Spring Boot Basic

스프링 부트 기초

스프링 부트의 로깅과 빌드

한국기술교육대학교
온라인평생교육원

학습내용

- 스프링 부트 로깅
- 스프링 부트 빌드

학습목표

- 스프링 부트가 제공하는 로깅 기능을 이해하고 커스터마이징 할 수 있다.
- 스프링 부트로 개발한 프로젝트를 배포 가능한 파일로 패키징하고 독립적으로 실행할 수 있다.

스프링 부트 로깅

스프링 부트 로깅

1 스프링 부트의 로깅

1 SLF4J(Simple Logging Facade for Java)

로그 메시지

- 애플리케이션을 운용하다 문제가 발생하면 가장 먼저 확인하는 것

스프링 부트 로깅

1 스프링 부트의 로깅

1 SLF4J(Simple Logging Facade for Java)

- 스프링 부트는 로그 관리를 위해 SLF4J를 사용하는데, SLF4J는 프레임워크가 아닌 단순한 퍼사드(façade)에 불과함

퍼사드	복잡한 서브시스템을 쉽게 사용할 수 있도록 일관된 인터페이스를 제공(GoF 패턴)
SLF4J	구체적인 로깅 프레임워크를 몰라도 로그를 처리할 수 있으며, 로그 프레임워크를 쉽게 교체할 수 있음

스프링 부트 로깅

스프링 부트 로깅

1 스프링 부트의 로깅

〈2〉 로깅 프레임워크

- 스프링 부트는 SLF4J라는 퍼사드를 통해 궁극적으로 LogBack이라는 로그 프레임워크를 사용함

LogBack

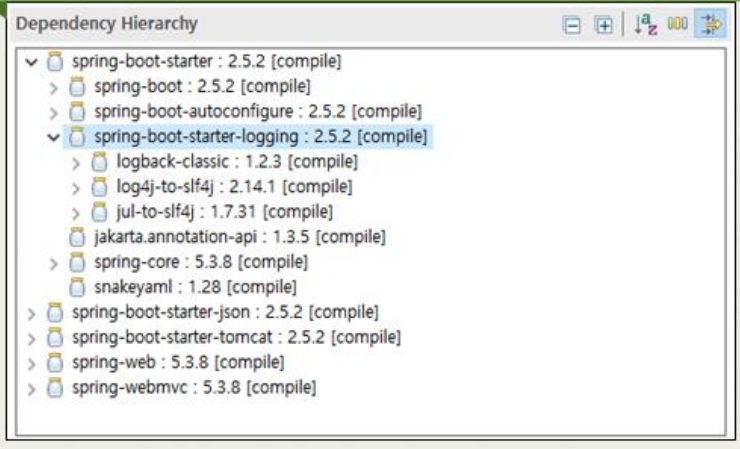
- Log4j 개발자인 Ceki Gulku가 기존의 Log4j의 성능과 기능을 향상시킨 것

스프링 부트 로깅

1 스프링 부트의 로깅

〈2〉 로깅 프레임워크

spring-boot-starter-logging 스타터를 통해 자동으로 추가된 로그 관련 라이브러리



스프링 부트 로깅

스프링 부트 로깅

1 스프링 부트의 로깅

〈3〉 로깅 프레임워크 변경

- 기본 로깅 프레임워크인 LogBack을 다른 프레임워크로 교체할 수 있음
- spring-boot-starter 스타터에서 spring-boot-starter-logging 스타터를 제거하고 원하는 로깅 프레임워크 라이브러리를 추가함

스프링 부트 로깅

1 스프링 부트의 로깅

〈3〉 로깅 프레임워크 변경

- Log4j2를 사용하는 경우, log4j2를 추가하면 됨 (실제 설정은 하지 않음)

