Spring Boot Basic

스프링 부트 기초

스프링 부트 자동 설정

한국기술교육대학교
온라인평생교육원

학습내용

- 자동 설정 개념과 동작 원리 이해
- 사용자 정의 자동 설정

학습목표

- 스프링 부트가 제공하는 **자동 설정의 개념과 동작 원리**를 설명할 수 있다.
- **사용자 정의 자동 설정** 클래스를 작성하고 적용할 수 있다.

자동 설정 개념과 동작 원리 이해

자동 설정 개념과 동작 원리 이해

1 자동 설정 개요

〈1〉 자동 설정이란?

- 스프링 부트에서 복잡한 XML 설정 없이 웹 애플리케이션을 작성할 수 있는 이유는 내부적으로 자동 설정이 동작하기 때문

자동 설정 개념과 동작 원리 이해

1 자동 설정 개요

〈1〉 자동 설정이란?

- 스프링 기반의 애플리케이션은 두 종류의 객체를 사용

1. 스프링이 제공하는 객체
2. 사용자가 작성한 객체

자동 설정 개념과 동작 원리 이해

자동 설정 개념과 동작 원리 이해

1 자동 설정 개요

〈2〉 @SpringBootApplication

다음 어노테이션들을 포함

- @SpringBootConfiguration
- @EnableAutoConfiguration
- @ComponentScan

```
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@SpringBootConfiguration
@EnableAutoConfiguration
@ComponentScan(excludeFilters
ype.CUSTOM, classes = TypeExcludeFilter.class),
ConfigurationExcludeFilter.class) })
```

자동 설정 개념과 동작 원리 이해

1 자동 설정 개요

〈2〉 @SpringBootApplication

아래로 대체 가능

- @SpringBootConfiguration
- @EnableAutoConfiguration
- @ComponentScan

```
@SpringBootConfiguration
@EnableAutoConfiguration
@ComponentScan
public class BoardWebApplication {

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

자동 설정 개념과 동작 원리 이해

자동 설정 개념과 동작 원리 이해

1 자동 설정 개요

〈3〉 @SpringBootConfiguration

- 설정 클래스를 지정할 때 사용하는 어노테이션
- @Configuration과 동일한 기능을 제공하며 이름만 변경한 것

자동 설정 개념과 동작 원리 이해

1 자동 설정 개요

〈3〉 @SpringBootConfiguration

- 스프링 컨테이너는 설정 클래스를 로딩하고 설정 클래스에 @Bean으로 등록된 객체를 생성하고 관리함

```
@SpringBootConfiguration
@EnableAutoConfiguration
@ComponentScan
public class BoardWebApplication {

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

    @Bean
    public BoardService boardService() {
        return new BoardService();
    }
}
```

자동 설정 개념과 동작 원리 이해

자동 설정 개념과 동작 원리 이해

1 자동 설정 개요

④ @ComponentScan

- @Configuration, @Repository, @Service, @Controller, @RestController가 붙은 클래스의 객체를 메모리에 생성

```
@Service("boardService")
public class BoardService {
    public BoardService() {
        System.out.println("===> BoardService 생성");
    }
}
```

자동 설정 개념과 동작 원리 이해

1 자동 설정 개요

④ @ComponentScan

- @Configuration, @Repository, @Service, @Controller, @RestController가 붙은 클래스는 @ComponentScan이 설정된 클래스의 하위 패키지에 위치해야 함

자동 설정 개념과 동작 원리 이해

자동 설정 개념과 동작 원리 이해

1 자동 설정 개요

④ @ComponentScan

```
package com.mycompany;
import org.springframework.boot.SpringApplication;

@SpringBootConfiguration
@EnableAutoConfiguration
@ComponentScan
public class BoardWebApplication {

