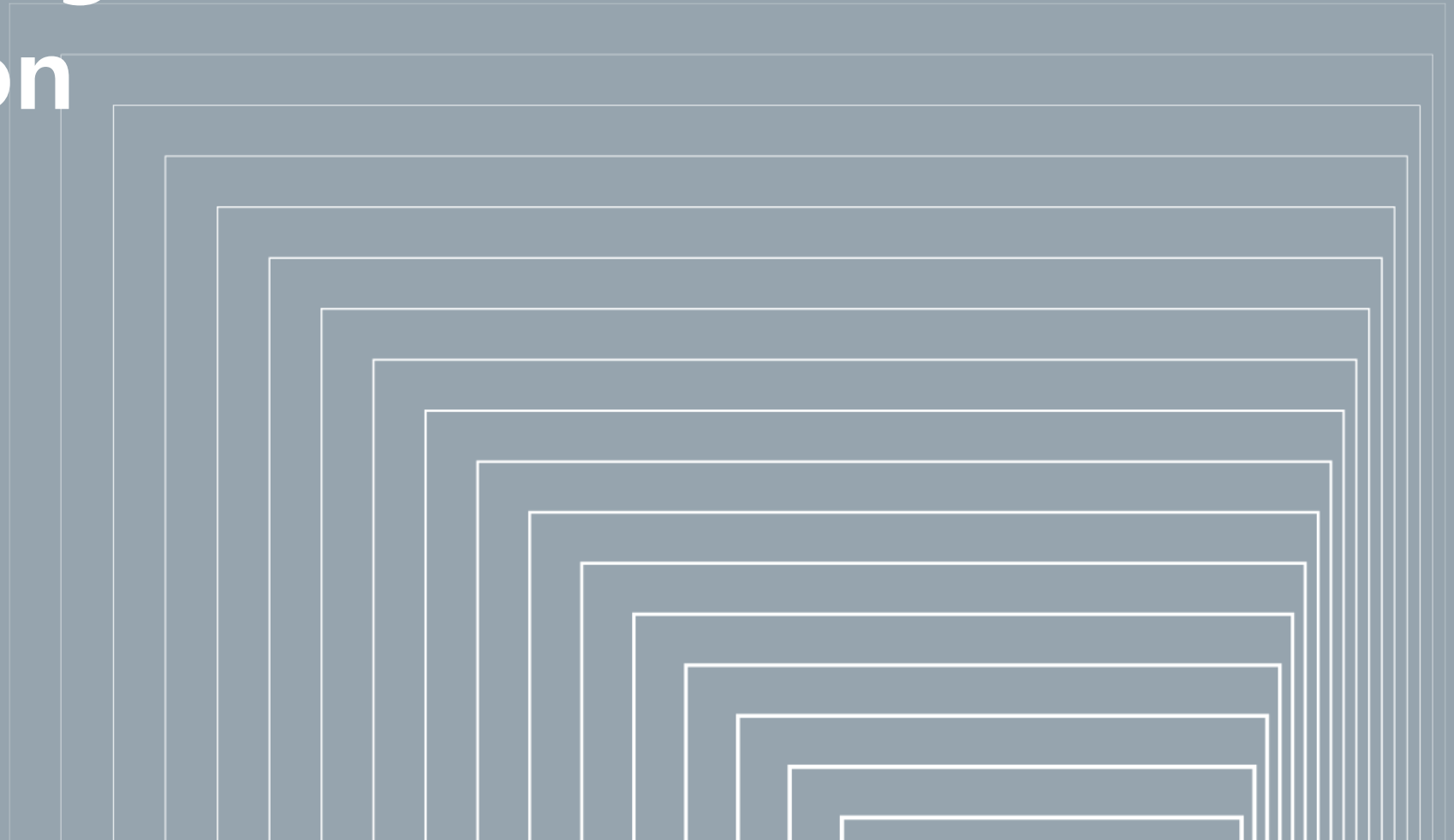
NHN Academy

6. Auto Configuration & Externalized Configuration



Auto Configuration

@EnableAutoConfiguration

- Auto Configuration은 애플리케이션에서 필요한 Bean을 유추해서 구성해 주는 기능을 담당
- @EnableAutoConfiguration 설정은 spring-boot의 AutoConfiguration 을 사용하겠다는 선언
- @SpringBootApplication 에 포함