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

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-log4j2</artifactId>
    </dependency>
</dependencies>
```

스프링 부트 로깅

스프링 부트 로깅

1 스프링 부트의 로깅

④ 로그 레벨

- 스프링 부트로 만든 애플리케이션은 다섯 가지의 레벨의 로그를 저장할 수 있음

레 벨	색 깔	의 미
ERROR	Red	요청을 처리하는 과정에 발생한 치명적인 에러 메시지 출력
WARN	Yellow	향후에 에러의 원인이 될 수 있는 경고성 메시지 출력
INFO	Green	상태 변경과 같은 정보 전달 목적의 메시지 출력
DEBUG	Green	디버깅을 위해 출력하는 메시지 출력
TRACE	Green	DEBUG 레벨 보다 좀 더 상세한 메시지 출력

스프링 부트 로깅

1 스프링 부트의 로깅

④ 로그 레벨

- 특정 로그 레벨을 지정하면 상위 로그가 모두 출력됨

개발 단계	• DEBUG나 TRACE로 설정
운영 단계	• WARN 이상으로 변경

스프링 부트 로깅

스프링 부트 로깅

1 스프링 부트의 로깅

④ 로그 레벨

- 로그 레벨을 변경하기 위해서는 메인 설정 파일(application.yml)에 로그 설정을 추가하면 됨

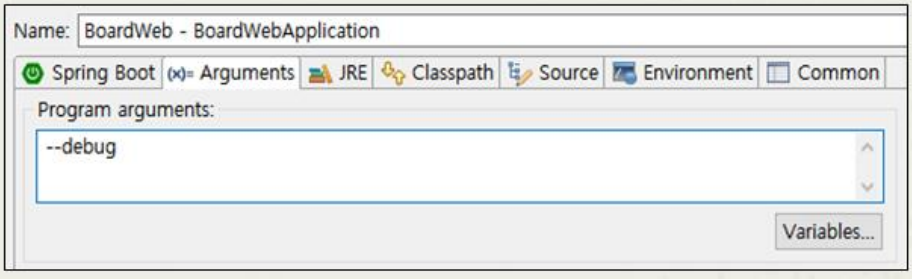
```
logging:
  level:
    '[org.springframework]': debug
```

스프링 부트 로깅

1 스프링 부트의 로깅

④ 로그 레벨

- 일시적으로 로그 레벨을 변경하려면 애플리케이션을 실행할 때, 로그 레벨을 인자로 전달하면 됨



스프링 부트 로깅

스프링 부트 로깅

2 애플리케이션 로그 출력

〈1〉 로그 출력하기

- SLF4J가 제공하는 LoggerFactory를 통해 Logger 객체를 획득
- 획득한 Logger를 이용하여 레벨별로 로그를 출력

스프링 부트 로깅

2 애플리케이션 로그 출력

〈1〉 로그 출력하기

```
package com.mycompany;

import org.slf4j.Logger;

@Service
public class LoggingRunner implements ApplicationRunner {
    private Logger logger = LoggerFactory.getLogger(LoggingRunner.class);

    @Override
    public void run(ApplicationArguments args) throws Exception {
        logger.trace("TRACE 레벨 로그");
        logger.debug("DEBUG 레벨 로그");
        logger.info("INFO 레벨 로그");
        logger.warn("WARN 레벨 로그");
        logger.error("ERROR 레벨 로그");
    }
}
```

스프링 부트 로깅

스프링 부트 로깅

2 애플리케이션 로그 출력

〈2〉 로그 레벨 변경

- 스프링 부트는 별도의 설정이 없으면 기본적으로 INFO 레벨의 로그를 출력

```
2021-07-12 19:34:30.671  INFO 7916 --- [  restartedMain] o.s.b.d.a.OptionalLiveReloadServer       : LiveReload server is running on port 35729
2021-07-12 19:34:30.712  INFO 7916 --- [  restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http) with
2021-07-12 19:34:30.722  INFO 7916 --- [  restartedMain] com.mycompany.BoardWebApplication        : Started BoardWebApplication in 2.406 seconds
2021-07-12 19:34:30.724  INFO 7916 --- [  restartedMain] com.mycompany.LoggingRunner              : INFO 레벨 로그
2021-07-12 19:34:30.725  WARN 7916 --- [  restartedMain] com.mycompany.LoggingRunner              : WARN 레벨 로그
2021-07-12 19:34:30.725 ERROR 7916 --- [  restartedMain] com.mycompany.LoggingRunner              : ERROR 레벨 로그
```

스프링 부트 로깅

2 애플리케이션 로그 출력

〈2〉 로그 레벨 변경

- 로그 레벨을 변경하려면 외부 프로퍼티 파일(application.yam)을 수정

```yaml
logging:
  level:
    '[org.springframework]': error
    '[com.mycompany]': error
```

스프링 부트 로깅

스프링 부트 로깅

2 애플리케이션 로그 출력

〈3〉 파일에 로그 출력하기

- 로그 메시지를 콘솔 뿐만 아니라 파일에도 출력하고 싶으면 메인 설정 파일(application.yml)에 로그 파일의 이름과 위치를 지정함

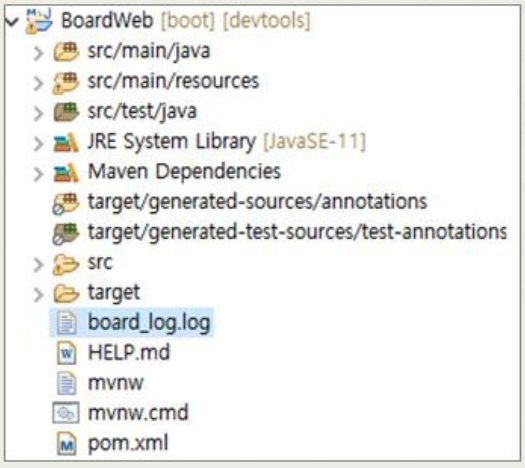
```
logging:
  level:
    '[org.springframework]': error
    '[com.mycompany]': error

