

Update 2023.02.07 17:21 ☐

1.Introducing

목차

1. Introducing
2. Simple Rest Application Development using Spring Boot
3. Initialize and Run Spring Boot Project
4. Simple Web Application Development using Spring Boot
5. Dependency Management and Main Function
6. Auto Configuration And Externalized Configuration
7. Developer Tools
8. Spring Boot Actuator
9. Test
10. Custom Spring Boot Starter

교육의 목적

이 교육을 수료하면

- Spring Boot의 구조를 알게 된다.
- Spring Boot로 애플리케이션 개발이 가능해 진다.
- Spring Boot의 배포 모듈인 Starter를 개발할 수 있게 된다.
- Spring Boot의 Product-Ready 기능을 활용할 수 있게 된다.

교육의 대상

교육의 대상

- Spring Framework 의 DI, IoC를 알고 있는 개발자.
- Spring Boot로 개발하고 있지만
- 조금 더 잘 사용하고 싶은 개발자.
- 동작 원리를 알고 싶은 개발자.

교육 난이도

- 초급 ~ 중급

Spring Boot 프로젝트 시작 (2012.10.17)

- <https://jira.spring.io/browse/SPR-9888>

요청 요약

- 서블릿이 필요없는 통합 컴포넌트 모델
- 개발자가 애플리케이션 설정을 위해 하나의 설정 모델만 학습하면 되는 환경
- public static void main 으로 실행/종료 단순화
- 단순한 자바 클래스로딩 구조
- 단순한 개발툴

회신 (by Phil Webb 2013.08)

- 스프링 프레임워크를 부분적으로 수정하는 대신, Spring Boot 라는 프로젝트를 시작
- 이 요청은 Spring Boot 의 기원이라고 할 수 있다.



Spring Boot Release

Spring Boot Release Note

- <https://github.com/spring-projects/spring-boot/wiki>

Spring Boot 0.5.0.M1 발표

- <https://spring.io/blog/2013/08/06/spring-boot-simplifying-spring-for-everyone>

Spring Boot 1.5.X.RELEASE (2017.01 – EOL)

- java 8 이상 지원
- Spring Framework 4.3
- Tomcat 8.5, Hibernate 5.0 · Configuration Properties 에 JSR303 지원

Spring Boot 2.0.X

- Java Base line : java 8 (java 7 이하를 지원하지 않음)
- Spring Framework 5.0
- Default Datasource : HikariCP

Spring Boot 2.3.X.RELEASE(2020.05)

- java 14 지원
- graceful shutdown 지원
- spring-boot-starter-validation 이 spring-boot-starter-web 에서 제외됨

Spring Boot 2.4(2020.11)

- java 15 지원
- 새로운 버전 스킴 적용 (2.3.5.RELEASE -->2.4.0)
- Docker Image Building 지원(jar)

Spring Boot 2.5(2021.05)

- java 16 지원
- 환경변수 Prefix
- Docker Image Building 지원(war)

Spring Boot 2.6(2021.11)

- java 16의 record 를 @ConfigurationProperties 로 사용가능
- 순환참조 빈은 금지가 기본 (spring.main.allow-circular-references)

Spring Boot 2.7(2022.05)

- auto configuration 파일 위치 변경
- spring.factories --> META-INF/spring/org.springframework.boot.autoconfigure.AutoConfiguration.imports

Spring Boot 3.0.0-GA(2022.11)

- java 17 지원 (java 17 이상부터 사용가능)
- Spring Framework 6

스프링 부트의 Major 버전이 변경될때, Spring Framework, Java 버전이 변경됩니다.

Spring Boot 목표-1

Java -jar 로 실행이 가능

- \$ java -help 사용법:

java [-options] class [args...] (클래스 실행)

java [-options] -jar jarfile [args...]



Spring Boot 목표-2

- 빠르고 광범위한 getting-started 경험
- 별도의 설정 없이 바로 사용 (out-of-box)
- 비기능 요구사항 기본제공
- 코드 생성이나 XML 설정이 필요 없음

Spring Boot의 기능

단독으로 실행가능한 애플리케이션 생성

- 실행형 jar, 실행형 war

내장형 웹 애플리케이션 서버 지원

- Tomcat, Jetty, Undertow, Netty for WebFlux

기본 설정된 Starter 모듈

- 의존성 (library dependency)
- 버전 호환성 보장 (dependencyManagement)
- 자동 설정(Auto Configuration)

상용화에 필요한 통계, 상태점검 외부설정 지원

- Actuator (Health, metrics)
- 외부 설정

Spring Framework과의 비교(1/2)

라이브러리 의존성을 pom.xml 직접 설정해야 한다.

· spring boot에서는 spring-boot-starter-{module}만 설정하면 필요한 라이브러리 설정 완료

버전 정보를 직접 설정하고 테스트 해야 한다.

· spring-boot-starter-parent에서 spring 모듈의 버전 및 3rd Party 라이브러리 버전도 제공
· 런타임에만 확인 가능한 성가신 작업!

Web Application Server에 배포해야 한다.

· spring boot에서는 내장형 Web Application Server를 제공하기 때문에 서버를 구매하거나 설정할 필요가 없다.

image.png

Quiz

- [\[quiz\] Introducing](#)

Update 2022.12.12 12:57 ☐

2.Simple Rest Application Development using Spring Boot

02.SimpleRestApplicationDevelopmentUsingSpringBoot.20211228.pptx

[시연] 간단한 Dependency Injection 실습

목표

· 학생 점수 조회 시스템을 개발한다.
· 테스트 코드로 결과를 확인한다.

image.png

[시연] 간단한 Dependency Injection 실습

프로젝트 생성

- <https://start.spring.io>

Project Metadata

- groupId: com.nhnacademy.edu.springboot
- artifactId: student
- version: 1.0.0
- Java : 11

Dependencies

- lombok

image.png

[시연] 간단한 Dependency Injection 실습

자동생성 코드 살펴보기 - pom.xml

spring-boot-starter-parent

- spring-boot-starter-parent의 버전 정보가 전체 프로젝트의 버전정보를 관리한다.
- BOM (Bill of Materials - 자제 명세서)
- BOM 에 기술된 정보로 3rd Party 라이브러리 호환성을 보장할 수 있다.
- 프로젝트의 dependency 에는 버전 정보를 기술하지 않는다.

spring-boot-starter

- spring boot starter 의 이름은 항상 spring-boot-starter 으로 시작한다.
- 스프링의 다른 기능을 사용하고 싶으면 spring-boot-starter-{기술명} 으로 대부분 작성할 수 있다.

[시연] 간단한 Dependency Injection 실습

자동생성 코드 살펴보기 – StudentApplication.java

SpringApplication

- spring-boot 실행의 시작점
- static method 인 run으로 실행한다.
- SpringApplication의 객체를 생성 후 실행하거나 SpringApplicationBuilder 로 실행 가능

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

[시연] 간단한 Dependency Injection 실습

자동생성 코드 살펴보기 – @SpringBootApplication

다음 Annotation 을 포함한 Meta Annotation

- **@EnableAutoConfiguration**
 - 자동설정 기능을 활성화 한다.
 - 클래스 패스에 라이브러리가 존재하면 자동으로 Bean 을 설정한다.
- **@ComponentScan**
 - basePackage 하위의 컴포넌트를 스캔하여 Bean 으로 등록한다.
- **@SpringBootConfiguration**

- 설정된 클래스 파일은 설정(java config)으로 사용할 수 있다.

[시연] 간단한 Dependency Injection 실습

학생 정보 클래스 (이름, 점수)

```
@Getter
@Setter
@EqualsAndHashCode
public class Student {
    private final String name;
    private final Integer score;
    public Student(String name, Integer score) {
        this.name = name;
        this.score = score;
    }
}
```

StudentRepository.java 개발

- NhnStudentService.java 가 참조하여 사용할 인터페이스
- 모든 학생정보를 반환할 findAll 메서드를 선언
- 인터페이스이므로 메소드 구현은 작성하지 않는다.

```
package com.nhn.edu.springboot.student;
public interface StudentRepository {
    List<Student> findAll();
}
```

DummyStudentRepository.java 개발

- StudentRepository 인터페이스의 구현체
- 이번 실습에서는 임의의 Student 객체를 2개 반환

```
@Component
public class DummyStudentRepository implements StudentRepository {
```

```
@Override public List<Student> findAll() { return List.of(new Student("zbum",100),new
Student("manty",80)); }
```

StudentService.java 개발

- Student 정보를 관리용 Service 인터페이스
- 모든 학생정보를 조회하는 getStudents 메서드 선언

```
public interface StudentService {  
    List<Student> getStudents();  
}
```

[시연] 간단한 Dependency Injection 실습

Dependency Injection 방식 3가지

Spring Boot 에서 생성자 주입은 3가지 방식 제공

- 생성자 주입
 - 생성자를 선언하면 인자에 객체 주입
 - 권장하는 방식
- @Autowired
 - 클래스 변수에 @Autowired 애너테이션을 설정하여 객체 주입
- Setter
 - setter 메서드를 선언하여 객체 주입

[시연] 간단한 Dependency Injection 실습

NhnStudentService.java 개발

- StudentRepository 인터페이스의 findAll 메서드를 사용
- Dependency Injection 으로 StudentRepository 에는 DummyStudentRepository 객체 할당

생성자 주입

```
@Service  
public class NhnStudentService implements StudentService {  
    private final StudentRepository studentRepository;  
    public NhnStudentService(StudentRepository studentRepository) {  
        this.studentRepository = studentRepository;  
    }  
    @Override  
    public List<Student> getStudents() {  
        return studentRepository.findAll();  
    }  
}
```

Autowired

```
@Service
public class NhnStudentService implements StudentService {
    @Autowired
    private StudentRepository studentRepository;
```

```
@Override public List<Student> getStudents() { return studentRepository.findAll(); }
```

setter 메소드

```
@Service
public class NhnStudentService implements StudentService {
    @Autowired
    private StudentRepository studentRepository;
```

```
public void setStudentRepository(StudentRepository studentRepository) { this.studentRepository =
studentRepository;
}
```

```
@Override public List<Student> getStudents() { return studentRepository.findAll(); }
```

[시연] 간단한 Dependency Injection 실습

NhnStudentServiceTest.java 개발

- NhnStudentRepositoryTest 빈의 통합 테스트
- @SpringBootTest 를 선언하여 모든 설정을 로딩한다.

```
//@RunWith(SpringRunner.class) junit4 에서 필수..
@SpringBootTest
class NhnStudentServiceTest {
    @Autowired
    StudentService studentService;
```

```
@Test void testGetStudents() { // when List<Student> actual = studentService.getStudents(); // then
Assertions.assertThat(actual).hasSize(2); }
```

실습

- [Spring Boot로 Dependency Injection 실습](#)

[시연] JPA 실습

목표

- [시연]을 수정하여 DBMS 에 데이터를 저장한다.

- 데이터에 접근하는 코드를 JPA 로 작성한다.
- 테스트 코드로 결과를 확인한다.