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

자동 설정 개념과 동작 원리 이해

1 자동 설정 개요

⑤ @EnableAutoConfiguration

- 스프링 부트는 스프링 컨테이너를 구동할 때, 두 단계로 나누어 객체를 생성함

1. @EnableAutoConfiguration에 의해 자동 설정 클래스에 의해 객체들이 생성됨
2. @ComponentScan이 동작하여 사용자가 작성한 객체들이 생성됨

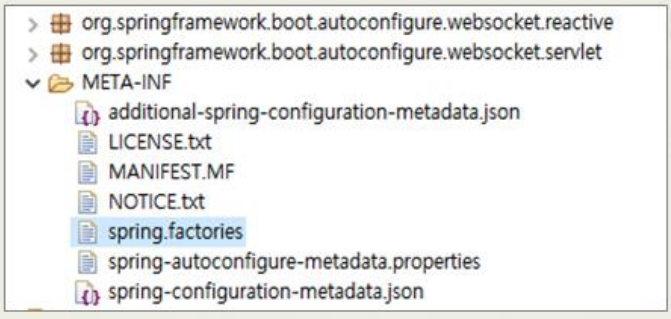
자동 설정 개념과 동작 원리 이해

자동 설정 개념과 동작 원리 이해

1 자동 설정 개요

⑤ @EnableAutoConfiguration

- spring-boot-autoconfigure-x.x.x.RELEASE.jar 파일에 포함되어 있음
- META-INF/spring.factories 파일에 등록된 자동 설정 클래스들을 처리함



자동 설정 개념과 동작 원리 이해

2 자동 설정 동작 원리

① spring.factories 파일

- 스프링 부트는 spring.factories 파일을 로딩하여 애플리케이션 운용에 필요한 다양한 정보를 처리함

자동 설정 개념과 동작 원리 이해

자동 설정 개념과 동작 원리 이해

2 자동 설정 동작 원리

〈1〉 spring.factories 파일

spring.factories 파일에 설정된 정보들

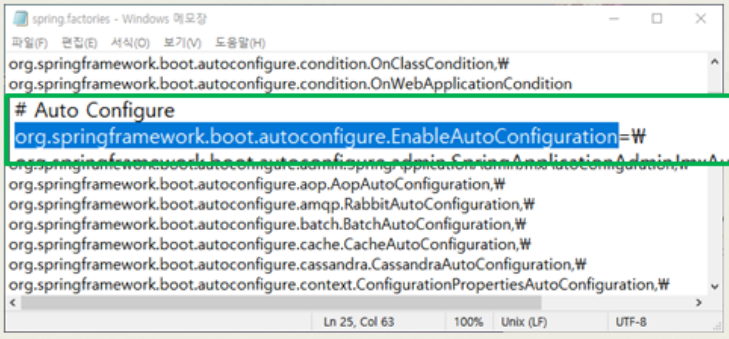
- # Initializers
- # Application Listeners
- # Environment Post Processors
- # Auto Configuration Import Listeners
- # Auto Configuration Import Filters
- # Auto Configure
- # Failure analyzers
- # Template availability providers
- # DataSource initializer detectors

자동 설정 개념과 동작 원리 이해

2 자동 설정 동작 원리

〈2〉 웹 애플리케이션 자동 설정 클래스

- '# Auto Configure' 밑에 있는 org.springframework.boot.autoconfigure.EnableAutoConfiguration에 수많은 자동 설정 클래스들이 등록되어 있음



자동 설정 개념과 동작 원리 이해

자동 설정 개념과 동작 원리 이해

2 자동 설정 동작 원리

〈2〉 웹 애플리케이션 자동 설정 클래스

org.springframework.boot.autoconfigure.web.servlet.WebMvcAutoConfiguration 클래스

- 웹 애플리케이션 개발과 관련된 자동 설정 클래스

자동 설정 개념과 동작 원리 이해

2 자동 설정 동작 원리

〈2〉 웹 애플리케이션 자동 설정 클래스

```
@Configuration(proxyBeanMethods = false)
@ConditionalOnWebApplication(type = Type.SERVLET)
@ConditionalOnClass({ Servlet.class, DispatcherServlet.class, WebMvcConfigurer.class })
@ConditionalOnMissingBean(WebMvcConfigurationSupport.class)
@AutoConfigureOrder(Ordered.HIGHEST_PRECEDENCE + 10)
@AutoConfigureAfter({ DispatcherServletAutoConfiguration.class, TaskExecutionAutoConfiguration.class,
        ValidationAutoConfiguration.class })
public class WebMvcAutoConfiguration {
```