  file:
    name: ./board_log.log
```

스프링 부트 로깅

2 애플리케이션 로그 출력

〈3〉 파일에 로그 출력하기

- 프로젝트를 새로고침하면 생성된 로그파일 확인 가능



스프링 부트 로깅

스프링 부트 로깅

3 로그 커스터마이징

〈1〉 로그백(logback.xml) 설정

- 클래스 패스에 logback.xml 파일을 작성하면 직접 로그 관리 가능
- logback.xml 설정보다 application.yml의 우선순위가 높음

1

```yaml
logging:
  level:
    '[org.springframework]': info
#    '[com.mycompany]': info
  file:
    name: ./board_log.log
  config: classpath:log/logback.xml
```

스프링 부트 로깅

3 로그 커스터마이징

〈1〉 로그백(logback.xml) 설정

- 클래스 패스에 logback.xml 파일을 작성하면 직접 로그 관리 가능
- logback.xml 설정보다 application.yml의 우선순위가 높음

2

```xml
<?xml version="1.0" encoding="UTF-8"?>

<configuration>

    <include resource="org/springframework/boot/logging/logback/base.xml" />
    <logger name="com.mycompany" level="TRACE" />

</configuration>
```

스프링 부트 로깅

스프링 부트 로깅

3 로그 커스터마이징

〈1〉 로그백(logback.xml) 설정

- 스프링 부트가 제공하는 base.xml 파일에는 어펜더(Appender)를 비롯한 로깅 관련한 다양한 설정들이 기본적으로 제공됨
- Logger를 추가하여 원하는 로그를 레벨별로 출력함

스프링 부트 로깅

3 로그 커스터마이징

〈2〉 Appender 설정

- 로그가 출력되는 위치와 패턴을 지정할 때 사용함
- ConsoleAppender는 콘솔에, FileAppender는 파일에 로그를 출력함

스프링 부트 로깅

스프링 부트 로깅

3 로그 커스터마이징

〈2〉 Appender 설정

```
<?xml version="1.0" encoding="UTF-8"?>

<configuration>

    <appender name="consoleAppender" class="ch.qos.logback.core.ConsoleAppender">
        <encoder>
            <pattern>
                %d{yyyy:MM:dd HH:mm:ss.SSS} %-5level --- [%thread] %logger{35} : %msg %n
            </pattern>
        </encoder>
    </appender>

    <logger name="com.mycompany" level="warn" additivity="false">
        <appender-ref ref="consoleAppender" />
    </logger>

    <root level="error">
        <appender-ref ref="consoleAppender" />
    </root>
</configuration>
```

스프링 부트 로깅

3 로그 커스터마이징

〈3〉 로그 패턴 설정

- 〈encoder〉를 사용하여 출력할 로그의 패턴을 지정할 수 있음
- 로그 패턴은 〈pattern〉 엘리먼트를 이용하여 설정 추가 가능

스프링 부트 로깅

스프링 부트 로깅

3 로그 커스터마이징

③ 로그 패턴 설정

패 턴	의 미
%d	날짜 및 시간 (yyyy-MM-dd HH:mm:ss,SSS 형태) 출력
%date{format}	원하는 형태로 시간 정보 출력 예) %date{yyyy-MM-dd}
%logger{length}	Logger 이름을 출력하며, length에 따라 로거 이름이 축약됨
%thread	스레드 이름
%-5level	로그 레벨, 5는 출력 고정폭 값
%msg	로그 메시지

스프링 부트 로깅

3 로그 커스터마이징

④ Appender 변경

- 파일에 로그를 출력하기 위해 FileAppender를 사용해야 함
- 로그 파일의 이름과 위치, 새로운 로그파일에 대한 정책, 패턴 등을 지정 가능

스프링 부트 로깅

스프링 부트 로깅

3 로그 커스터마이징

④ Appender 변경