UML



[시연] JPA 실습

Maven 라이브러리 의존성 추가

- pom.xml의 에 다음 라이브러리 의존성 추가
- in memory 데이터베이스인 h2 database 를 사용

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

```
<dependency> <groupId>com.h2database</groupId> <artifactId>h2</artifactId> <scope>runtime</scope>
</dependency>
```

[시연] JPA 실습

Student.java 수정

- @Entity 추가
- @Id 추가
- 기본 생성자 추가

```
@Getter @Setter @EqualsAndHashCode
@Entity public class Student {
@Id
private Long id;    private String name;    private Integer score;
```

```
public Student() {} public Student(Long id, String name, Integer score) { this.id = id; this.name = name;
this.score = score; }
```

StudentRepository.java 수정

- StudentRepository.java 는 JpaRepository 를 상속하도록 변경한다.
- DummyStudentRepository.java를 삭제한다.
- findAll() 메서드를 삭제한다. (JpaRepository 에 findById, findAll, save 등의 데이터 처리메서드가 이미 존재한다.)

```
package com.nhn.edu.springboot.student;
```

```
public interface StudentRepository extends JpaRepository<Student, Long> { }
```

StudentRepositoryTest.java 수정

```
@Test
void testStudentRepository() {
    //given
    Student zbum = new Student(1L, "zbum", 100);
    studentRepository.save(zbum);
    //when
    Optional<Student> actual = studentRepository.findById(1L);
    //then
    Assertions.assertThat(actual).isPresent();
    Assertions.assertThat(actual.get()).isEqualTo(zbum);
}
```

[실습] JPA 적용

- [Spring Boot로 JPA 적용 실습](#)

[시연] MySQL 사용

목표

- MySQL 에 데이터를 저장하도록 수정한다.
- mysql 은 docker 로 실행한다.

UML



[시연] MySQL 사용

MySQL 준비

- 다음 명령어로 MySQL을 실행합니다.

```
$ docker run --name edu-mysql -e MYSQL_ROOT_PASSWORD=test -d -p3306:3306
mysql:5.7.35
```

- 접속 테스트

```
$ mysql -u root -p -P3306 -h 127.0.0.1
```

- 데이터베이스 생성

```
mysql> create database student_test;
```

[시연] MySql 사용

dependency 추가

- h2 삭제
- mysql-connector-java 추가

```
<dependency>  
<groupId>mysql</groupId>  
<artifactId>mysql-connector-java</artifactId>  
<version>8.0.31</version>  
</dependency>
```

application.properties 수정

- JPA 테이블 생성 및 SQL 로깅.

```
spring.jpa.generate-ddl=true  
spring.jpa.show-sql=true
```

- datasource

```
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver  
spring.datasource.url=jdbc:mysql://localhost:3306/student_test?  
serverTimezone=UTC&characterEncoding=UTF-8
```

```
spring.jpa.hibernate.naming.physical-  
strategy=org.hibernate.boot.model.naming.PhysicalNamingStrategyStandardImpl
```

```
spring.datasource.username=root spring.datasource.password=test
```

[실습] MySql 사용

- [Spring Boot로 MySql 적용 실습](#)

[시연] RestApi 개발

목표

- 학생정보 조회/등록 RestApi를 개발한다.

개발 API

- GET /students
- GET /students/{id}
- POST /students
- DELETE /students/{id}

UML



[시연] RestApi 개발

API 설계

GET /students

* Response

```
[
  {
    "id" : 1,
    "name" : "zbum",
    "score" : 100
  },
  {
    "id" : 2,
    "name" : "manty",
    "score" : 80
  }
]
```

GET /students/{id}

* Response

```
{
  "id" : 1,
  "name" : "zbum",
  "score" : 100
}
```

POST /students

- Request

```
{
  "id" : 1,
  "name" : "zbum",
  "score" : 100
}
```

- Response
 - status code : 201

```
{
  "id" : 1,
  "name" : "zbum",
  "score" : 100
}
```

DELETE /students/{id}

- Response

```
{
  "result" : "OK"
}
```

[시연] RestApi 개발

의존성 변경

- spring-boot-starter 를 spring-boot-starter-web 으로 변경

[시연] RestApi 개발

StudentService.java 수정

- StudentService 에 다음과 같이 메서드를 추가한다.

```
List<Student>getStudents();
```

```
StudentcreateStudent(Student student);
```

```
Student getStudent(Long id);
```

```
void deleteStudent(Long id);
```

[시연] RestApi 개발

StudentController.java 수정

- GET /students API – 학생정보 리스트 조회
- StudentService 인터페이스의 getStudents() 메서드를 사용

```
@RestController
public class StudentController {
    private final StudentService studentService;
```

```
public StudentController(StudentService studentService) { this.studentService = studentService; }
```

```
@GetMapping("/students") public List<Student> getStudents() { return studentService.getStudents(); } }
```

[시연] RestApi 개발

StudentController.java 수정

- GET /students/{id} API – 학생정보 1건 조회
- StudentService 인터페이스의 getStudent 메서드를 사용

```
@RestController
public class StudentController {
    << 생략 >>
    @GetMapping("/students/{id}")
    public Student getStudent(@PathVariable Long id) {
        return studentService.getStudent(id);
    }
}
```

[시연] RestApi 개발

StudentController.java 수정

- POST /students API – 학생정보 등록
- StudentService 인터페이스의 createStudent 메서드를 사용

```
@RestController
public class StudentController {
    << 생략 >>
    @PostMapping("/students")
```

```
@ResponseStatus(HttpStatus.CREATED)
public Student createStudent(@RequestBody Student student) {
    return studentService.createStudent(student);
}
}
```

[시연] RestApi 개발

StudentController.java 수정

- DELETE /students/{id} API – 학생정보 삭제
- StudentService 인터페이스의 deleteStudent 메서드를 사용

```
@RestController
public class StudentController {
    <<생략>>
    @DeleteMapping("/students/{id}")
    public String deleteStudent(@PathVariable Long id) {
        studentService.deleteStudent(id);
        return "{\"result\":\"OK\"}";
    }
}
```

[시연] RestApi 개발

NhnStudentService.java 개발 - getStudents

- @Service 비즈니스 로직 구현
- student table 의 전체 데이터 조회
- JpaRepository 가 제공하는 findAll() 메서드 사용

```
@Service
public class NhnStudentService implements StudentService {
```

```
<<생략>>
```

```
@Override @Transactional(readOnly=true) public List<Student> getStudents() { return
studentRepository.findAll(); }
```

[시연] RestApi 개발

NhnStudentService.java 개발 - getStudent

- id 에 해당하는 데이터 조회

- JpaRepository의 findById() 메서드 사용
- 존재하지 않는 ID 요청시 에러처리

```
@Service
public class NhnStudentService implements StudentService {
```

```
<< 생략 >>
```

```
@Override @Transactional(readOnly=true) public Student getStudent(Long id) {
    return studentRepository.findById(id).orElseThrow();
}
```

[시연] RestApi 개발

NhnStudentService.java 개발 - createStudent

- Student 정보를 DB 에 저장
- JpaRepository의 save() 메서드 사용
- 이미 존재하는 ID 는 IllegalStateException 처리
- commit/rollback 을 해야 하는 경우 @Transactional 사용

```
@Service
public class NhnStudentService implements StudentService {
```

```
<< 생략 >>
```

```
@Override @Transactional public Student createStudent(Student student) {
    boolean present = studentRepository.findById(student.getId()).isPresent();
    if (present) throw new IllegalStateException("already exist " + student.getId());

```

```
    return studentRepository.save(student);
}
```

```
}
```

[시연] RestApi 개발

NhnStudentService.java 개발 - deleteStudent

- Student 정보를 DB 에서 삭제
- JpaRepository의 deleteById() 메서드 사용

- commit/rollback 을 해야 하는 경우 @Transactional 사용

```
@Service
public class NhnStudentService implements StudentService {
```

```
<< 생략 >>
```

```
@Override @Transactional public void deleteStudent(Long id) { studentRepository.deleteById(id); }
```

[시연] RestApi 개발

API Test - curl 사용

- Student 정보 등록

```
$ curl -XPOST -H"Content-Type: application/json" \
-d '{"id": 2, "name": "Manty", "score": 100}' \
http://localhost:8080/students
```

- Student 정보 등록 결과

```
{
  • "id" :
    2,
  • "name" :
    "Manty",
  • "score" :
    100
}
```

[시연] RestApi 개발

API Test - curl 사용

- Student 정보 목록 조회

```
$ curl -XGET http://localhost:8080/students
```

- Student 목록 정보 조회 결과

```
[
  • {
    ◦ "id" :
      1,
    ◦ "name" :
      "Manty",
    ◦ "score" :
      100
  },
  • {
    ◦ "id" :
      2,
    ◦ "name" :
      "Manty",
    ◦ "score" :
      100
  }
]
```

[시연] RestApi 개발

API Test - curl 사용

- Student 정보 단건 조회

```
$ curl -XGET http://localhost:8080/students/1
```

- Student 정보 조회 결과

```
{
  • "id" :
    1,
  • "name" :
    "Manty",
  • "score" :
    100
}
```

[시연] RestApi 개발

API Test - curl 사용

- Student 정보 삭제

```
$ curl -XDELETE http://localhost:8080/students/1
```

- Student 정보 삭제 결과

```
{  
  • "result" :  
    "OK"  
}
```

[실습] Spring Boot 로 RestApi 서비스 개발

- [Spring Boot로 RestApi 서비스 개발 실습](#)

[Quiz]

- [\[quiz\] Simple Rest Application Develop](#)

Update 2022.12.12 10:34 ☐

Spring Boot로 Dependency Injection 실습

목표

- 계좌관리 시스템을 개발합니다.
- 계좌의 클래스 이름은 Account 입니다.
- 모든 계좌를 조회하는 기능을 제공합니다.
- DefaultAccountService의 테스트 코드를 작성합니다.

예상시간

- 15분

설계



Update 2022.12.11 21:05 ☐

Spring Boot로 JPA 적용 실습

목표

- 계좌 정보를 데이터베이스에 저장한다.
- JPA 기술을 사용한다.
- 데이터는 H2 DB에 저장한다

예상시간

- 10분

설계



Update 2022.12.11 21:05 ☐

Spring Boot로 MySql 적용 실습

목표

· 계좌 정보를 MySql 데이터베이스에 저장한다.

예상시간

- 10분

설계



Update 2022.12.11 21:05 ☐

Spring Boot로 RestApi 서비스 개발 실습

목표

- 계좌 정보의 조회, 전체 조회, 등록, 삭제 Rest API 를 개발한다.