Auto Configuration

@EnableAutoConfiguration

· java configuration 이 autoconfiguration으로 동작하기 위해서 클래스패스의 META-INF/spring.factories 에 설정

· spring-boot-autoconfigure/META-INF/spring.factories 에 spring-boot 가 제공하는 모든 AutoConfiguration 이 설정되어 있음

```
# 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.batch.BatchAutoConfiguration,\
org.springframework.boot.autoconfigure.cache.CacheAutoConfiguration,\
```

Auto Configuration

AutoConfiguration에서 제외

· auto configuration. 에서 설정을 제외하고 싶다면 @EnableAutoConfiguration의 exclude를 설정한다.
· @SpringBootApplication 을 사용한 경우도 동일한 방법으로 제외 할 수 있다.

```
@SpringBootApplication(exclude= RedisAutoConfiguration.class)
public class StudentApplication {

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

}
```

Auto Configuration

Auto Configuration 예

· RabbitAutoConfiguration 의 내용
· @ConditionalOnClass, @ConditionalOnMissingBean 등의 애너테이션으로 설정 제어

```
 * Author: Greg Turnquist, Josh Long, Stephane Nicoll, Gary Russell, Phillip Webb, Artsiom Yudovin
@Configuration(proxyBeanMethods = false)
@ConditionalOnClass({ RabbitTemplate.class, Channel.class })
@EnableConfigurationProperties(RabbitProperties.class)
@Import(RabbitAnnotationDrivenConfiguration.class)
public class RabbitAutoConfiguration {

    @Configuration(proxyBeanMethods = false)
    @ConditionalOnMissingBean(ConnectionFactory.class)
    protected static class RabbitConnectionFactoryCreator {
```

Auto Configuration

@Conditional

- Spring Framework 4.0 부터 제공
- 설정된 모든 Condition 인터페이스의 조건이 TRUE 인 경우 동작

```
@Retention(RetentionPolicy.RUNTIME)
@Target({ElementType.TYPE, ElementType.METHOD})
public @interface Conditional {

  /**
  * All {@link Condition}s that must {@linkplain Condition#matches match}
  * in order for the component to be registered.
  */
  Class? extends Condition[] value();

}
```

Auto Configuration

Condition.class 인터페이스

· matches 메소드의 반환 값이 true 인 경우, 동작

```
public interface Condition {

    boolean matches(ConditionContext context, AnnotatedTypeMetadata metadata);

}
```

Auto Configuration

@ConditionalOnXXX

· spring-boot 가 제공하는 @Conditional 의 확장

구분	내용	비고
@ConditionalOnWebApplication	프로젝트가 웹 애플리케이션이면 설정 동작	-
@ConditionalOnBean	해당 Bean 이 Spring Context 에 존재하면 동작	Auto configuration only
@ConditionalOnMissingBean	해당 Bean 이 Spring Context 에 존재하지 않으면 동작	Auto configuration only
@ConditionalOnClass	해당 클래스가 존재하면 자동설정 등록	-
@ConditionalOnMissingClass	해당 클래스가 존재하지 않으면 자동설정 등록	-

Auto Configuration

@ConditionalOnXXX

구분	내용	비고
@ConditionalOnResource	자원이(file 등) 존재하면 동작	-
@ConditionalOnProperty	프로퍼티가 존재하면 동작	-
@ConditionalOnJava	JVM 버전에 따라 동작여부 결정	-
@ConditionalOnWarDeployment	전통적인 war 배포 방식에서만 동작	-
@ConditionalOnExpression	SpEL 의 결과에 따라 동작여부 결정	-