```
<appender name="fileAppender" class="ch.qos.logback.core.rolling.RollingFileAppender">
    <file>src/main/resources/logs/board_log.log</file>
    <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
        <fileNamePattern>
            src/main/resources/log/myboard.%d{yyyy-MM-dd}.log.gz
        </fileNamePattern>
        <maxHistory>30</maxHistory>
    </rollingPolicy>
    <encoder>
        <pattern>
            %d{yyyy:MM:dd HH:mm:ss.SSS} %-5level --- [%thread] %logger{35} : %msg %n
        </pattern>
    </encoder>
</appender>
```

스프링 부트 빌드

스프링 부트 빌드

1 스프링 부트 빌드

〈1〉 빌드

- 빌드는 개발된 소스를 실제 운영 서버에 배포하는 일련의 과정을 의미함

스프링 부트 빌드

1 스프링 부트 빌드

〈1〉 빌드

1. 소스를 컴파일하고 컴파일 된 소스를 적절한 폴더에 취합해야 함
2. 취합된 파일들을 JAR나 WAR 파일로 패키징하고 실제 운영 서버에 배포해야 함

스프링 부트 빌드

스프링 부트 빌드

1 스프링 부트 빌드

① 빌드

- 빌드는 개발된 소스를 실제 운영 서버에 배포하는 일련의 과정을 의미함

JAVA 애플리케이션	• JAR(Java Archive) 파일로 패키징
웹 애플리케이션	• WAR(Web Archive) 파일로 패키징

스프링 부트 빌드

1 스프링 부트 빌드

① 빌드

스프링 부트의 경우

- 독립적으로 실행 가능한 애플리케이션을 빠르게 개발하는 것을 목적으로 하기 때문에 웹 애플리케이션도 JAR 파일로 패키징하여 처리함

스프링 부트 빌드

스프링 부트 빌드

1 스프링 부트 빌드

〈2〉 패키징

패키징을 처리하는 표준 빌드 도구

JAVA	Ant
스프링 부트	메이븐(Maven)이나 Gradle

스프링 부트 빌드

1 스프링 부트 빌드

〈2〉 패키징

- 메이븐이 프로젝트를 빌드할 때 사용하는 패키징 폴더는 target

스프링 부트 빌드

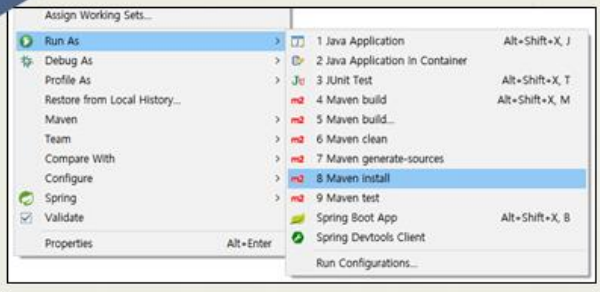
스프링 부트 빌드

1 스프링 부트 빌드

〈2〉 패키징

- 메이븐의 install 단계를 실행 시 target 폴더에 패키징 결과 파일 생성됨
- 메이븐의 로컬 리파지토리에도 jar 파일이 업로드 됨

1



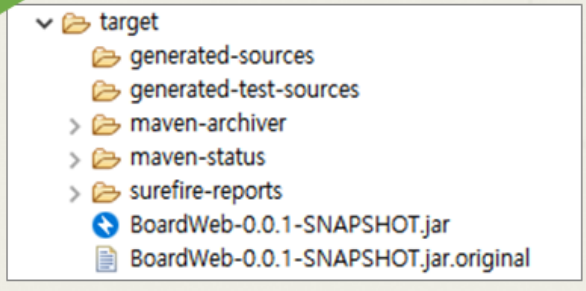
스프링 부트 빌드

1 스프링 부트 빌드

〈2〉 패키징

- 메이븐의 install 단계를 실행 시 target 폴더에 패키징 결과 파일 생성됨
- 메이븐의 로컬 리파지토리에도 jar 파일이 업로드 됨

2



스프링 부트 빌드

스프링 부트 빌드

2 패키징 파일 구조

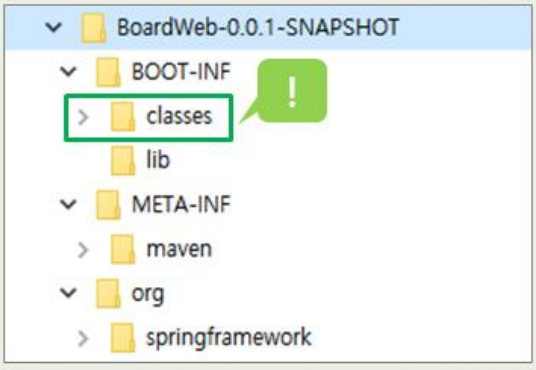
패키징 된 jar 파일은
BOOT-INF, META-INF, org 폴더로 구성됨

스프링 부트 빌드

2 패키징 파일 구조

〈1〉 BOOT-INF 폴더

- BOOT-INF/classes는 src/main/java에서 컴파일한 .class 파일들과 src/main/resources에 작성한 여러 설정 파일들이 포함됨



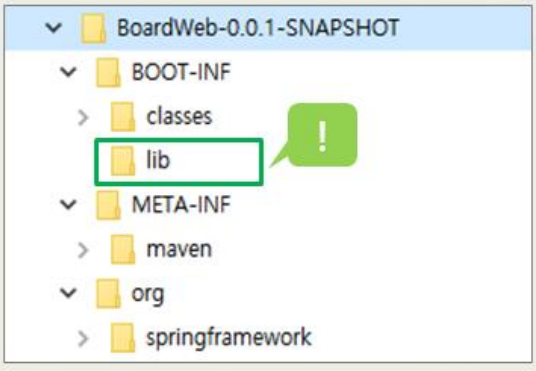
스프링 부트 빌드

스프링 부트 빌드

2 패키징 파일 구조

① BOOT-INF 폴더

- BOOPT-INF/lib 폴더에는 프로젝트의 [Maven Dependencies…]에 등록된 모든 라이브러리들이 포함됨



스프링 부트 빌드

2 패키징 파일 구조

② META-INF 폴더

- META-INF/maven 폴더에는 메이븐이 사용하는 POM(pom.xml) 파일이 있음

스프링 부트 빌드

스프링 부트 빌드

2 패키징 파일 구조

⟨2⟩ META-INF 폴더

- JAR 파일에는 반드시 MENIFEST라는 메타데이터 파일이 포함됨
- 메타데이터 파일을 기반으로 애플리케이션 운용에 필요한 다양한 정보들을 해석하고 처리함

스프링 부트 빌드

2 패키징 파일 구조

⟨2⟩ META-INF 폴더

MANIFEST.MF - Windows 메모장

파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