예상시간

- 10분

설계

image.png

Update 2022.12.12 00:57 ☐

3. Initialize and Run Spring-Boot project

Spring Boot 프로젝트의 생성지원

Spring Boot initializr

- <https://start.spring.io>

IntelliJ IDEA Ultimate

- community edition 에서는 지원하지 않음

Spring Tools 4 for Eclipse

- eclipse 프로젝트 기반
- <https://spring.io/tools>

Spring Tools 4 for Visual Studio Code

- Spring Boot 확장팩 설치 후 사용가능
- Spring Boot support in Visual Studio Code

Spring Boot initializr

- 웹기반 Spring Boot 프로젝트 생성 도구
 - <https://start.spring.io>
- 선택 옵션
 - build tool
 - language,
 - spring-boot version
 - java version
 - 라이브러리 의존성(dependency)
- Spring Boot 프로젝트팀 통계 수집



IntelliJ IDEA Ultimate

- 통합개발도구(IDE)
 - Spring Boot | IntelliJ IDEA (jetbrains.com)
 - 배포판에 포함된 Spring and SpringBoot 플러그인 사용
 - **Spring Boot initializr** 사용
 - actuator endpoint 도구 제공
 - Bean 조회 도구 제공



Spring Tools For Eclipse

- 통합개발도구(IDE)
 - Eclipse IDE 기반
 - spring 프로젝트에서 제공 Spring | Tools

- Spring Tool Suite의 새버전



Spring Tools 4 for Visual Studio Code

- 통합개발도구(IDE)
 - Visual Studio Code IDE 기반
 - spring 프로젝트에서 제공 Spring | Tools
 - 확장팩 설치 후 사용
 - Spring Boot Extension Pack - Visual Studio Marketplace



Spring Boot 프로젝트의 실행

Executable Jar/War

- 실행가능한 jar, war 생성

Build Tool

- maven , gradle 로 직접 실행

Unix/Linux Services

- init.d Service
- systemd Service

Docker/Kubernetes

- Docker Image 생성 지원

Executable Jar / War

- maven 또는 gradle 로 실행가능한 jar 또는 war 를 빌드한다.
- spring boot의 maven plugin 이나 gradle plugin 을 사용한다면 자동으로 생성할 수 있다.

```
$ mvn package //gradle bootjar
```

```
$ ls target student-0.0.1-SNAPSHOT.jar
```

```
$ java -jar target/student-0.0.1-SNAPSHOT.jar
```

Build Tool 사용

- maven 또는 gradle 로 직접 실행한다.
- 로컬, 개발환경에서 사용할 수 있다.

```
$ mvn spring-boot:run
```

```
$ gradle bootRun
```

Linux Services (CentOS, Ubuntu)

- Linux Service 에서 실행하려면 완전 실행가능한 jar 를 빌드한다.
- maven, gradle에서 아래와 같이 spring-boot plugin 설정을 수정한다.

```
<plugin>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-maven-plugin</artifactId>
<configuration>
    <executable>true</executable>
</configuration>
</plugin>
```

```
bootJar {
    launchScript()
}
```

init.d

- init.d Service 설정 및 실행

```
$ sudo ln -s /var/app/student.jar /etc/init.d/student
```

```
$ service student start
```

systemd

- systemd Service 설정 및 실행
- /etc/systemd/system/student.service 파일을 생성한다.

```
[Unit]
Description=student
After=syslog.target
```

```
[Service] User=irteam ExecStart=/var/app/student.jar SuccessExitStatus=143
```

[Install] WantedBy=multi-user.target

```
$ systemctl enable student.service
```

Docker 실행

- Dockerfile 을 직접 만들거나 빌드툴로 Docker 이미지를 생성한다.

```
$ mvn spring-boot:build-image -Dspring-boot.build-image.imageName=student
```

```
$ gradle bootBuildImage --imageName=student
```

- Docker 로 컨테이너 실행

```
$ docker run -p8080:8080 student:latest
```

Kubernetes 실행

- tag 설정 및 registry 설정.

```
$ docker tag student:0.0.1-SNAPSHOT registry.op.internal.dooray.io/nhn-  
edu/student:latest  
$ docker push registry.op.internal.dooray.io/nhn-edu/student:latest
```

- kubernetes 배포용 YAML 작성.

```
$ kubectl create deployment student \  
--image=registry.op.internal.dooray.io/nhn-edu/student:latest \  
--dry-run=client -o yaml > student.yaml
```

- Kubernetes deployment 배포

```
$ kubectl apply -f student.yaml  
deployment.apps/student created
```


- 8080 포트 노출.

```
$ kubectl expose deployment student --type=NodePort --name=student-service --port 8080
```

Quiz

- [\[quiz\] Initialize and Run Spring-Boot project](#)

실습

- [Account 서비스 실행](#)

Update 2022.12.11 21:28 ☐

Account 서비스 실행

목표

- 다양한 방법으로 Account 서비스를 실행합니다.
 - 완전실행형 jar 실행
 - java -jar 실행
 - Docker 로 실행

준비사항

Docker Desktop 설치

- <https://docs.docker.com/desktop/install/mac-install/>

예상시간

- 10분

Update 2022.12.12 00:58 ☐

4. Simple Web Application Development using Spring Boot

Spring-Boot 의 View

Spring-boot의 view 지원

- spring boot 에서 Thymeleaf, FreeMarker, Mustache, Groovy Templates ,Velocity 를 view template 으로 제공한다.

Thymeleaf

- Spring Boot는 Thymeleaf를 기본 지원하여 간단한 설정으로 사용 가능
- html 문법 내에서 view 를 구현할 수 있는 장점

Spring-Boot 의 View (Thymeleaf)

Thymeleaf

- Phil Webb (SpringOne 2017)
- <https://youtu.be/MQamx7-bCVI?t=377>



Spring-Boot의 View (Thymeleaf)

Thymeleaf의 사용

- spring-boot-starter-thymeleaf 의존성 추가

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

Spring-Boot의 View (Thymeleaf)

Thymeleaf의 사용

- 프로젝트 루트의 /src/main/resource 경로에 templates 디렉토리를 생성하고 템플릿 작성
- Ex) student.html

```
<!DOCTYPEhtml>
<htmlxmlns:th="http://www.thymeleaf.org">
<head><metacharset="utf-8"><title>welcome</title>
</head>
<body>
<h1>Name</h1>
<h2th:text="${student.name}"></h2>
<h1>Score</h1>
<h2th:text="${student.score}"></h2>
</body>
</html>
```

Spring-Boot의 View (Thymeleaf)

Thymeleaf의 사용

- view template 을 사용하려면 @Controller를 사용
- /src/main/java 경로의 com.nhnent.edu.springboot 패키지에 StudentWebController 작성
- getStudent 가 반환하는 student 는 view template의 경로 중 일부

```
@Controller
publicclassStudentWebController {
```

```
@GetMapping("/web/students/{id}")
public String getStudent(@PathVariable Long id,
                        Model model){
    model.addAttribute("student", new Student("zbum", 100));
    return "student";
}
```

Spring-Boot의 View (Thymeleaf)

Thymeleaf의 사용

- view template 의 경로변경
- 만약, application.properties 에 아래와 같이 설정되어 있고 getStudent 메서드가 "student"를 반환한다면 템플릿은 "/src/main/resources/templates/main/student.html" 에 위치해야 합니다.

```
spring.thymeleaf.enabled=true
spring.thymeleaf.prefix=classpath:/templates/main/
spring.thymeleaf.suffix=.html
```

[실습] thymeleaf 으로 html 렌더링

- [thymeleaf 으로 html 렌더링](#)

Spring-Boot의 View (JSP)

JSP의 사용

- spring-boot 에서 JSP를 사용할 수 있지만 권장하지 않습니다.
- legacy의 코드를 재사용하기 위해 학습 필요

Spring-Boot에서 JSP 제약

- war 패키징 된 경우에만 사용이 가능 (실행형 war 또는 tomcat에서 동작)
- 실행형 jar에서는 동작하지 않음
- undertow 는 JSP를 지원하지 않음

Spring-Boot의 View (JSP)

패키징 변경

- jar 에는 WEB-INF 를 포함하지 않기 때문에 war로 패키징한다.
- JSTL지원 및 JSP 컴파일을 위해서 두개의 라이브러리를 추가한다.

```
<packaging>war</packaging>
```

라이브러리 의존성

```
<dependencies>
...
<!-- JSTL for JSP -->
<dependency>
<groupId>javax.servlet</groupId>
<artifactId>jstl</artifactId>
</dependency>
<!-- Need this to compile JSP -->
<dependency>
<groupId>org.apache.tomcat.embed</groupId>
<artifactId>tomcat-embed-jasper</artifactId>
<scope>provided</scope>
</dependency>
</dependencies>
```

Spring-Boot의 View (JSP)

main 클래스 변경

- war로 패키징 한 경우, main 클래스가 SpringBootServletInitializer을 상속받도록 수정해야 합니다.

```
@SpringBootApplication
public class StudentApplication extends SpringBootServletInitializer {
    public static void main(String[] args) {
        SpringApplication.run(StudentApplication.class);
    }
    @Override
    protected SpringApplicationBuilder configure(SpringApplicationBuilder builder) {
        return builder.sources(StudentApplication.class);
    }
}
```

Spring-Boot의 View (JSP)

JSP 파일 작성

- 프로젝트에 /src/main/webapp/WEB-INF/jsp 디렉토리를 생성하고 jsp file 작성

```
<!DOCTYPEhtml>
<html lang="ko">
<head>
<meta charset="utf-8">
<title>welcome</title>
```

```
</head>
<body>
<h1>JSP Sample</h1>
<h2>${message}</h2>
</body>
</html>
```

Spring-Boot의 View (JSP)

Controller 클래스 생성

- /src/main/java 경로의 com.nhn.edu.springboot 패키지에 Controller 클래스를 작성
- 화면에 표시할 attribute 는 Map 객체를 사용

```
@Controller
public class WelcomeController {
    @GetMapping("/welcome")
    public String welcome(Map model) {
        model.put("message", "Welcome to the world!");
        return "welcome";
    }
}
```

Spring-Boot의 View (JSP)

JSP의 경로 변경

- 만약, application.properties에 아래와 같이 설정되어 있고 welcome 메서드가 "welcome"를 반환한다면 jsp 파일은 "/src/main/webapp/WEB-INF/jsp/welcome.jsp"에 위치해야 합니다.

```
spring.mvc.view.prefix=WEB-INF/jsp/
spring.mvc.view.suffix=.jsp
```

Spring-Boot의 View (JSP)

JSP를 사용하는 Spring-Boot 애플리케이션의 실행

- Executable War를 이용한 실행

```
$ mvn clean package
$ java -jar target/student-1.0.0-SNAPSHOT.war
```

- spring-boot maven plugin을 이용한 실행

```
$ mvn spring-boot:run
```

- 또는, tomcat 등 Web application Server 에 deploy

Quiz

- [\[quiz\] Simple Web Application Development using Spring Boot](#)

Update 2022.12.12 00:39 ☐

5. Dependency Management and Main Function

Dependency management

Spring Boot Starter

- Spring Framework 관련 기술을 사용하기 위한 의존성 관리 세트
- 40개 이상의 Spring Boot starter를 Spring Boot에서 제공
- 3rd Party 에서 제공

| Starter 이름 | 설명 |
|----------------------------|---|
| spring-boot-starter-parent | spring boot 프로젝트에서 상속 받아야 할 pom |
| spring-boot-starter | Auto Configuration 을 포함한 핵심 starter, logging, yaml 지원 |
| spring-boot-starter-web | RESTful, Web 애플리케이션 구축을 위한 starter, 내장 tomcat 포함 |
| spring-boot-starter-amqp | Spring AMQP, Rabbit MQ 사용을 위한 설정 |
| spring-boot-starter-mail | Java mail 을 사용하기 위한 설정, spring framework의 메일 발송기능 |

Dependency management

Spring Boot Starter

- Pivotal Software사의 공식 starter는 spring-boot-starter-* 패턴으로 명명한다.
- spring-boot-starter-* 의 라이브러리 의존성을 추가하는 것 만으로도 기본 설정으로 기능이 동작한다.
- 공식 starter가 아닌 경우는 spring-boot로 시작하지 않아야 한다. 보통 {function}-spring-boot-starter 과 같이 명명

Dependency management

spring-boot-starter-parent

- spring-boot-starter-parent는 spring-boot-dependencies를 상속
- spring-boot 버전별로 지원하는 라이브러리 의존성 목록(Bills of Materials)
- spring-boot 버전을 업그레이드하면 라이브러리 의존성도 모두 자동 업그레이드

```
<parent>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-dependencies</artifactId>
<version>2.7.6</version>
<relativePath>../../spring-boot-dependencies</relativePath>
</parent>
```