자동 설정 개념과 동작 원리 이해

자동 설정 개념과 동작 원리 이해

2 자동 설정 동작 원리

〈3〉 자동 설정 클래스 어노테이션

1. @Configuration은 현재 클래스가 스프링 설정 클래스임을 의미함
2. @ConditionalOnWebApplication은 웹 애플리케이션 타입이 어떻게 설정되어 있는지 확인함
3. @ConditionalOnClass은 특정 클래스가 클래스 패스에 존재하는지를 확인함
   반대 설정은 @ConditionalOnMissingClass

자동 설정 개념과 동작 원리 이해

2 자동 설정 동작 원리

〈3〉 자동 설정 클래스 어노테이션

4. @ConditionalOnMissingBean은 특정 클래스의 객체가 메모리에 없는지를 확인함
   반대 설정은 @ConditionalOnBean
5. @AutoConfigureOrder는 자동 설정 클래스들의 우선 순위를 지정할 때 사용함
6. @AutoConfigureAfter는 현재의 자동 설정 클래스가 다른 자동 설정 클래스 다음에 적용되도록 지정할 때 사용함

자동 설정 개념과 동작 원리 이해

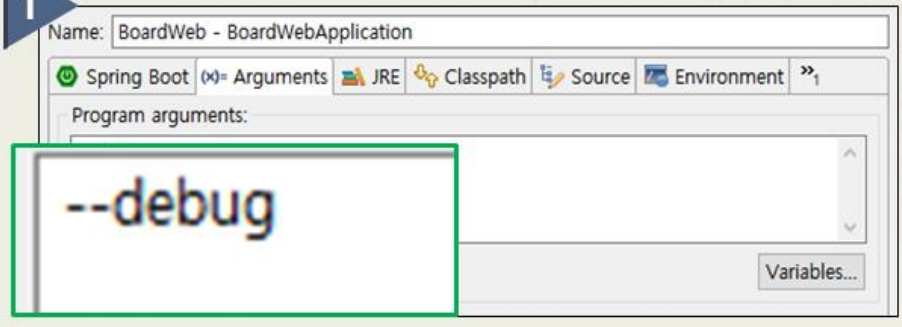
자동 설정 개념과 동작 원리 이해

2 자동 설정 동작 원리

④ 자동 설정 클래스 적용 확인

- 애플리케이션을 실행할 때, 적용되는 자동 설정 클래스를 보고 싶으면 로그 레벨을 DEBUG로 설정하면 됨(스프링 부트의 기본 로깅 레벨은 INFO)

1



자동 설정 개념과 동작 원리 이해

2 자동 설정 동작 원리

④ 자동 설정 클래스 적용 확인

- 애플리케이션을 실행할 때, 적용되는 자동 설정 클래스를 보고 싶으면 로그 레벨을 DEBUG로 설정하면 됨(스프링 부트의 기본 로깅 레벨은 INFO)

2

```
WebMvcAutoConfiguration#hiddenHttpMethodFilter:
   Did not match:
      - @ConditionalOnProperty (spring.mvc.hiddenmethod.filter.enabled) did not find property 'enabled' (OnPropertyCondition)

WebMvcAutoConfiguration.ResourceChainCustomizerConfiguration:
   Did not match:
      - @ConditionalOnEnabledResourceChain did not find class org.webjars.WebJarAssetLocator (OnEnabledResourceChainCondition)

WebMvcAutoConfiguration.WebMvcAutoConfigurationAdapter#beanNameViewResolver:
   Did not match:
      - @ConditionalOnMissingBean (types: org.springframework.web.servlet.view.BeanNameViewResolver; SearchStrategy: all) found
```