```
Manifest-Version: 1.0
Created-By: Maven Jar Plugin 3.2.0
Build-Jdk-Spec: 15
Implementation-Title: BoardWeb
Implementation-Version: 0.0.1-SNAPSHOT
Main-Class: org.springframework.boot.loader.JarLauncher
Start-Class: com.mycompany.BoardWebApplication
Spring-Boot-Version: 2.5.2
Spring-Boot-Classes: BOOT-INF/classes/
Spring-Boot-Lib: BOOT-INF/lib/
Spring-Boot-Classpath-Index: BOOT-INF/classpath.idx
Spring-Boot-Layers-Index: BOOT-INF/layers.idx
```

Ln 11, Col 52 | 100% | Windows (CRLF) | UTF-8

스프링 부트 빌드

스프링 부트 빌드

2 패키징 파일 구조

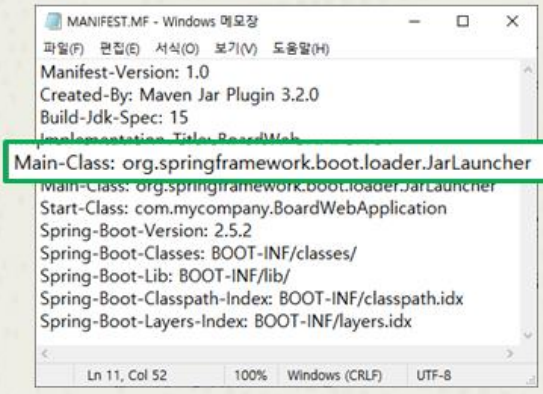
⟨2⟩ META-INF 폴더

Main-Class

- 애플리케이션 실행을 위한 메인 클래스가 JarLauncher 클래스라는 것

Start-Class

- 실제로 BoardWebApplication 클래스를 시작으로 애플리케이션을 실행하라는 의미