Dependency management

spring-boot-dependencies

- 사용하는 라이브러리의 버전을 property 로 관리

```
<properties>
    <activemq.version>5.16.5</activemq.version>
<antlr2.version>2.7.7</antlr2.version>
<appengine-sdk.version>1.9.98</appengine-sdk.version>
<artemis.version>2.19.1</artemis.version>
<aspectj.version>1.9.7</aspectj.version>
<assertj.version>3.22.0</assertj.version>
<atomikos.version>4.0.6</atomikos.version>
<awaitility.version>4.2.0</awaitility.version>
</properties>
```

Dependency management

spring-boot-dependencies

- dependencyManagement 로 사용할 라이브러리의 버전을 미리 지정

```
<dependencyManagement>
<dependencies>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot</artifactId>
    <version>2.7.6</version>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-web</artifactId>
    <version>2.7.6</version>
</dependency>
...
</dependencies>
</dependencyManagement>
```

Dependency management

spring-boot-starter-web

- spring-core, spring-web, spring-webmvc, 내장 tomcat 서버 및 관련 라이브러리 설정을 일괄처리

```
<dependencies>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter</artifactId>
  <version>2.7.6</version>
  <scope>compile</scope>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-json</artifactId>
  <version>2.7.6</version>
  <scope>compile</scope>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-tomcat</artifactId>
  <version>2.7.6</version>
  <scope>compile</scope>
</dependency>
```

[실습]

- [tomcat 대신 jetty 로 실행하기](#)

Main Class Function

spring-boot 프로젝트의 main 메소드

- public static void main 메서드에서 SpringApplication.run을 실행시킨다.
- args는 command 라인에서 보낸 인자를 전달한다. (--debug, --spring.profiles.active)

SpringApplication 사용 방법

- static method
- use construction
- use builder

Main Class Function

Static 메서드

- 가장 일반적인 사용방법


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

Main Class Function

생성자 사용

- static 메소드 내부에 동일한 구현이 있다.

```
@SpringBootApplication
public class StudentApplication {
    public static void main(String[] args) {
        SpringApplication application = new
        SpringApplication(StudentApplication.class);      application.run(args);
    }
}
```

Main Class Function

builder 사용

- 빌더로 여러 개의 web context를 구성할 수 있으며 parent-child 의 계층구조로 설정가능

```
@SpringBootApplication
public class StudentApplication {
```

```
    public static void main(String[] args) { new SpringApplication.Builder()
        .sources(Student.class).web(WebApplicationType.NONE)
        .child(FirstChildConfig.class).web(WebApplicationType.SERVLET)
        .sibling(SecondChildConfig.class).web(WebApplicationType.SERVLET) .run(args); }
}
```

Main Class Function

Custom Banner

- spring-boot 프로젝트의 banner custom



Main Class Function



Custom Banner

- org.springframework.boot.Banner 인터페이스를 구현하여 custom banner 개발
- banner 제거하기

```
@SpringBootApplication
public class MyApplication {
    public static void main(String[] args) {
        SpringApplication application = new
        SpringApplication(MyApplication.class);
        application.setBannerMode(Banner.Mode.OFF);
        application.run(args);
    }
}
```

Main Class Function

Resource 파일

- src/main/resource/banner.txt 파일을 생성하여 기본 banner 대체

- banner.gif, banner.jpg, banner.png 를 src/main/resources 에 복사


Main Class Function

Banner에 사용할 수 있는 변수

- banner.txt 에는 다음의 변수를 사용할 수 있다.

| Starter 이름 | 설명 |
|-----------------------------------|---|
| \${application.version} | MANIFEST.MF 에 설정한 애플리케이션의 버전 |
| \${application.formatted-version} | \${application.version} 을 (v1.0.0) 형태로 포맷 |
| \${spring-boot.version} | 사용하는 spring-boot 의 버전 |
| \${spring-boot.formatted-version} | \${spring-boot.version} 을 (v1.0.0) 형태로 포맷 |
| \${Ansi.NAME} | Ansi escape 코드의 이름을 지정 |
| \${application.title} | MANIFEST.MF 에 설정한 애플리케이션 이름 |

[시연] banner 바꾸기

- [Banner 바꾸기](#)

Update 2022.12.12 00:13 ☐

tomcat 대신 jetty 로 실행하기

- 계좌 정보를 제공하는 API 를 jetty 로 동작하도록 합니다.

- 5분

- spring-boot-starter-web 에서 spring-boot-starter-tomcat 을 exclude 한다.
- spring-boot-starter-jetty 의 의존성을 추가한다.

Banner 바꾸기

- 프로젝트 실행시에 Banner 를 변경합니다.

- 배너에 placeholder 를 표시하려면 java -jar 방식으로 실행해야 합니다.

- 5분

[illegible]

- ## 6. Auto Configuration & Externalized Configuration

- Auto Configuration은 애플리케이션에서 필요한 Bean을 유추해서 구성해 주는 기능을 담당

- @EnableAutoConfiguration 설정은 spring-boot의 AutoConfiguration 을 사용하겠다는 선언
- @SpringBootApplication 에 포함

@EnableAutoConfiguration

- java configuration 은 auto configuration 으로 동작할 수 있음
- java configuration 이 auto configuration으로 동작하기 위해서 설정파일에 대상 Configuration 이 설정되어야함

2.6.x 이전

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



2.7.x 이후

- spring-boot-autoconfigure/META-INF/spring/org.springframework.boot.autoconfigure.AutoConfiguration.imports



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 예

- @ConditionalOnClass, @ConditionalOnMissingBean 등의 애너테이션으로 설정 제어
- EmbeddedWebServerFactoryCustomizerAutoConfiguration.java

```
@AutoConfiguration
@ConditionalOnWebApplication
@EnableConfigurationProperties(ServerProperties.class)
public class EmbeddedWebServerFactoryCustomizerAutoConfiguration {
```

```

<span class="hljs-comment">/**
 * Nested configuration if Tomcat is being used.
 */</span>
<span class="hljs-meta">@Configuration</span>(proxyBeanMethods =<span
class="hljs-keyword">>false</span>)
<span class="hljs-meta">@ConditionalOnClass</span>({ Tomcat<span
class="hljs-class">.<span class="hljs-keyword">class</span>,<span
class="hljs-title">UpgradeProtocol</span>.<span class="hljs-
title">class</span> })
<span class="hljs-title">public</span><span class="hljs-
title">static</span><span class="hljs-title">class</span><span
class="hljs-title">TomcatWebServerFactoryCustomizerConfiguration</span>
</span>{

    <span class="hljs-meta">@Bean</span>
    <span class="hljs-function"><span class="hljs-keyword">public</span>
TomcatWebServerFactoryCustomizer<span class="hljs-
title">tomcatWebServerFactoryCustomizer</span><span class="hljs-params">
(Environment environment,
    ServerProperties serverProperties)</span> </span>{
        <span class="hljs-keyword">return</span><span class="hljs-
keyword">new</span> TomcatWebServerFactoryCustomizer(environment,
serverProperties);
    }

}

<span class="hljs-comment">/**
 * Nested configuration if Jetty is being used.
 */</span>
<span class="hljs-meta">@Configuration</span>(proxyBeanMethods =<span
class="hljs-keyword">>false</span>)
<span class="hljs-meta">@ConditionalOnClass</span>({ Server<span
class="hljs-class">.<span class="hljs-keyword">class</span>,<span
class="hljs-title">Loader</span>.<span class="hljs-title">class</span>,<span
class="hljs-title">WebApplicationContext</span>.<span class="hljs-
title">class</span> })
<span class="hljs-title">public</span><span class="hljs-
title">static</span><span class="hljs-title">class</span><span
class="hljs-title">JettyWebServerFactoryCustomizerConfiguration</span>
</span>{

    <span class="hljs-meta">@Bean</span>
    <span class="hljs-function"><span class="hljs-keyword">public</span>
JettyWebServerFactoryCustomizer<span class="hljs-
title">jettyWebServerFactoryCustomizer</span><span class="hljs-params">
(Environment environment,
    ServerProperties serverProperties)</span> </span>{
        <span class="hljs-keyword">return</span><span class="hljs-
keyword">new</span> JettyWebServerFactoryCustomizer(environment,

```

```
serverProperties);
    }

}
<span class="hljs-comment">// 생략</span>
```

```
}
```

@Conditional

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

Conditional 애너테이션

```
@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();
}
```

Condition.java

- matches 메소드의 반환 값이 true 인 경우, 동작

```
public interface Condition {
```

```
boolean matches(ConditionContext context, AnnotatedTypeMetadata metadata);
```

```
}
```

@ConditionalOnXXX (1)

- 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 |

| 구분 | 내용 | 비고 |
|----------------------------|--|-------------------------|
| @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 | - |

@ConditionalOnXXX (2)

| 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 | - |

@ConditionalOnBean

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

```
@Configuration
public class MyAutoConfiguration {
```

```
<span class="hljs-meta">@ConditionalOnBean</span>
<span class="hljs-meta">@Bean</span>
<span class="hljs-function"><span class="hljs-keyword">public</span>
MyService<span class="hljs-title">myService</span><span class="hljs-params">()</span> </span>{
    ...
}
```

```
}
```

@ConditionalOnMissingBean

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

```
@Configuration
public class MyAutoConfiguration {
```

```
<span class="hljs-meta">@ConditionalOnMissingBean</span>
<span class="hljs-meta">@Bean</span>
<span class="hljs-function"><span class="hljs-keyword">public</span>
MyService<span class="hljs-title">myService</span><span class="hljs-params">()</span> </span>{
    ...
}
```

```
}
```

실습

- [@Condition](#), [@ConditionalOnXXX](#) 사용하기

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.properties, application.yaml 이 동시에 존재하면 application.properties 가 우선함

[시연] 포트 변경

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
```

환경변수


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

실행 명령어 인자 (Command Line argument)

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

[시연] 포트 변경

- spring-boot가 제공하는 @ConfigurationProperties 바인딩으로 동작
- spring-boot-autocofiguration.jar:org.springframework.boot.ServerProperties 에서 @ConfigurationProperties 바인딩 제공

image.png

Externalized Configuration example

java property (application.properties)

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

YAML (application.yaml)

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

Externalized Configuration

- Spring Boot 는 설정값을 바인딩 하기 위한 2가지 방법을 제공합니다.

@Value 바인딩

@ConfigurationProperties 바인딩

@Value 바인딩

- 속성값(properties)을 @Value 애너테이션으로 바인딩하여 사용

```
@Component
public class MyBean {
```

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

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

@ConfigurationProperties 바인딩

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

```
@ConfigurationProperties("nhn.student")
public class StudentProperties {
    private String name;
    // getters / setters...
}
```

Externalized Configuration 자동완성

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

maven

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

gradle

```
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

ConfigurationProperties 구현예

```
@ConfigurationProperties("nhn-academy.student")
public class StudentProperties {
    private String firstName;
    // getters / setters...
}
```

바인딩 가능한 속성

| 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) | |

@ConfigurationProperties 활성화

- @ConfurationProperties 를 활성화 하여 빈으로 등록해야 사용가능

@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);
    }
}
```

@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);
    }
}
```

spring profile

- 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.
- 중복되는 경우, 덮어쓰게 된다(override)

| 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 | |

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 | |

| 구분 | 내용 | 비고 |
|----------------------|--|----|
| 실행 디렉토리 | <code>\${current directory}/application.properties</code> | |
| 실행 디렉토리의 config 디렉토리 | <code>\${current directory}/config/application.properties</code> | |

[실습] Account 시스템 외부화 설정

- [Account 시스템 외부화 설정](#)

[Quiz]

- [\[quiz\] Auto Configuration & Externalized Configuration](#)

Update 2022.12.12 01:26 ☐

@Condition, @ConditionalOnXXX 사용하기

목표

- spring boot 의 auto configuration 의 핵심 애너테이션인 @ConditionalOnXXX 을 사용하여 제공하는 코드의 모든 Unit Test가 통과하도록 코드를 수정한다.