사용자 정의 자동 설정

사용자 정의 자동 설정

1 사용자 정의 자동 설정

〈1〉 자동으로 생성할 클래스 작성

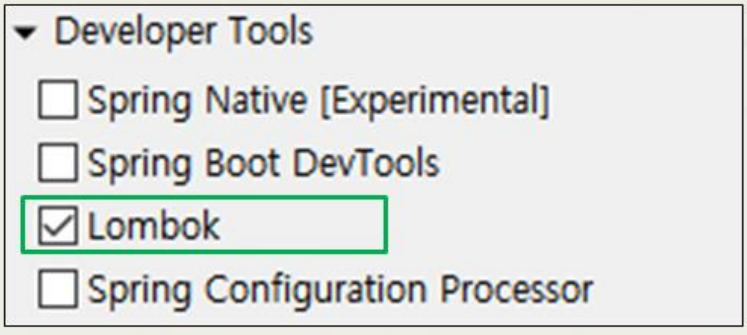
- 특정 도메인이나 비즈니스에 최적화된 자동 완성 클래스를 만들어서 사용할 수 있음

사용자 정의 자동 설정

1 사용자 정의 자동 설정

〈1〉 자동으로 생성할 클래스 작성

- JAVA 클래스를 작성할 때, Lombok을 이용하면 JAVA 클래스에 Getter, Setter, toString 같은 메소드를 쉽게 추가할 수 있음



사용자 정의 자동 설정

사용자 정의 자동 설정

1 사용자 정의 자동 설정

〈1〉 자동으로 생성할 클래스 작성

- Lombok이 제공하는 어노테이션

@Getter	• 멤버 변수에 대한 Getter 메소드를 제너레이션함
@Setter	• 멤버 변수에 대한 Setter 메소드를 제너레이션함
@ToString	• toString 메소드를 재정의함

사용자 정의 자동 설정

1 사용자 정의 자동 설정

〈1〉 자동으로 생성할 클래스 작성

- Lombok이 제공하는 어노테이션

@AllArgs Constructor	• 모든 멤버변수를 초기화 하는 생성자를 제너레이션함
@EqualsAnd HashCode	• equals와 hashCode 메소드를 제너레이션함
@Data	• 위에 열거한 모든 어노테이션을 포함

사용자 정의 자동 설정

사용자 정의 자동 설정

1 사용자 정의 자동 설정

⟨1⟩ 자동으로 생성할 클래스 작성

- MyDataSource는 나중에 자동 설정 클래스에서 객체 생성할 클래스

```
package jdbc.common;

import lombok.Getter;
import lombok.Setter;
import lombok.ToString;

@Getter
@Setter
@ToString
public class MyDataSource {
    private String driverClass;
    private String url;
    private St{
    private St("===> MyDataSource 생성");

    public MyD
        System
    }
}
```

사용자 정의 자동 설정

1 사용자 정의 자동 설정

⟨2⟩ 자동 설정 클래스 작성

- 자동 설정 클래스는 @Configuration을 가짐
- 컨테이너가 생성할 객체를 @Bean으로 등록함

```
package jdbc.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class BoardAu on {

    @Bean
    public MyDataSourc rce() {
                        w MyDataSource();
                        racle.jdbc.driver.OracleDriver");
                        e:thin:@localhost:1521:xe");
        dataSource.setUsername("hr");
        dataSource.setPassword("hr");
        return dataSource;
    }
}
```

사용자 정의 자동 설정

사용자 정의 자동 설정

1 사용자 정의 자동 설정

〈3〉 자동 설정 클래스 등록

- 애플리케이션이 실행될 때, 자동 설정 클래스가 동작하기 위해서는 spring.factories 파일에 해당 클래스가 자동 설정 클래스로 등록되어야 함