Auto Configuration

@ConditionalOnBean

- Bean 이 이미 설정된 경우에 동작
- MyService 타입의 Bean 이 BeanFactory 에 이미 등록된 경우에 동작한다.
- Configuration 이 **AutoConfiguration**에 등록된 경우에 사용할 수 있다.

```
@Configuration
public class MyAutoConfiguration {

    @ConditionalOnBean
    @Bean
    public MyService myService() {
        ...
    }

}
```

Auto Configuration

@ConditionalOnMissingBean

- BeanFactory에 Bean이 설정되지 않은 경우에 동작
- MyService 타입의 Bean이 BeanFactory에 등록되지 않은 경우에 동작한다.
- Configuration 이 **AutoConfiguration**에 등록된 경우에 사용할 수 있다.

```
@Configuration
public class MyAutoConfiguration {

    @ConditionalOnMissingBean
    @Bean
    public MyService myService() {
        ...
    }

}
```

[실습] @ConditionalOnXXX 사용하기

목표

- 제공하는 코드의 모든 Unit Test가 통과하도록 코드를 수정한다.
- ConditionalDemoConfig.java 및 기타 소스코드를 수정하십시오.

- **https://github.com/edu-springboot/edu-springboot-conditional-workshop.git**

예상시간

- 10분

Externalized Configuration

· spring-boot는 같은 소스코드로 여러 환경에서 동작할 수 있도록 외부화 설정을 제공한다.

· **java properties**, **YAML**, **환경변수**, **실행 인자**로 설정 가능

· 전체 프로젝트의 설정은 .properties, .yaml 중 하나만 사용하는 것을 권장.

· 같은 곳에 application.propreties, application.yaml 이 동시에 존재하면 application.propreties 이 우선함

[시연] 포트 변경

NHN Academy

목표

- 학생정보 시스템의 서비스 포트를 8080에서 8888로 변경한다.

방법

- application.properties
- 환경변수
- 실행 명령어 인자 (Command Line argument)

[시연] 포트 변경

application.properties

```
server.port=8888
```

환경 변수

```
$ SERVER_PORT=8888 java -jar target/student.jar
```

Command Line Argument

```
$ java -jar target/student.jar --server.port=8888
```

[시연] 포트 변경

spring-boot가 제공하는 @ConfigurationProperties 바인딩

· spring-boot-autocofiguration.jar:org.springframework.boot.ServerProperties 에서 @ConfigurationProperties 바인딩 제공.

```
        Mandujano, Chris Bono, Parviz Rozikov
@ConfigurationProperties(prefix = "server", ignoreUnknownFields = true)
public class ServerProperties {

    Server HTTP port.
    private Integer port;
```

Externalized Configuration

Externalized Configuration example

· java proprety (application.properties)

```
nhn.student.name=zbum
```

· YAML (application.yaml)

```
nhn:
  student:
    name: zbum
```

Externalized Configuration

@Value 바인딩

· 속성값(properties)을 @Value 애너테이션으로 바인딩하여 사용

```
@Component
public class MyBean {

  @Value("${nhn.student.name}")
  private String name;

  // ...
}
```

Externalized Configuration

@ConfigurationProperties 바인딩

- 속성값(properties)을 @ConfigurationProperties로 바인딩하여 사용
- @ConfigurationProperties 로 설정된 클래스는 Dependency Injection으로 참조하여 사용

```
@ConfigurationProperties("nhn-academy.student")
public class StudentProperties {

private String firstName;

// getters / setters...

}
```

Externalized Configuration

Annotation Processor

· configuration metadata를 작성하면 IDE에서 "자동 완성" 기능을 사용할 수 있다.
· spring-boot-configuration-processor 를 의존성에 설정하면 configuration metadata 를 자동 생성한다.