작업 방법

- 다음의 url 에서 소스코드를 clone 합니다.
 - <https://github.com/edu-springboot/edu-springboot-conditional-workshop.git>
- ConditionalDemoConfig.java 및 기타 소스코드를 수정합니다.
- 모든 테스트 케이스를 PASS 하면 성공!!

Update 2022.12.13 13:28 ☐

Account 시스템 외부화 설정

목표

- application의 버전 정보를 제공하는 API를 작성.
- @ConfigurationProperties 를 사용합니다.
- com.nhn.account.system.version 속성을 사용. (prefix="com.nhn.account.system")
- 개발 url : GET /system/version
- 결과 : {"version": "1.0.0"}

예상시간

- 15분

Update 2022.12.13 14:15 ☐

7. Developer Tools

spring-boot 개발자 도구

spring-boot 는 개발자 편의를 위한 툴을 제공한다.

- 자동재시작
- 라이브 리로드
- 전역 설정
- 원격 애플리케이션

spring-boot 개발자 도구 활성화

- spring-boot-devtools Module 의존성을 추가하여 개발자도구 활성화

maven

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

gradle

```
dependencies {
    developmentOnly("org.springframework.boot:spring-boot-devtools")
}
```

개발자 도구 비활성화

- java -jar 로 실행하면 상용(Production) 환경으로 판단하고 비활성화됨
- 서블릿 컨테이너에서 동작하면 자동 비활성화 되지 않기 때문에 -Dspring.devtools.restart.enabled=false 설정 필요
- maven plugin의 repackaged 로 생성된 바이너리파일에는 자동으로 devtools가 제거됨

개발자 도구와 캐시(cache)

- 개발자 도구는 뷰 템플릿, 정적 리소스 캐시(cache) 를 자동으로 비활성화 함

개발자 도구 - 자동 재시작

- 기본적으로 애플리케이션의 클래스 패스 내 파일이 변경되면 자동으로 재시작
- 정적 자원이나 뷰 템플릿은 재시작하지 않음
- IDE에서 코드 변경 후 잦은 테스트를 할 경우 유용함

| IDE | 자동 재시작 실행 방법 |
|---------------|---|
| Eclipse | 수정한 파일을 저장하면 바로 실행 |
| IntelliJ IDEA | Build -> Build Project (Ctrl + F9 , Cmd + F9) |
| maven | \$ mvn compile |
| gradle | \$ gradle build |

실습

- [개발자 도구 테스트\(자동 재시작\)](#)

개발자 도구 - 라이브 리로드

- spring-boot-devtools 에 라이브 리로드 서버가 포함
- 자동 재시작(Automatic Restart) 될 시 브라우저 자동 새로고침!!
- 지원 브라우저 : Chrome, Firefox, Safari
- 브라우저에 확장팩을 설치해야함
 - <http://livereload.com/extensions/>
- <https://chrome.google.com/webstore/detail/livereload/jnihajbhpnppcggbcgedagnkighmdlei>

실습

- [개발자 도구 테스트\(라이브 리로드\)](#)

개발자 도구 - 전역설정

- \$HOME/.config/spring-boot 디렉토리에 다음을 파일을 추가하여 개발자 도구 설정
 - spring-boot-devtools.properties
 - spring-boot-devtools.yaml
 - spring-boot-devtools.yml
- 개발자 도구를 사용하는 모든 애플리케이션에 설정 적용
- 예) 자동 재시작 트리거 파일 전역설정

```
spring.devtools.restart.trigger-file=.reloadtrigger
```

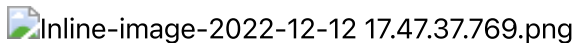
개발자 도구 - 원격 애플리케이션

- 원격에서 동작하는 애플리케이션에서 개발자 도구의 자동 재시작 기능을 사용할 수 있음
- 신뢰할 수 있는 네트워크(trusted network)나 SSL 통신에서 사용해야함
- 활성화하려면 빌드가 개발자 도구를 포함하고, spring.devtools.remote.secret 속성을 설정

```
<build>
<plugins>
  <plugin>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-maven-plugin</artifactId>
    <configuration>
      <excludeDevtools>>false</excludeDevtools>
    </configuration>
  </plugin>
</plugins>
</build>
```

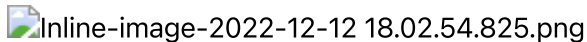
개발자 도구 - 원격 애플리케이션

- IDE 에서 다음의 순서로 설정
 - +버튼 : application 선택
 - 실행 main 클래스 : org.springframework.boot.devtools.RemoteSpringApplication
 - program 인자 : 서비스 주소 예) <http://localhost:8080>

Inline-image-2022-12-12 17.47.37.769.png

개발자 도구 - 원격 애플리케이션 클라이언트

- IDE 설정을 실행
- IDE 에서 클래스 수정 및 빌드시 원격 서버로 전송 및 자동 재시작 동작

Inline-image-2022-12-12 18.02.54.825.png

Update 2022.12.12 17:39 ☐

개발자 도구 테스트(자동 재시작)

목표

- Account 서비스가 spring-boot developer tools 의 자동 재시작이 동작하도록 수정하세요.

예상시작

- 10 분

Update 2022.12.12 17:42 ☐

개발자 도구 테스트(라이브 리로드)

목표

- 제공하는 코드가 spring-boot developer tools 의 라이브 리로드가 동작하도록 플러그인을 설치하고 테스트 하세요.

준비사항

- live reload extension download

- <http://livereload.com/extensions/>

예상시간

- 5분

Update 2022.12.13 01:57 ☐

8. Spring Boot Actuator

Spring Boot Actuator

- 상용화 준비(Production-Ready)기능을 위한 Spring Boot 모듈
- 실행 중인 애플리케이션을 관리하고 정보를 수집하고 상태를 점검하는 진입점 제공
- HTTP 또는 JMX 를 사용할 수 있음.

Actuator 설치

maven

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

gradle

```
dependencies {
    compile("org.springframework.boot:spring-boot-starter-actuator")
}
```

Spring Boot Actuator - Endpoint

- Actuator 엔드포인트로 spring boot 애플리케이션의 **모니터링** 및 **상호작용** 가능
- 스프링 부트는 다양한 빌트인 엔드포인트를 제공

| ID | 설명 |
|-------------|--|
| auditevent | 응용시스템의 모든 감사 이벤트 목록을 제공, AuditEventRepository 빈 필요 |
| beans | 애플리케이션의 모든 빈의 목록을 제공 |
| caches | 가능한 캐시를 노출 |
| conditions | 설정 및 자동설정 클래스를 평가한 조건의 목록과 조건의 부합 여부에 대한 이유를 제공 |
| configprops | 값이 설정된 모든 @ConfigurationProperties 의 목록을 제공 |

| ID | 설명 |
|------------|--|
| env | 스프링의 ConfigurableEnvironment 의 속성을 제공 |
| health | 애플리케이션의 health 정보를 제공 |
| httptrace | http 의 요청,응답 내용을 표시, (기본 설정으로 100개 까지만 제공, HttpTraceRepository 빈 필요) |
| info | 애플리케이션의 정보 제공 |
| shutdown | 애플리케이션의 섯다운 명령 |
| startup | startup 단계 데이터를 제공 (SpringApplication 을 BufferingApplicationStartup으로 설정 필요) |
| threaddump | 쓰레드 덤프를 실행 |

Spring Boot Actuator - Endpoint 활성화

- 기본설정으로 shutdown 을 제외한 모든 end point 는 활성화
- management.endpoint.{id}.enabled 속성으로 활성화/비활성화 설정

shutdown endpoint 활성화

```
management.endpoint.shutdown.enabled=true
```

```
management:
  endpoint:
    shutdown:
      enabled: true
```

shutdown endpoint Opt-in 설정

```
management.endpoints.enabled-by-default=false ## 모두 비활성화
management.endpoint.info.enabled=true ## info만 활성화
```

```
management:
  endpoints:
    enabled-by-default: false ## 모두 비활성화
  endpoint:
    info:
      enabled: true ## info만 활성화
```

Spring Boot Actuator - Endpoint 노출방식(JMX, Web) 설정

- acutator 는 민감한 정보를 노출하기 때문에 노출방식을 신중하게 설정해야 함
- Web은 health Endpoint 만 제공함

| ID | JMX | Web |
|------------------|-----|-----|
| auditevents | Yes | No |
| beans | Yes | No |
| caches | Yes | No |
| conditions | Yes | No |
| configprops | Yes | No |
| env | Yes | No |
| flyway | Yes | No |
| health | Yes | Yes |
| heapdump | N/A | No |
| httptrace | Yes | No |
| info | Yes | No |
| integrationgraph | Yes | No |
| jolokia | N/A | No |
| logfile | N/A | No |
| loggers | Yes | No |
| liquibase | Yes | No |
| metrics | Yes | No |
| mappings | Yes | No |
| prometheus | N/A | No |
| quartz | Yes | No |
| scheduledtasks | Yes | No |
| sessions | Yes | No |
| shutdown | Yes | No |
| startup | Yes | No |
| threaddump | Yes | No |

Spring Boot Actuator - Endpoint 노출방식(JMX, Web) 설정

- JMX는 모든 Endpoint 를 노출하고, Web은 health 만 노출하는 것이 기본 설정
- include, exclude 프로퍼티로 노출방식을 활성화 할 수 있음

| Property | 기본설정 |
|---|--------|
| management.endpoints.jmx.exposure.exclude | |
| management.endpoints.jmx.exposure.include | * |
| management.endpoints.web.exposure.exclude | |
| management.endpoints.web.exposure.include | health |

Spring Boot Actuator - Endpoint 노출방식(JMX, Web) 설정

- exclude 설정은 include 설정보다 우선한다.
- 예) health, info 만 JMX에서 노출

```
management.endpoints.jmx.exposure.include=health,info
```

- 예) env, bean 를 제외한 모든 Endpoint를 web에서 노출

```
management.endpoints.web.exposure.include=*
management.endpoints.web.exposure.exclude=env,bean
```

Spring Boot Actuator 보안

Spring Security 설정

- spring-security가 클래스패스에 존재하면 health를 제외한 모든 Endpoint는 기본 자동설정기능에 의해 보호된다.
- WebSecurityConfigurerAdapter 또는 SecurityFilterChain 빈을 설정하여 기본 자동설정을 제거하고 보안설정을 정의할 수 있다.

```
@Configuration(proxyBeanMethods = false)
public class MySecurityConfiguration {
```

```
    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws
    Exception {
        http.securityMatcher(EndpointRequest.toAnyEndpoint());
        http.authorizeHttpRequests((requests) ->{
            requests.anyRequest().permitAll();
            return http.build();
        });
    }
}
```

```
}
```

EndPoint 사용자 정의

- 다음의 애너테이션을 사용하여 사용자정의 EndPoint 를 노출할 수 있다.