스프링 부트 빌드

2 패키징 파일 구조

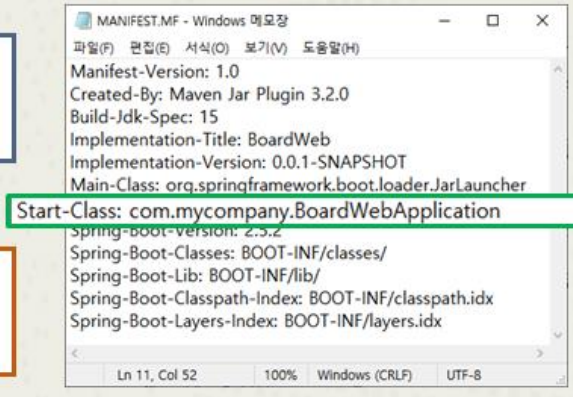
⟨2⟩ META-INF 폴더

Main-Class

- 애플리케이션 실행을 위한 메인 클래스가 JarLauncher 클래스라는 것

Start-Class

- 실제로 BoardWebApplication 클래스를 시작으로 애플리케이션을 실행하라는 의미



스프링 부트 빌드

스프링 부트 빌드

3 JAR 실행

① 스프링 부트 로더

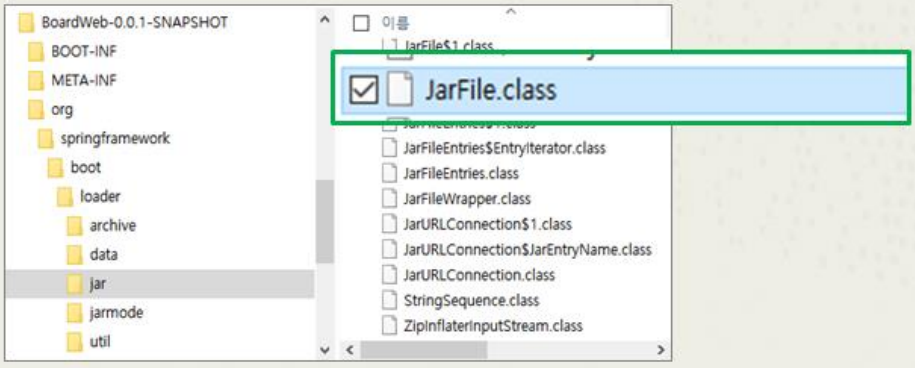
- 애플리케이션이 정상적으로 동작하기 위해서는 외부에서 제공하는 라이브러리가 필요함
- 스프링 부트는 패키징에 포함된 또 다른 JAR 파일을 로딩하는 유틸리티를 제공함

스프링 부트 빌드

3 JAR 실행

① 스프링 부트 로더

- org/springframework.boot/loader/jar 폴더에 있는 JarFile이라는 클래스가 BOOT-INF/lib 폴더에 있는 JAR 파일들을 로딩함



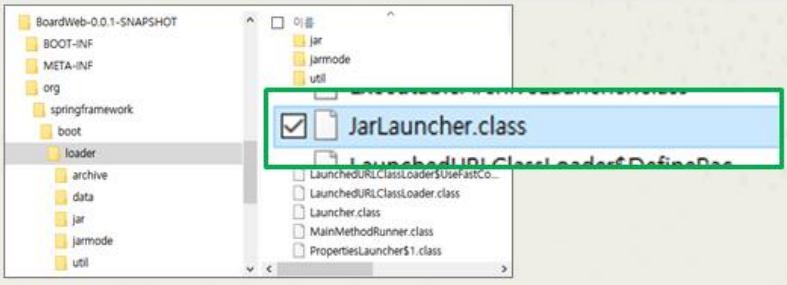
스프링 부트 빌드

스프링 부트 빌드

3 JAR 실행

〈2〉 JarLauncher

- org/springframework.boot/loader 폴더에 있는 JarLauncher 클래스
- 패키징된 Jar 파일의 메타데이터 (META-INF/MANIFEST.MF)를 로딩하여 메인 클래스를 찾아 실행함