사용자 정의 자동 설정

1 사용자 정의 자동 설정

〈3〉 자동 설정 클래스 등록

1. src/main/resources 소스 폴더에 META-INF 폴더를 생성함
2. spring.factories 파일을 다음과 같이 작성함

```
org.springframework.boot.autoconfigure.EnableAutoConfiguration=\
jdbc.config.BoardAutoConfiguration
```

사용자 정의 자동 설정

사용자 정의 자동 설정

1 사용자 정의 자동 설정

⟨4⟩ 자동 설정 클래스 적용 확인

- 스프링 부트는 기본적으로 애플리케이션 로그 레벨을 INFO로 설정하여 실행함
- 추가된 자동 설정 클래스가 동작하는지 확인하기 위해서는 애플리케이션 로그 레벨을 DEBUG로 변경함

사용자 정의 자동 설정

1 사용자 정의 자동 설정

⟨5⟩ 자동 생성 객체 사용

```
package com.mycompany.biz.board;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.ApplicationArguments;
import org.springframework.boot.ApplicationRunner;
import org.springframework.stereotype.Service;

import jdbc.common.MyDataSource;

@Service("boardService")
public class BoardService implements ApplicationRunner {

    @Autowired
    private MyDataSource dataSource;

    public BoardService() {
        System.out.println("===> BoardService 생성");
    }

    @Override
    public void run(ApplicationArguments args) throws Exception {
        System.out.println(dataSource.getUrl() + "DB 연결 성공");

    }
}
```

사용자 정의 자동 설정

사용자 정의 자동 설정

2 객체 재정의(Overriding)

〈1〉 자동 설정 객체 재정의

- 자동 설정 클래스가 제공하는 객체가 아닌 새로운 객체를 사용하기 위해 메인 클래스에 @Bean 설정을 추가함

사용자 정의 자동 설정

2 객체 재정의(Overriding)

〈1〉 자동 설정 객체 재정의

```
@SpringBootConfiguration
@EnableAutoConfiguration
@ComponentScan
public class BoardWebApplication {

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

    @Bean
    public MyDataSource getMyDataSource() {
        MyDataSource dataSource = new MyDataSource();
        dataSource.setDriverClass("org.h2.Driver");
        dataSource.setUrl("jdbc:h2:tcp://localhost/~/test");
        dataSource.setUsername("sa");
        dataSource.setPassword("");
        return dataSource;
    }
}
```

사용자 정의 자동 설정

사용자 정의 자동 설정

2 객체 재정의(Overriding)

〈2〉 객체 재정의 설정 추가

- 자동 설정 클래스도 MyDataSource를 생성하고, 메인 클래스에서도 동일한 객체를 생성하면 다음과 같은 에러가 발생함

```
Description:

The bean 'getMyDataSource', defined in class path resource
[jdbc/config/BoardAutoConfiguration.class],
could not be registered. A bean with that name has already been defined in
com.mycompany.BoardWebApplication  and
overriding is disabled.

Action:

Consider renaming one of the beans or enabling overriding by setting
spring.main.allow-bean-definition-overriding=true
```

사용자 정의 자동 설정

2 객체 재정의(Overriding)

〈2〉 객체 재정의 설정 추가

- 동일한 타입의 객체가 중복되어 생성될 때 객체를 재정의할 수 있도록 application.yml 파일을 수정함

```yaml
# WebApplicationType 설정
spring:
  main:
    web-application-type: none
    allow-bean-definition-overriding: true
```

사용자 정의 자동 설정

사용자 정의 자동 설정

2 객체 재정의(Overriding)

〈3〉 @Conditional 적용

@SpringBootApplication에 포함되는 것

- @EnableAuto Configuration
- @ComponentScan

사용자 정의 자동 설정

2 객체 재정의(Overriding)

〈3〉 @Conditional 적용

1. 스프링 부트는 @ComponentScan이 먼저 동작하여 사용자가 등록한 빈을 먼저 생성함
2. @EnableAutoConfiguration이 동작하여 자동 설정에 의한 빈을 등록함