@Endpoint: Endpoint로 노출할 빈에 설정

- @WebEndpoint: HTTP Endpoint로만 노출할 때
- @JmxEndpoint: JMX Endpoint로만 노출할 때

@ReadOperation, @WriteOperation, @DeleteOperation

- HTTP 의 GET, POST, DELETE 메소드

@EndpointWebExtension, @EndpointJmxExtension

- 이미 존재하는 Endpoint에 기술 전용 오퍼레이션을 추가할 때 사용

EndPoint 사용자 정의 예

- Counter를 관리하는 Endpoint 예

```
@Component
@Endpoint(id ="counter")
public class CounterEndpoint {
    private final AtomicLong counter = new AtomicLong();
```

```
// curl -X GET http://localhost:8080/actuator/counter @ReadOperation public Long read() { return
counter.get(); }
```

```
// curl -X POST -H"Content-Type: application/json" -d '{"delta":100}' http://localhost:8080/actuator/counter
@WriteOperation public Long increment(@Nullable Long delta) { if (delta == null) { return
counter.incrementAndGet(); } return counter.addAndGet(delta ); }
```

```
// curl -X DELETE http://localhost:8080/actuator/counter @DeleteOperation public Long reset() {
counter.set(0); return counter.get(); } }
```

EndPoint 사용자 정의

- 이미 존재하는 Endpoint에 특정 기술에서 동작하는 Endpoint를 추가하고 싶으면 @EndpointWebExtension, @EndpointJmxExtension을 사용한다.

```
@EndpointWebExtension(endpoint = CounterEndpoint.class)
@Component
public class CounterWebEndPoint {
    private final CounterEndpoint target;
    public CounterWebEndPoint(CounterEndpoint target) {
        this.target = target;
    }
```

```
@WriteOperation
public WebEndpointResponse<Long>increment(@Nullable Long delta) {
    returnnew WebEndpointResponse<>(target.increment(delta));
}
}
```

실습

- [Acutator EndPoint](#) 추가 실습

Health Endpoint

- 애플리케이션의 **정상동작 정보**를 제공한다.
- ApplicationContext 내의 HealthContributor 타입의 빈을 모두 활용해서 정보를 제공한다.
- HealthContributor 는 HealthIndicator 나 CompositeHealthContributor의 형태로 사용
 - HealthIndicator : 실제 Health 정보 제공
 - CompositeHealthIndicator : HealthContributor 들의 조합정보를 제공
- **management.endpoint.health.show-details=always** 를 설정하면 각각의 HealthContributor 상세 정보를 볼 수 있다.

```
$ http://localhost:8080/actuator/health
```

Spring Boot의 기본 HealthIndicators

- Auto Configuration에 의해서 동작여부 결정


| 이름 | 기본설정 |
|--------------------------------|--------------------------------|
| CassandraDriverHealthIndicator | 카산드라 데이터베이스 상태 체크 |
| CouchbaseHealthIndicator | 카우치베이스 클러스터 상태 체크 |
| DiskSpaceHealthIndicator | 디스크 공간 체크 |
| DataSourceHealthIndicator | DataSource에서 커넥션을 얻을 수 있는 지 체크 |
| RedisHealthIndicator | 레디스 서버의 상태 체크 |

Kubernetes 용 HealthIndicators

| Key | Name | Description |
|----------------|-------------------------------|----------------|
| livenessstate | LivenessStateHealthIndicator | "Liveness" 상태 |
| readinessstate | ReadinessStateHealthIndicator | "Readiness" 상태 |


LivenessProbe

- 운영중에 Pod 의 LivenessProbe 을 점검하여 실패하면 Pod 삭제 후 다시 생성

 Inline-image-2022-12-12 22.46.57.982.png

ReadinessProbe

- 운영 중에 Pod 의 ReadinessProbe 을 점검하여 서비스에서 제외한다.

 Inline-image-2022-12-12 22.49.35.487.png

커스텀 HealthIndicator 작성하기

- 커스텀 health 정보를 제공하려면 HealthIndicator 인터페이스를 구현한다.
- health() 메소드에서 Health 응답을 반환한다.

```
@Component
public class MyHealthIndicator implements HealthIndicator {
```

```
@Override public Health health() { int errorCode = check(); if (errorCode != 0) { return
Health.down().withDetail("Error Code", errorCode).build(); } return Health.up().build(); }
```

```
private int check() { // perform some specific health check return ... }
```

```
}
```

실습

- [Actuator Custom HealthIndicator 작성하기](#)

info EndPoint

- 애플리케이션의 정보를 제공한다.
- ApplicationContext 내의 InfoContributor 타입의 빈을 모두 활용해서 정보를 제공한다.

```
http://localhost:8080/actuator/info
```

info EndPoint - EnvironmentInfoContributor

- info.* 형식의 모든 환경변수 정보 제공 (spring boot 2.6 이후 부터 기본 비활성화)
- application.properties 설정 추가

```
management.info.env.enabled=true
info.edu.springboot.version=10.1.1
info.edu.springboot.instructor=manty
```

- info endpoint 호출 결과

```
{
  • "edu" :
    {
      • "springboot" :
        {
          ▪ "version" :
            "10.1.1",
          ▪ "instructor" :
            "manty"
        }
      },
  • "app" :
    {
      • "java" :
        {
          ▪ "source" :
            "11"
        }
      }
    }
}
```

info EndPoint - GitInfoContributor

- 클래스 패스상의 git.properties 정보 제공, 실행 중인 서비스의 git 정보 확인용
- maven, gradle 설정 필요

maven 설정

```
<build>
<plugins>
  ...
<plugin>
  <groupId>pl.project13.maven</groupId>
  <artifactId>git-commit-id-plugin</artifactId>
</plugin>
</plugins>
</build>
```

gradle 설정

```
plugins {
  id"com.gorylenko.gradle-git-properties" version"1.5.1"
}
```


- info endpoint 호출 결과

```
{
  • "git" :
    {
      ◦ "branch" :
        "master",
      ◦ "commit" :
        {
          ▪ "id" :
            "077a397",
          ▪ "time" :
            "2022-02-01T05:12:05Z"
        }
    }
}
```

info EndPoint - BuildInfoContributor

- 클래스 패스의 META-INF/build-info.properties 파일 정보 제공
- maven, gradle 설정 필요

maven 설정

```
<plugin>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-maven-plugin</artifactId>
<executions>
<execution>
  <goals>
    <goal>build-info</goal>
  </goals>
</execution>
</executions>
</plugin>
```

gradle 설정

```
springBoot {
    buildInfo()
}
```

- info endpoint 호출 결과

```
{
  • "build" :
    {
      ◦ "artifact" :
        "student",
      ◦ "name" :
        "student",
      ◦ "time" :
        "2022-02-01T07:07:41.030Z",
      ◦ "version" :
        "0.0.1-SNAPSHOT",
      ◦ "group" :
        "com.nhn.edu.springboot"
    }
}
```

info EndPoint - InfoContributor 사용자 정의

- InfoContributor 인터페이스의 구현체를 개발하여 빈으로 등록합니다.

```
@Component
public class ExampleInfoContributor implements InfoContributor {
    @Override
    public void contribute(Info.Builder builder) {
        builder.withDetail("example", Map.of("key", "value"));
    }
}
```

- 사용자 정의 InfoContributor 호출 결과

```
{
  • "example" :
    {
      ◦ "key" :
        "value"
    }
}
```

실습

- [Actuator Custom InfoContributor 작성하기](#)

Endpoint 경로변경

- Spring Boot Actuator 의 기본 경로는 /actuator 이다.
- management.endpoints.web.base-path 속성을 변경하여 경로를 변경할 수 있다.

actuator endpoint 경로변경

```
management.endpoints.web.base-path=/actuator2      # 2.x
management.context-path=/actuator2                # 1.x : Set
/actuator
```

Endpoint Port 변경

- Spring Boot Actuator 의 기본 포트는 서비스 포트와 동일하다.
- management.server.port 속성을 변경하여 포트를 변경할 수 있다.

```
management.server.port=8888
```

prometheus Endpoint

prometheus

- prometheus(<https://prometheus.io> 라는 시계열 데이터베이스에 데이터를 제공
- micrometer-registry-prometheus 라이브러리 의존성을 추가해야 함
- <http://localhost:8080/actuator/prometheus> 경로로 호출

```
<dependency>
<groupId>io.micrometer</groupId>
<artifactId>micrometer-registry-prometheus</artifactId>
</dependency>
```

prometheus 설치

- Docker 로 설치

start.sh

```
#!/bin/bash

docker run \
  --platform=linux/arm64 \
  -d \
  --name prometheus \
  -p 9090:9090 \
  -v $(pwd)/config:/etc/prometheus \
```

```
-v $(pwd)/data:/prometheus:rw \
prom/prometheus:v2.33.4
```

stop.sh

```
#!/bin/bash

docker stop prometheus
docker rm prometheus
```

config 파일 생성

- config/prometheus.yml 파일을 다음과 같이 생성합니다.

```
# my global config
global:
  scrape_interval: 15s # Set the scrape interval to every 15 seconds.
  Default is every 1 minute.
  evaluation_interval: 15s # Evaluate rules every 15 seconds. The default is
  every 1 minute.
  # scrape_timeout is set to the global default (10s).
```

Alertmanager configuration alerting: alertmanagers: -static_configs: -targets: # - alertmanager:9093

Load rules once and periodically evaluate them according to the global 'evaluation_interval'. rule_files: # - "first_rules.yml" # - "second_rules.yml"

A scrape configuration containing exactly one endpoint to scrape: # Here it's Prometheus itself.

scrape_configs: # The job name is added as a label `job=<job_name>` to any timeseries scraped from this config. -job_name:'prometheus'

metrics_path defaults to '/metrics' # scheme defaults to 'http'.

static_configs: -targets:['localhost:9090']

-job_name:'student' metrics_path:actuator/prometheus static_configs: -targets:['\${pcipaddress}:8080']

data 디렉토리 생성


```
$ mkdir data
```

조회

- hikaricp_connections_active

Grafana

- Grafana 로 prometheus 정보를 시각화
- 성능 추이를 추적하여 인프라 scale up 또는 scale out 지표로 사용

 Inline-image-2022-12-13 01.35.06.108.png

Update 2022.12.12 22:33 ☐

Acuator EndPoint 추가 실습

목표

- Account 시스템에 다음의 Actuator Endpoint 가 동작하도록 @Endpoint 생성 코드를 추가하세요.
- curl -XGET <http://localhost:8080/actuator/counter>
- curl -X POST -H"Content-Type: application/json" -d'{"delta":100}'
<http://localhost:8080/actuator/counter>
- curl -X DELETE <http://localhost:8080/actuator/counter>

예상시간

- 10분

Update 2022.12.13 01:12 ☐

Actuator Custom HealthIndicator 작성하기

목표

- 사용자 정의 HealthIndicator 를 개발합니다.

기능

- 다음 api 를 호출하면 actuator health 가 fail 이 되도록 HealthIndicator 를 개발합니다.

```
$ curl -XPost http://localhost:8080/management/fail
```

예상시간

- 15분

Update 2022.12.13 01:27 ☐

Actuator Custom InfoContributor 작성하기

목표

- Account 시스템에 다음의 Actuator Endpoint 에 AuthorInfoContributor 를 구성하여 info endpoint에 정보를 추가하세요.

예상시간

- 10분

AuthorInfoContributor 호출 결과

```
{
  • "author" :
    {
      ◦ "name" :
        "${자신의 이름}"
    }
}
```

Update 2022.12.13 16:22 ☐

promethues 설치 및 설정

목표

- Spring boot Actuator 의 Prometheus EndPoint 를 활성화 합니다.
- prometheus 를 활용하여 데이터를 조회합니다.
 - prometheus 에서 process_cpu_usage 를 조회합니다.