스프링 부트 빌드

3 JAR 실행

〈3〉 애플리케이션 실행

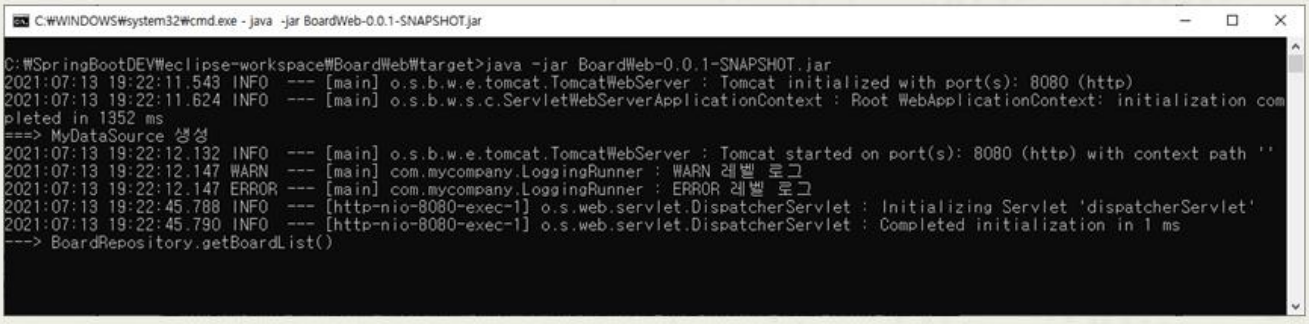
- Jar 파일로 패키징된 애플리케이션은 콘솔에서 바로 실행 가능
- 패키징된 Jar 파일이 위치한 디렉토리로 작업 디렉토리를 이동한 후에 'java -jar 압축파일이름'을 입력하면 애플리케이션이 실행됨

스프링 부트 빌드

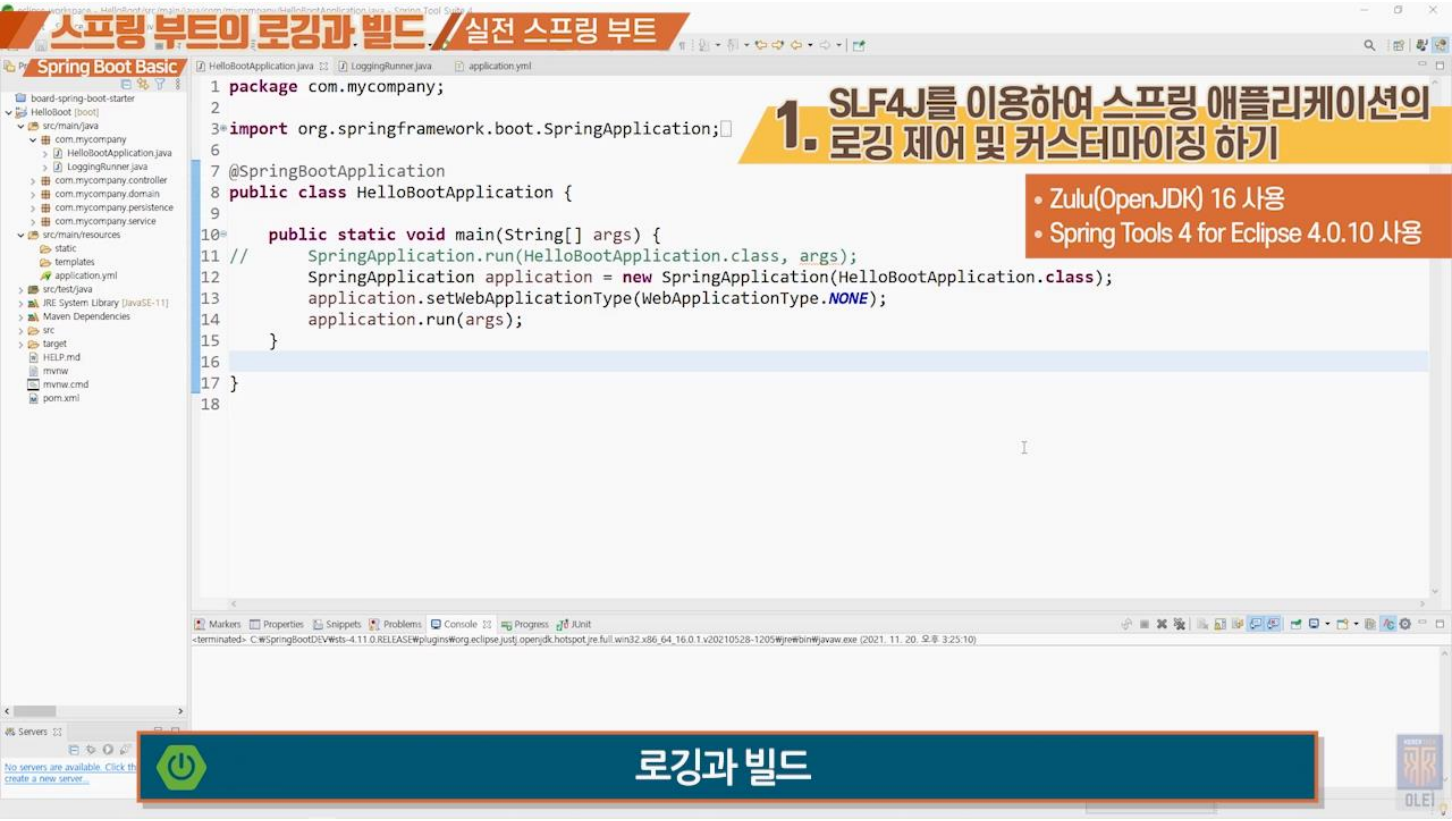
스프링 부트 빌드

3 JAR 실행

〈3〉 애플리케이션 실행



실전 스프링 부트

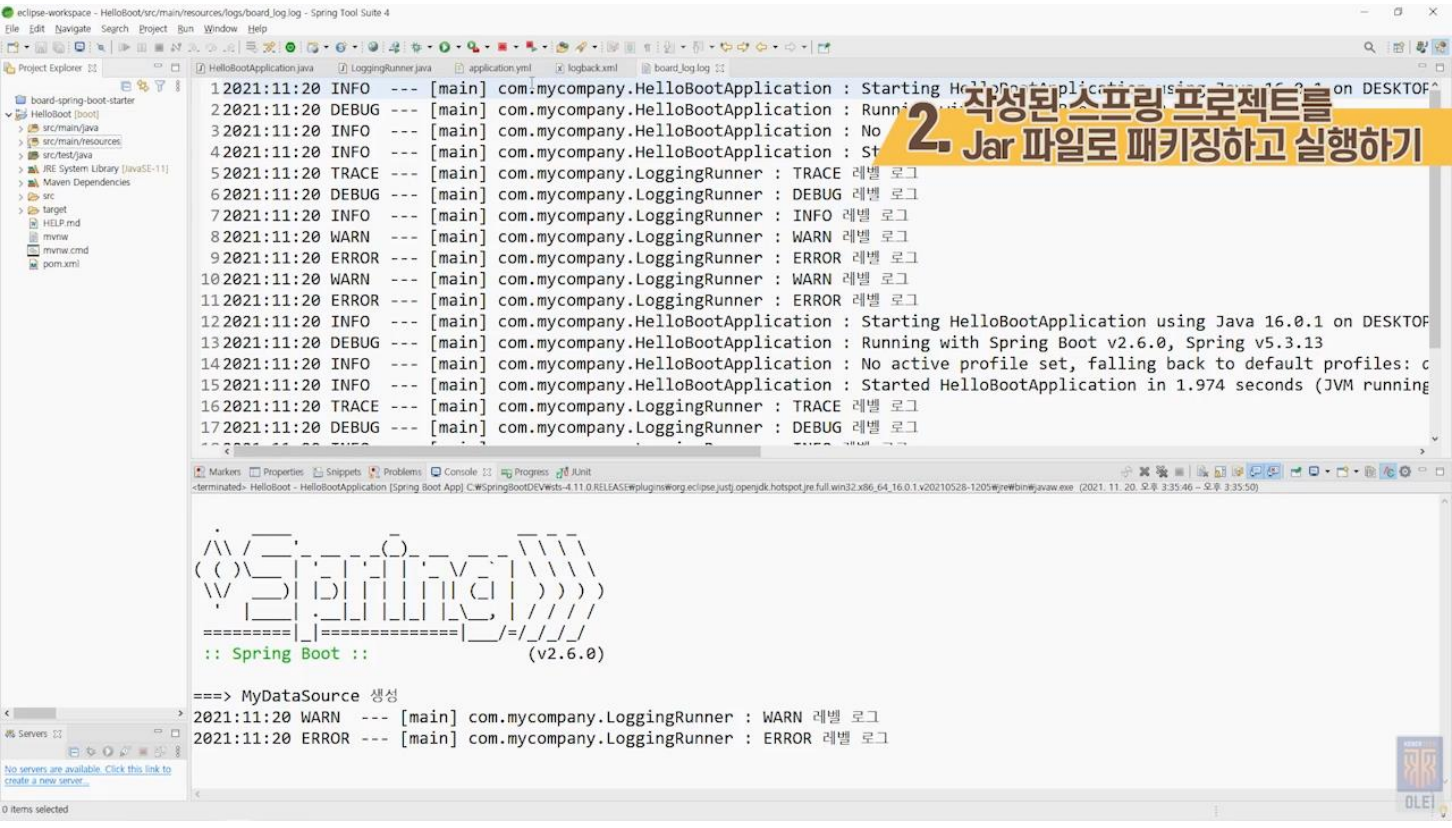


실습단계
로깅과 빌드
기본 로그 설정 확인, src/test/java → com.mycompany에 LoggingRunner.java 테스트 클래스 작성
LoggerFactory 클래스에 getLogger 메소드를 통해서 Logger 객체를 획득
5개의 로그 레벨이 출력되도록 타이핑할 것
로그 레벨 별 출력
주의! 로그 레벨이 디폴트로 INFO가 세팅되기 때문에 info 레벨 이상 로그만 출력됨
로그 레벨 변경 : yml 파일에 로그 레벨 변경 설정 추가
WARN 이상 출력
로그 레벨을 trace로 낮추고 실행하면 trace부터 전부 출력됨
SLF4J 구현체에 해당하는 로그백을 이용하여 디테일하게 제어할 경우
src/main/resources → New → Other... → XML File 형태로 작성