사용자 정의 자동 설정

사용자 정의 자동 설정

2 객체 재정의(Overriding)

〈3〉 @Conditional 적용

- @Conditional 어노테이션을 사용하면 특정 조건을 만족하는 객체를 선택적으로 사용할 수 있음

사용자 정의 자동 설정

2 객체 재정의(Overriding)

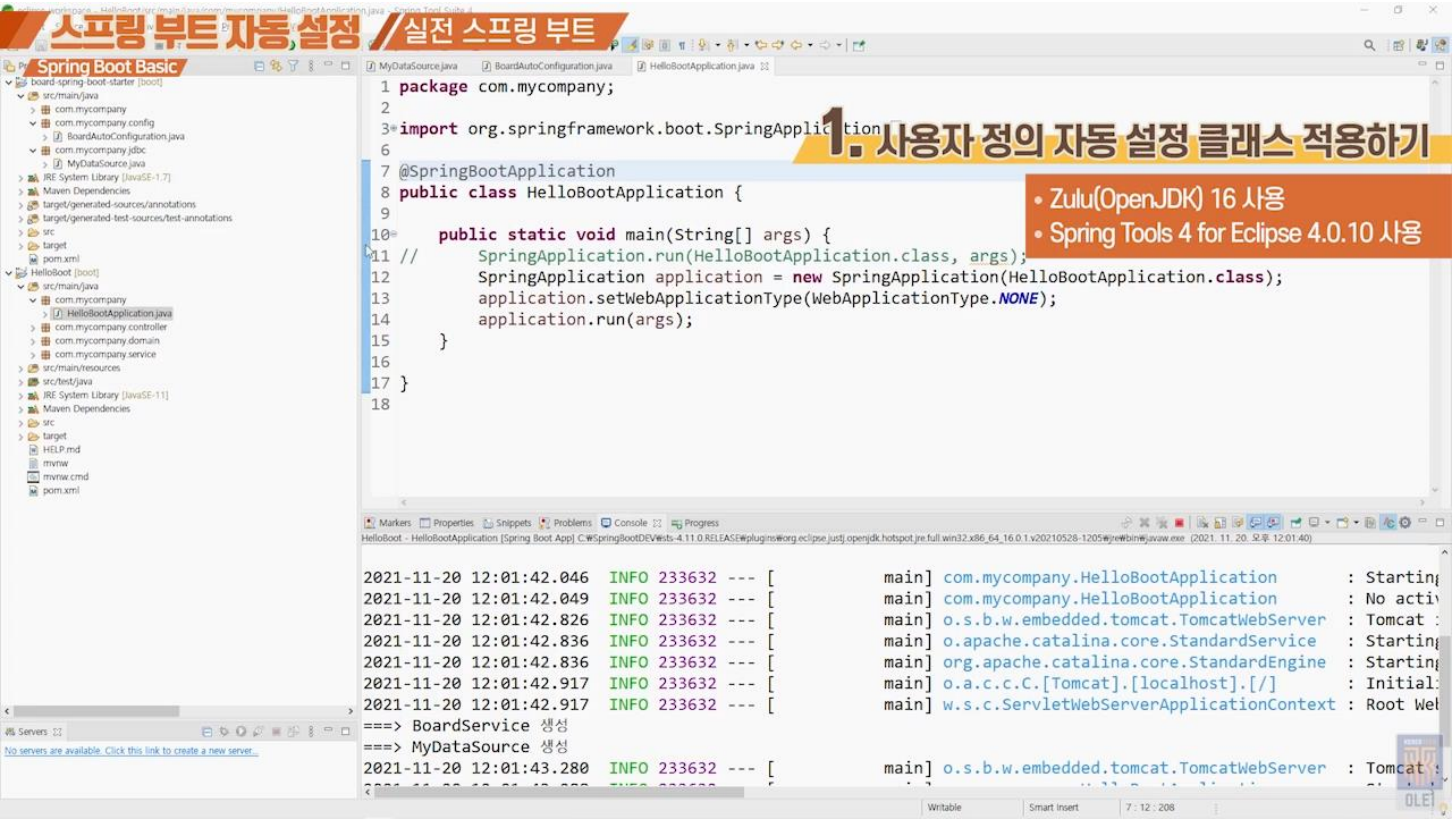
〈3〉 @Conditional 적용

- 이 설정은 MyDataSource 타입의 객체가 메모리에 없을 경우에만 자동 설정에 의한 객체가 생성되기 때문에 H2 기반의 MyDataSource 객체가 먼저 적용됨

```
@Configuration
public class BoardAutoConfiguration {

    @Bean
    @ConditionalOnMissingBean
    public MyDataSource getMyDataSource() {
        MyDataSource dataSource = new MyDataSource();
        dataSource.setDriverClass("oracle.jdbc.driver.OracleDriver");
        dataSource.setUrl("jdbc:oracle:thin:@localhost:1521:xe");
        dataSource.setUsername("hr");
        dataSource.setPassword("hr");
        return dataSource;
    }
}
```