예상시간

- 15 분

Spring Boot Testing 설치

- spring boot는 테스트를 위한 다양한 애너테이션과 유틸리티를 제공한다.
- 테스트 지원은 spring-boot-test, spring-boot-test-autoconfigure 모듈로 제공된다.
- 개발자는 spring-boot-starter-test 의존성을 추가하여 설치할 수 있다.

maven

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

gradle

```
dependencies {
    testCompile("org.springframework.boot:spring-boot-starter-test")
}
```

spring-boot-starter-test로 제공하는 라이브러리

| 구분 | 설명 |
|--------------------------------|---|
| JUnit5 | Java 애플리케이션 단위 테스트의 산업계 표준(de-facto standard) |
| Spring Test & Spring Boot Test | Spring Boot 애플리케이션 테스트 지원용 유틸리티와 통합테스트 지원도구 |
| AssertJ | Assertion 라이브러리 |
| Hamcrest | Matcher 객체용 라이브러리 |
| Mockito | Mocking framework |
| JSONassert | JSON Assertion 용 |

Spring Boot Testing 애너테이션

@SpringBootTest

- @SpringBootTest를 사용하면 spring boot의 기능을 사용하면서 통합 테스트 할 때 필요합니다.
- 실제 애플리케이션 기동 시와 거의 유사 하게 전체 빈 설정이 로딩됩니다.

주의

- JUnit 4: @RunWith(SpringRunner.class) 추가
- JUnit 5 : @ExtendWith(SpringExtension.class) 는 이미 포함되어 있음

Spring Boot Testing 애너테이션

@SpringBootTest

- webEnvironment를 설정해서 서버를 실행할 수 있다.

SpringBootTest.webEnvironment

| 구분 | 설명 | 비고 |
|--------------|---|----------------------|
| MOCK | MockMvc로 테스트 가능 | 기본 |
| RANDOM_PORT | Embedded WAS 실행, 임의의 포트로 실행, (rollback 동작하지 않음) | @LocalServerPort로 주입 |
| DEFINED_PORT | Embedded WAS 실행, 설정한 포트로 실행, (rollback 동작하지 않음) | server.port 속성으로 결정 |
| NONE | WEB 이 아닌 일반 서비스 테스트용 | - |

MOCK environment

- MOCK 환경에서는 서버를 실행하지 않기 때문에 MockMvc나 WebTestClient 로 테스트 해야 한다.

```
@SpringBootTest
@AutoConfigureMockMvc
class MyMockMvcTests {
    @Autowired private MockMvc mockMvc;
```

```
@Test
void testWithMockMvc(@Autowired MockMvc mvc) throws Exception {

    mvc.perform(get("/")).andExpect(status().isOk()).andExpect(content().string("Hello World"));
}

// If Spring WebFlux is on the classpath, you can drive MVC tests with a
WebTestClient
@Test
void testWithWebTestClient(@Autowired WebTestClient webClient) {
    webClient
        .get().uri("/")
        .exchange()
        .expectStatus().isOk()
        .expectBody(String.class).isEqualTo("Hello World");
}
```

```
}
```

[시연] Student 시스템 통합테스트

- 다음과 같이 테스트 클래스를 생성합니다.

```
import com.fasterxml.jackson.databind.ObjectMapper;
import com.nhnacademy.edu.springboot.student.Student;
import org.junit.jupiter.api.MethodOrderer;
import org.junit.jupiter.api.Order;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.TestMethodOrder;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.http.MediaType;
import org.springframework.test.web.servlet.MockMvc;
```



```
importstatic org.hamcrest.Matchers.equalTo; importstatic
org.springframework.test.web.servlet.request.MockMvcRequestBuilders.; importstatic
org.springframework.test.web.servlet.result.MockMvcResultMatchers.;
```

```
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.MOCK) @AutoConfigureMockMvc
@TestMethodOrder(MethodOrderer.OrderAnnotation.class) classStudentControllerTest { }
```

[시연] Student 시스템 통합테스트 (GET /students)

- MockMvc 를 주입받아 /students 를 호출합니다.
- jsonPath 를 이용하여 json 경로상의 값을 비교합니다.

```
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.MOCK)
@AutoConfigureMockMvc
@TestMethodOrder(MethodOrderer.OrderAnnotation.class)
classStudentControllerTest {
```

```
@Autowired private MockMvc mockMvc;
```

```
@Test @Order(1) voidtestGetStudents()throws Exception{ mockMvc.perform(get("/students"))
.andExpect(status().isOk()) .andExpect(content().contentType(MediaType.APPLICATION_JSON))
.andExpect(jsonPath("$[0].name", equalTo("manty"))); }
```

[시연] Student 시스템 통합테스트 (GET /students/{id})

```
@Test
@Order(2)
voidtestGetStudent()throws Exception{
```

```
mockMvc.perform(get("/students/{id}",1L)) .andExpect(status().isOk())
.andExpect(content().contentType(MediaType.APPLICATION_JSON)) .andExpect(jsonPath("$.name",
equalTo("manty"))); }
```

[시연] Student 시스템 통합테스트 (POST /students)

```
@Test
@Order(3)
voidtestCreateStudent()throws Exception{
    ObjectMapper objectMapper =new ObjectMapper();
    Student zbum =new Student(3L,"zbum1",100);
    mockMvc.perform(post("/students")
        .content(objectMapper.writeValueAsString(zbum))
        .contentType(MediaType.APPLICATION_JSON))
        .andExpect(status().isCreated()); }
```

```

        .andExpect(content().contentType(MediaType.APPLICATION_JSON))
        .andExpect(jsonPath("$.name", equalTo("zbum1")));
    }

```

[시연] Student 시스템 통합테스트 (DELETE /students)

```

@Test
@Order(4)
void deleteStudent() throws Exception {
    this.mockMvc.perform(delete("/students/{id}", 3L))
        .andExpect(status().isOk())

    .andExpect(content().contentType(MediaType.APPLICATION_JSON))
        .andExpect(jsonPath("$.result", equalTo("OK")));
}

```

실습

- [SpringBootTest 사용하기 - 통합테스트](#)

RANDOM_PORT, DEFINED_PORT environment

- WebFlux에서 서버를 실행하는 환경에서는 테스트하려면 WebClient를 사용 해야 한다.

```

@SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)
class MyRandomPortWebTestClientTests {

```

```

@Test
void exampleTest(@Autowired WebTestClient webClient) {
    webClient
        .get().uri("/")
        .exchange()
        .expectStatus().isOk()
        .expectBody(String.class).isEqualTo("Hello World");
}

```

```

}

```

RANDOM_PORT, DEFINED_PORT environment

- WebFlux를 사용할 수 없는 환경에서는 TestRestTemplate을 사용할 수 있다.

```
@SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)
class MyRandomPortTestRestTemplateTests {
```

```
@Test void exampleTest(@Autowired TestRestTemplate restTemplate) { String body =
restTemplate.getForObject("/", String.class); assertThat(body).isEqualTo("Hello World"); }

}
```

[시연] @SpringBootTest(RANDOM_PORT)

- Student 시스템 통합테스트

```
import com.nhnacademy.edu.springboot.student.Student;
import org.junit.jupiter.api.MethodOrderer;
import org.junit.jupiter.api.Order;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.TestMethodOrder;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.boot.test.web.client.TestRestTemplate;
import org.springframework.core.ParameterizedTypeReference;
import org.springframework.http.*;
import java.util.List;
import static org.assertj.core.api.Assertions.assertThat;
@SpringBootTest(webEnvironment =
SpringBootTest.WebEnvironment.RANDOM_PORT)
@TestMethodOrder(MethodOrderer.OrderAnnotation.class)
class StudentControllerTest {
}
```

[시연] @SpringBootTest(RANDOM_PORT)

Student 시스템 통합테스트 (GET /students)

```
@Autowired
private TestRestTemplate testRestTemplate;
```

```
@Test @Order(1) void testGetStudents() throws Exception { HttpHeaders headers = new HttpHeaders();
headers.setAccept(List.of(MediaType.APPLICATION_JSON)); HttpEntity<Student> entity = new
HttpEntity<>(headers); ResponseEntity<List<Student>> exchange = testRestTemplate.exchange(
"/students", HttpMethod.GET, entity, new ParameterizedTypeReference<List<Student>>() { });
```

```
assertThat(exchange.getBody())
    .contains(<span class="hljs-keyword">new</span> Student(<span
```

```
class="hljs-number">1L</span>,<span class="hljs-string">"manty"</span>,<span class="hljs-number">100</span>));
```

```
}
```

Student 시스템 통합테스트 (GET /students/{id})

```
@Test
@Order(2)
void testGetStudent() throws Exception {
    ResponseEntity<Student> result = testRestTemplate.getForEntity(
        "/students/{id}",
        Student.class,
        1L);
}
```

```
assertThat(result.getBody())
    .isEqualTo(<span class="hljs-keyword">new</span> Student(<span class="hljs-number">1L</span>,<span class="hljs-string">"manty"</span>,<span class="hljs-number">100</span>));
```

```
}
```

Student 시스템 통합테스트 (POST /students)

```
@Test
@Order(3)
void testCreateStudent() throws Exception {
    Student zbum = new Student(3L, "zbum1", 100);
    ResponseEntity<Student> result = testRestTemplate.postForEntity(
        "/students",
        zbum,
        Student.class);
}
```

```
assertThat(result.getBody())
    .isEqualTo(zbum);
```

```
}
```

Student 시스템 통합테스트 (DELETE /students)

```

@Test
@Order(4)
void testDeleteStudent() throws Exception {
    testRestTemplate.delete(
        "/students/{id}",
        3L);
}

```

실습

- [SpringBootTest 사용하기 - 통합테스트 RANDOM_PORT](#)

단위 테스트 - Mocking Beans

- 테스트환경에서 사용할 수 없는 리모트 서비스등을 시뮬레이션 하도록 특정 컴포넌트를 Mocking
- @MockBean으로 빈을 생성하거나 빈을 대체할 수 있다.

```

@SpringBootTest
class MyTests {

```

```

@Autowired private Reverser reverser;

```

```

@MockBean //RemoteService 를 대체하는 예제 private RemoteService remoteService;

```

```

@Test void exampleTest() { given(this.remoteService.getValue()).willReturn("spring"); String reverse
= this.reverser.getReverseValue(); // Calls injected RemoteService assertThat(reverse).isEqualTo("gnirps"); }
}

```

[시연] Student 시스템 - @MockBean

```

@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.MOCK)
@AutoConfigureMockMvc
class StudentControllerTest {

```

```

@Autowired private MockMvc mockMvc;

```

```

@MockBean private StudentRepository studentRepository;

```

```

@Test void testGetStudents() throws Exception { given(studentRepository.findAll()).willReturn(List.of(new
Student(100L, "AA", 90)));

```

```

mockMvc.perform(get("/students")
    .andExpect(status().isOk())
    .andExpect(content().contentType(MediaType.APPLICATION_JSON))

```

```

        .andExpect(jsonPath(<span class="hljs-string">"${0}.name"
</span>, equalTo(<span class="hljs-string">"AA"</span>)));
    }

```

[실습] controller test with @MockBean

- [SpringBootTest 사용하기 - Controller 테스트 with @MockBean](#)

Spying Beans

- 테스트환경에서 이미 존재하는 빈을 래핑하여 특정 메소드가 다른 동작을 하도록 설정할 수 있다.

```

class MyTests {

```

```

@SpyBean private RemoteService remoteService;

```

```

@Autowired private Reverser reverser;

```

```

@Test public void exampleTest() {
    given(this.remoteService.someCall()).willReturn("mock");
    String reverse = reverser.reverseSomeCall();
    assertThat(reverse).isEqualTo("kcom");
    then(this.remoteService).should(times(1)).someCall();
}

```

[시연] Student 시스템 @SpyBean

```

@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.MOCK)
@AutoConfigureMockMvc
@TestMethodOrder(MethodOrderer.OrderAnnotation.class)
class StudentControllerTest {

```

```

@Autowired private MockMvc mockMvc;

```

```

@SpyBean private StudentService studentService;

```

```

@Test @Order(1) void testGetStudents() throws Exception {
    given(studentService.getStudents())
    .will(invocation -> {
        System.out.println("Spy!!");
        return List.of(new Student(100L, "AA", 90));
    });

```

```

        mockMvc.perform(get(<span class="hljs-string">"/students"</span>))
            .andExpect(status().isOk())
            .andExpect(content().contentType(MediaType.APPLICATION_JSON))
            .andExpect(jsonPath(<span class="hljs-string">"${0}.name"
</span>, equalTo(<span class="hljs-string">"AA"</span>)));
    }
}

```

```

}

```

Auto-configured JSON Tests

- 자동설정 환경에서 객체의 JSON 직렬화, 역직렬화를 테스트하기 위하여 @JsonTest 을 사용합니다.
- AssertJ 기반의 테스트 지원을 제공하기 때문에 객체-JSON 매핑의 결과를 검증할 수 있습니다.
- JacksonTester, GsonTester, JsonbTester, BasicJsonTester 클래스를 각각의 JSON 라이브러리 헬퍼로 사용할 수 있습니다.