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

```
dependencies {
 annotationProcessor "org.springframework.boot:spring-boot-configuration-processor"
}
```

@ConfigurationProperties의 Relaxed Binding

· 속성값을 @ConfigurationProperties빈에 바인딩하기 위해 Relaxed Binding 을 사용하기 때문에 이름이 정확히 일치할 필요는 없음.
· @Value 를 사용한 경우, Relaxed Binding 을 지원하지 않음

구분	내용	비고
nhn-academy.student.first-name	권장	
nhnAcademy.student.firstName	카멜케이스 표현	
nhn_academy.student.first_name	언더스코어 표현	
NHNACADEMY_STUDENT_FIRSTNAME	대문자 형식 (시스템 환경변수에 권장)	

Externalized Configuration

@ConfigurationPropertiesScan

· @ConfurationProperties 는 @ConfigurationPropertiesScan 사용하여 Bean으로 활성화 해야 함

· 설정한 클래스의 base package 하위의 모든 @ConfurationProperties 을 스캔

```
@SpringBootApplication
@ConfigurationPropertiesScan
public class StudentApplication {

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

}
```

Externalized Configuration

@EnableConfigurationProperties

· @ConfurationProperties 는 @EnableConfigurationProperties 를 사용하여 Bean으로 활성화 해야 함

· value 에 지정한 ConfigurationProperties 클래스를 Bean 으로 활성화

```
@SpringBootApplication
@EnableConfigurationProperties(value= SystemNameProperties.class)
public class StudentApplication {

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

}
```

Externalized Configuration

profile 활성화

- 프로필 지정 설정파일은 spring.profiles.active 인자로 로딩 여부가 결정된다.
- 만약, prod 프로파일을 지정했다면, application.properties 와 application-prod.properties 를 모두 로딩한다.

```
$ java –jar target/student.jar ––spring.profiles.active=prod
```

Externalized Configuration 우선순위

· spring-boot 는 다음 순서로 설정을 읽어 들인다. 중복되는 경우, 덮어쓰게 된다(override).

구분	내용	비고
application.properties	application.properties 내의 설정, 프로파일에 따라 순위 변경	
OS 환경 변수	OS 환경 변수	
SPRING_APPLICATION_JSON	json 형식의 환경 변수	
실행 명령어와 함께 전달된 인자	java –jar student.jar --server.port=9999	
@TestPropertiesSource	테스트 코드에 포함된 애너테이션	

Application Properties 우선순위

· application.properties 는 다음의 순서로 설정을 읽어 들인다.
· 실행 인자로 제공하는 spring.profiles.active 설정으로 application-{profile}.properties 를 사용할 것인지 결정한다.

구분	내용	비고
application.properties (inside jar)	Jar 파일 내의 application.properties	
application-{profile}.properties (inside jar)	Jar 파일 내의 application-{profile}.properties	
application.properties (outside jar)	Jar 파일 밖의 application-{profile}.properties	
application-{profile}.properties (outside jar)	Jar 파일 밖의 application-{profile}.properties	

Externalized Configuration

Application Properties 우선순위

· application.properties 위치를 찾아가는 순서에 따라 최종 설정이 결정된다.

구분	내용	비고
Classpath root	classpath:/application.properties	
Classpath 의 /config 패키지	classpath:/config/application.properties	
실행 디렉토리	${current directory}/application.properties	
실행 디렉토리의 config 디렉토리	${current directory}/config/application.properties	

[실습] Account 시스템 외부화 설정

NHN Academy

목표

- application의 개발자 정보를 제공하는 API를 작성.
- com.nhn.account.author 속성을 사용.
- @ConfigurationProperties 사용