File name : logback.xml 작성

실전 스프링 부트

실습단계
Tip! 화면을 잠깐 멈추고 소스를 타이핑하며 진행할 것
2개의 Appender 1. FileAppender 2. ConsoleAppender
1. FileAppender : src/main/resources의 특정 파일을 만들어서 출력
2. ConsoleAppender : console 창에 로그 출력
<encoder> 태그를 이용하여 로그 패턴 지정
%d : 날짜 데이터 포맷 형식 지정
level : 로그 레벨 출력
-5 : 출력하는 고정 폭을 설정함
%thread : thread 이름을 출력
%logger : logger 이름을 출력, 이름이 길 경우 글자수 제한 값 입력
%msg : 메시지 출력, %n : 개행 처리
레벨 변경 가능
레벨을 'warn'으로 설정했으나 모두 출력되는 이유는 무엇인가?
yml 파일의 우선순위가 더 높기 때문!
<Ctrl> + /를 이용하여 주석 처리
src/main/resources → logs → board_log.log

실전 스프링 부트



실습단계
target 폴더가 비어있음
HelloBoot → Run As → Maven install, 빌드 진행
테스트 케이스 등이 실행됨
중요! BUILD SUCCESS 출력을 위해 테스트 케이스가 오류이면 안 됨
웹 애플리케이션이 jar 형태로 패키징 된 것을 확인할 수 있음