Auto-configured JSON Tests

- JacksonTester 사용예

```
@JsonTest
class MyJsonTests {
```

```
@Autowired private JacksonTester<VehicleDetails> json;
```

```
@Test void serialize() throws Exception { VehicleDetails details = new VehicleDetails("Honda", "Civic");
assertThat(this.json.write(details)).isEqualToJson("expected.json");
assertThat(this.json.write(details)).hasJsonPathStringValue("@.make");
assertThat(this.json.write(details)).extractingJsonPathStringValue("@.make").isEqualTo("Honda"); }
```

```
@Test void deserialize() throws Exception { String content = "{\"make\":\"Ford\",\"model\":\"Focus\"}";
assertThat(this.json.parse(content)).isEqualTo(new VehicleDetails("Ford", "Focus"));
assertThat(this.json.parseObject(content).getMake()).isEqualTo("Ford"); }
```

```
}
```

Auto-configured Spring MVC Tests

- Spring MVC의 Controller를 테스트 하기 위해서는 @WebMvcTest 를 사용한다.
- @WebMvcTest를 사용하면@Controller, @ControllerAdvice, @JsonComponent, Converter, GenericConverter, Filter, HandlerInterceptor, WebMvcConfigurer, WebMvcRegistrations, HandlerMethodArgumentResolver 등 만 스캔한다.
- 테스트에서 다른 컴포넌트를 스캔하고 싶다면 테스트코드에 @Import 로 직접 설정해 주어야 한다.

Auto-configured Spring MVC Tests

- @WebMvcTest 를 사용하면 MockMvc 객체를 얻을 수 있다.

```
@WebMvcTest(UserVehicleController.class)
class MyControllerTests {
```

```
@Autowired private MockMvc mvc;
```

```
@MockBean private UserVehicleService userVehicleService;
```

```
@Test void testExample() throws Exception { given(this.userService.getVehicleDetails("sboot"))
    .willReturn(new VehicleDetails("Honda", "Civic"));
    this.mvc.perform(get("/sboot/vehicle").accept(MediaType.TEXT_PLAIN)) .andExpect(status().isOk())
    .andExpect(content().string("Honda Civic")); } }
```

Auto-configured Spring MVC Tests

- HtmlUnit 과 Selenium 을 사용한다면 HtmlUnit의 WebClient 도 사용할 수 있다..

```
@WebMvcTest(UserVehicleController.class)
class MyHtmlUnitTests {
```

```
@Autowired private WebClient webClient;
```

```
@Test void testExample() throws Exception { HtmlPage page = this.webClient.getPage("/test");
    assertThat(page.getBody().getTextContent()).isEqualTo("Honda Civic"); }
```

```
}
```

[시연] @SpyBean

- Student 시스템 @WebMvcTest

```
@WebMvcTest(SystemController.class)
class SystemControllerTest {
```

```
@Autowired MockMvc mockMvc;
```

```
@MockBean SystemProperties systemProperties;
```

```
@Test void testGetAuthor() throws Exception { given(systemProperties.getAuthor()) .willReturn("ABCDEFGH");
```

```
    mockMvc.perform(get("<span class='hljs-string'>/system/author"
    </span>))
        .andExpect(status().isOk())
        .andExpect(content().contentType(MediaType.APPLICATION_JSON))
        .andExpect(jsonPath("<span class='hljs-string'>$.author"
    </span>, equalTo("<span class='hljs-string'>ABCDEFGH"
    </span>)));
}
```

```
}
```

Auto-configured Data JPA Tests

- JPA를 테스트 하기 위해서는 @DataJpaTest 를 사용한다.

- @DataJpaTest 를 사용하면 @Entity, Repository 만 스캔한다.
- 기본적으로 테스트 이후에 수정된 정보는 모두 롤백한다.
- H2와 같은 인메모리 데이터베이스가 클래스 패스에 존재하면 사용하지만 실제 데이터베이스에서 테스트하려면 @AutoConfigureTestDatabase(replace = Replace.NONE) 을 설정해야 한다.

Auto-configured Data JPA Tests

- @DataJpaTest 를 사용하면 TestEntityManager 객체를 얻을 수 있다.

```
@DataJpaTest
class MyRepositoryTests {
```

```
@Autowired private TestEntityManager entityManager;
```

```
@Autowired private UserRepository repository;
```

```
@Test void testExample() throws Exception { this.entityManager.persist(new User("sboot", "1234")); User
user = this.repository.findByUsername("sboot"); assertThat(user.getUsername()).isEqualTo("sboot");
assertThat(user.getEmployeeNumber()).isEqualTo("1234"); }
```

[시연] @DataJpaTest

- Student 시스템 @DataJpaTest

```
@DataJpaTest
@AutoConfigureTestDatabase(replace =
AutoConfigureTestDatabase.Replace.NONE)
class StudentRepositorySliceTest {
    @Autowired
    TestEntityManager entityManager;
```

```
@Autowired StudentRepository studentRepository;
```

```
@Test void testFindAll() { Student manty = new Student(10L, "Manty", 100); entityManager.merge(manty);
```

```
        Student student = studentRepository.findById(<span class="hljs-number">10L</span>).orElse(<span class="hljs-keyword">null</span>);
        assertThat(student).isEqualTo(manty);
    }
```

```
}
```

[실습] Account 시스템 @DataJpaTest

- [@DataJpaTest 사용하기](#)

Test Utilities

- ConfigDataApplicationContextInitializer
 - application.properties 를 읽어들이는데 사용한다.
 - @SpringBootTest가 제공하는 모든 기능이 필요 없을때 사용한다.
- ConfigDataApplicationContextInitializer 를 단독으로 사용하면 application.properties 내용을 스프링 Environment 에 로드 하는 것만 수행한다.

```
@ContextConfiguration(classes = Config.class, initializers =
ConfigDataApplicationContextInitializer.class)
class MyConfigFileTests {
```

```
// ...
```

```
}
```

OutputCapture

- System.out, System.err 으로 출력하는 내용을 잡아낼 수 있다..
- 수정할 수 없는 라이브러리의 결과를 확인할 때 사용할 수 있다.

```
@ExtendWith(OutputCaptureExtension.class)
class MyOutputCaptureTests {
```

```
@Test void testName(CapturedOutput output) { System.out.println("Hello World!");
assertThat(output).contains("World"); }
}
```

Update 2022.12.13 02:22 ☐

SpringBootTest 사용하기 - 통합테스트

목표

- Account 시스템의 모든 API 를 통합테스트 하는 테스트 코드를 작성합니다.

예상시간

- 15 분

Update 2022.12.13 11:44 ☐

SpringBootTest 사용하기 - 통합테스트 RANDOM_PORT

목표

- Account 시스템의 Controller 메소드를 @SpringBootTest(RANDOM_PORT) 로 통합테스트 작성하세요.

예상시간

- 10 분

Update 2022.12.13 11:44 ☐

SpringBootTest 사용하기 - Controller 테스트 with @MockBean

작업내용

- Account 시스템의 Controller 메소드를 @MockBean 을 사용하여 통합테스트를 작성하세요.(4개 메소드)

예상시간

- 15분

Update 2022.12.13 12:30 ☐

@DataJpaTest 사용하기

목표

- Account 시스템의 Repository 메소드를 @DataJpaTest 을 사용하여 통합테스트를 작성하세요.
(findAll, findById, save, delete)

예상시간

- 10분

Update 2022.12.13 17:41 ☐

10. Custom Spring Boot Starter

Spring Boot Starter 의 구성

자동설정 모듈

- 기능을 사용하기 위한 자동설정(auto-configure) 코드와 확장을 위한 설정키의 집합

Starter 모듈

- 필요한 라이브러리 집합을 제공하기 위한 starter

자동설정(auto-configure) 모듈

- 자동설정 모듈에는 라이브러리를 바로 사용할 수 있는 자동설정과 설정키 정의,
- 콜백 인터페이스를 포함한다.
- Spring Boot의 annotation processor를 사용하여 메타데이터 파일을 생성할 수 있다.

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

```
<optional>true</optional>
</dependency>
```

Starter 모듈

- starter 모듈에는 java 코드를 포함하지 않는다.
- 라이브러리 의존성 만을 제공하도록 구현한다.

Starter 명명법

- 적절한 네임스페이스를 제공해야 함
- 직접 작성하는 Starter에 spring-boot 라는 네임스페이스를 사용하지 말것
- \${기능이름}-spring-boot-starter 의 형식을 권장함
- 예) MyBatis-Spring-Boot-Starter

설정키의 구성

- 설정키를 제공하고자 한다면, 유일한 네임스페이스를 사용해야 한다.
- Spring Boot 가 사용하는 네임스페이스를 사용하지 말것 (server, management, spring 등)
- 가능하면 고유명사를 네임스페이스로 사용할 것

Update 2023.02.09 16:44 ☐

dooray-spring-boot-starter 개발

목표

- dooray 메신저에 메시지 발송용 dooray-spring-boot-starter 개발
- 메신저 대화방에 다음의 포맷으로 발송

```
name : 이름
message : 하고 싶은말.. (재미있을 수록 좋음)
```

예상시간

- 20분

제공코드

- <https://github.com/edu-springboot/edu-doorayclient-springboot-workshop.git>

dooray-hook-sender

- dooray 메신저에 메시지를 발송하는 일반 java 라이브러리

```
<dependency>
<groupId>com.nhn.dooray.messenger</groupId>
<artifactId>dooray-hook-sender</artifactId>
```

```
<version>1.2.0-RELEASE</version>
</dependency>
```

dooray-spring-boot-autoconfigure

- 자동설정 모듈

dooray-spring-boot-starter

- starter 모듈

dooray-spring-boot-starter-application

- dooray-spring-boot-starter를 사용하는 애플리케이션

결과물



hook url

- 2022-12-광주
 - <https://hook.dooray.com/services/3204376758577275363/3428358313478707480/onLmqawGQ768s5ZVBFZ4jw>
- 2023-02-09 NHN
 - https://hook.dooray.com/services/3036349505739914786/3469734536114851559/VvGYlw3aTEemVjB2_CFxUw