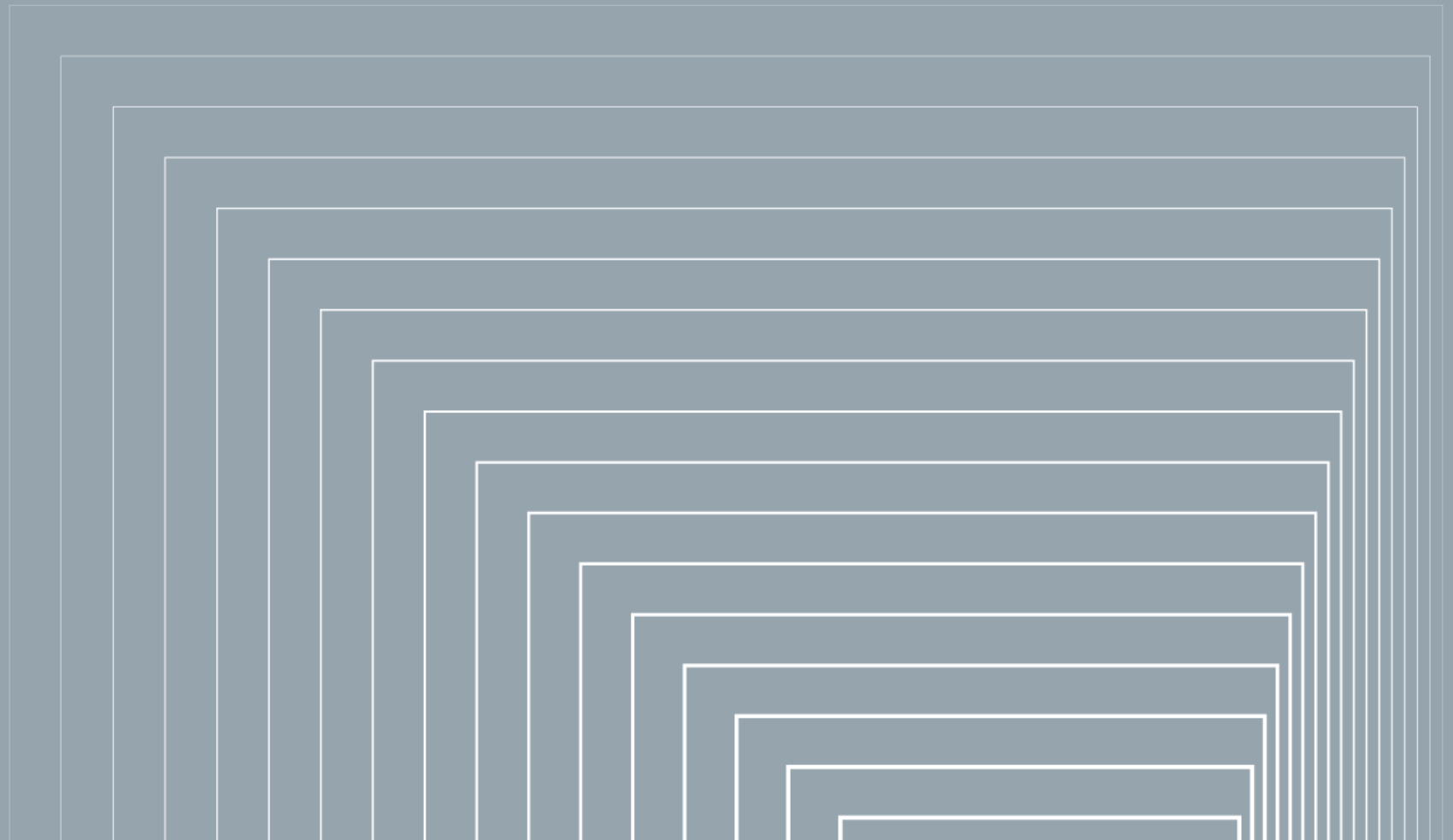
- 개발 url : **http://localhost:8080/system/author**
- 결과 : {"author": "test"}

예상시간

- 10분

NHN Academy

Q&A



NHN Academy

감사합니다