실전 스프링 부트

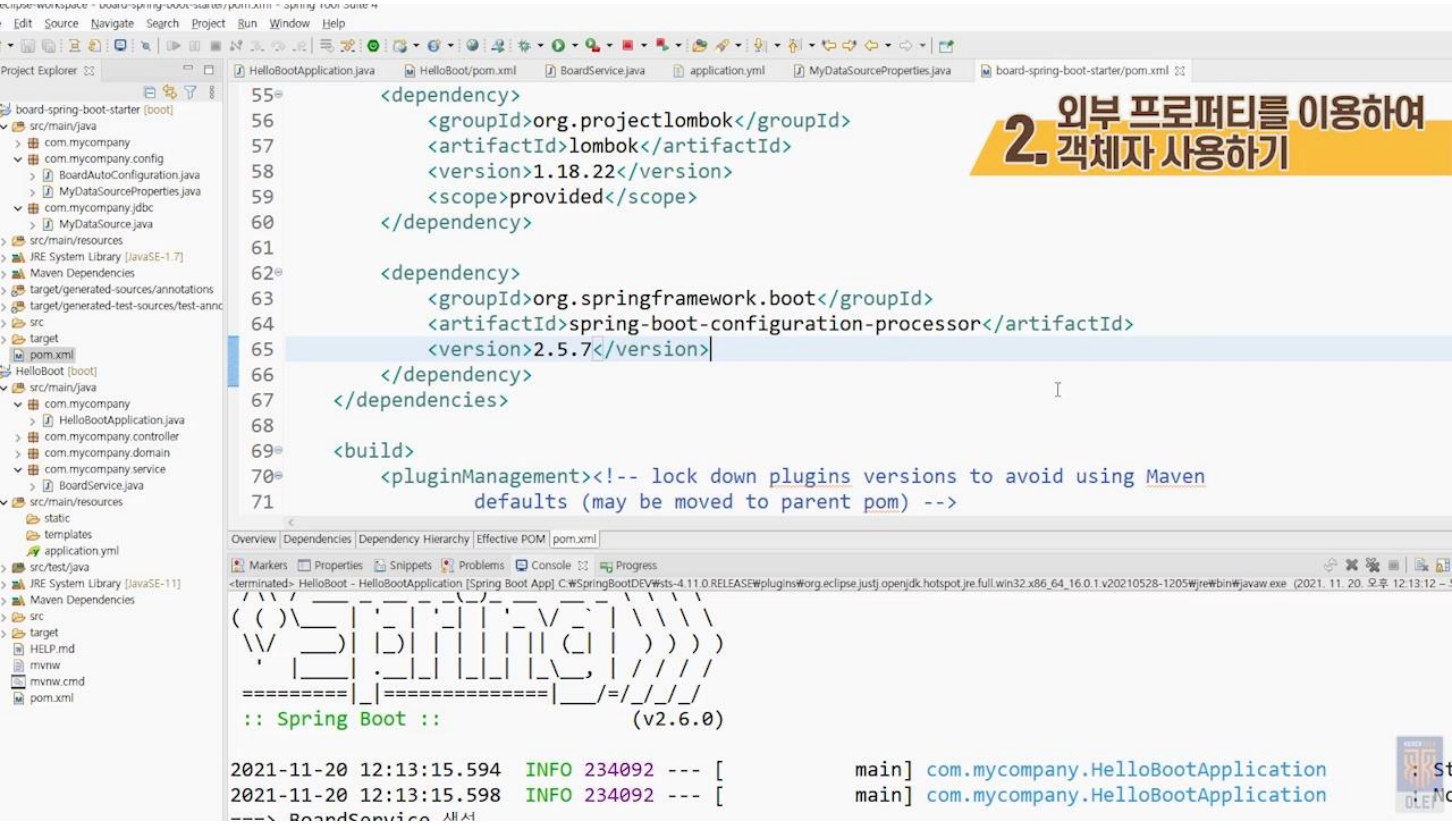


실습단계
스프링 부트의 자동 설정(Auto Configure)
board-spring-boot-starter → MyDataSource.java
커넥션을 위한 네 개의 변수 선언
BoardAutoConfiguration 클래스 생성
Oracle 기반의 DriverClass, JDBC Url, Usename, Password 등
환경설정 클래스가 로딩되는 순간 MyDataSource 객체가 메모리에 생성되도록 설정
중요! 자동설정 클래스가 로딩되도록 하려면 spring.factories 파일이 필요함
현재 프로젝트에 src/main/resources라는 소스 폴더 하나 생성
META-INF 폴더 생성
spring.factories 파일 작성
EnableAutoCounfiguration 프로퍼티로 BoardAutoConfiguration 클래스 등록
로딩해야 하는 자동설정 클래스임을 인지하게 됨
Run As → Maven install
BUILD SUCCESS가 뜨면 프로젝트를 닫아도 됨

실전 스프링 부트

실습단계
board-spring-boot-starter를 pom.xml에 dependency로 등록했음
자동 설정 클래스를 로딩해서 MyDataSource 객체를 이용할 수 있음
com.mycompany.service 패키지 생성 후 BoardService.java 클래스 작성
@Autowired로 의존성 주입
run 메소드가 무조건 실행됨
DataSource 객체에 있는 URL 정보 출력
서블릿으로 되어 있다면 일반 JAVA 애플리케이션으로 테스트하는 것이 빠르기 때문에 NONE으로 변경
oracle 기반의 MyDataSource가 생성되어 이런 결과가 출력됨
프로퍼티를 이용해 자동설정으로 메모리에 올라간 MyDataSource 객체 커스터마이징
com.mycompany.config → New → Class
MyDataSourceProperties 클래스 추가, 프로퍼티에 해당하는 변수 추가
프로퍼티 정보를 이용해 객체 설정된 프로퍼티 값을 변경
노란색 경고가 뜰 경우 링크를 누르면 필요한 파일이 자동으로 추가됨
문제점! 버전 정보가 누락이 되었기 때문에 <optional> 태그 대신에 버전 정보를 추가하여 에러 해결

실전 스프링 부트



실습단계
외부 프로퍼티 정보를 이용하기 위해서 EnableConfigurationProperties 어노테이션 추가
MyDataSourceProperties를 이용하여 MyDataSource 객체의 속성 값을 변경
의존성 주입
Properties를 이용해 MyDataSource 객체에 변수 값을 적절히 세팅
저장 → Run As → Maven install
application.yml 파일에 사용하고 싶은 데이터 소스 설정 추가
Oracle 기반의 정보를 H2 기반의 커넥션 정보로 변경
jdbc url을 Oracle 기반으로 변경할 경우
객체를 직접 수정하지 않고도 외부 프로퍼티 정보를 로딩해서 객체를 자동으로 띄울 수 있음