

# 스프링부트

## <제목 차례>

### <제목 차례>

1. 스프링부트 개요	3
가. 탄생배경	3
나. 스프링부트 기능	3
다. 스프링과 스프링부트 차이점	3
라. 스프링부트 구성요소	3
마. 스프링 부트 모듈	3
2. 스프링부트 프로젝트 만들기	4
가. 프로젝트 생성	4
나. dependency 설정	5
다. 프로젝트 구성	5
라. Application 테스트	8
마. Application 배포와 실행	8
3. 스프링부트웹 Maven, Mybatis, Oracle 이용	10
가. 프로젝트 생성	10
나. Dependency 설정	10
다. 의존성 구성 확인	10
라. Mybatis 환경설정	11
마. mapper	12
바. 컨트롤러	13
사. 뷰 페이지	14
아. 브라우저에서 테스트	14
4. thymeleaf	15
가. 타임리프 소개	15
나. using text	15
다. 표준표현식( thymeleaf tutorial )	15
라. 속성값 지정	17
마. 반복문 ( th:each )	17
바. 조건문	18
사. 주석	18
아. 인라인	19
자. Thymeleaf Layout Dialect	19
차. jsp 와 thymeleaf 비교	21

Spring	2
<hr/>	
5. 부록	22
가. junit	22
나. 스프링부트에서 JSP 사용	22
다. 웹프로젝트 예제	23
라. Lombok 라이브러리	24
마. postman	25

## 1. 스프링부트 개요

### 가. 탄생 배경

컨테이너 없는 웹 애플리케이션 아키텍처 구현을 목표로 2014년 4월 출시  
스프링을 개발하려면 서블릿 컨테이너를 설치하고 까다로운 환경 설정을 해야 한다.  
스프링의 단점이 개발환경 설정을 최소화하고 개발자가 비즈니스 로직에 집중할 수 있도록 한다.

### 나. 스프링부트 기능

- (1) 스프링 부트는 스프링 프레임워크를 기반한 개발 프레임워크다.
- (2) 내장형 톰캣, 제티 혹은 언더토우 컨테이너를 내장하고 있어 단독 실행 가능한 스프링 애플리케이션을 생성한다.
- (3) 스타터를 통해 간결한 의존성 구성 지원
- (4) 스타터를 통해 스프링에 대한 자동구성(auto-configuration) 제공하고 설정을 위한 XML 코드를 요구하지 않는다.
- (5) 상용화에 필요한 통계, 상태 점검, 모니터링 및 외부설정을 제공한다.

### 다. 스프링과 스프링부트 차이점

스프링 부트는 스프링 프레임워크라는 큰 틀에 속하는 도구일 뿐이다. 단지 'Just Run'에 가까워지도록 많은 설정을 간소화하기 위해 노력한 도구이다.

개발자 <---> 스프링부트 <---> 스프링

### 라. 스프링부트 구성요소

- 빌드도구(gradle or maven)
- 스프링 프레임워크(4.X or 5.X)
- 스프링부트
- 스프링부트 스타터 : 특정 모듈을 달성하기 위한 의존성 그룹

### 마. 스프링부트 스타터

참고사이트 : <https://docs.spring.io/spring-boot/docs/2.7.2/reference/htmlsingle/#using.build-systems.starters>

The following application starters are provided by Spring Boot under the `org.springframework.boot` group:

Name	Description
<code>spring-boot-starter</code>	Core starter, including auto-configuration support, logging and YAML
<code>spring-boot-starter-activemq</code>	Starter for JMS messaging using Apache ActiveMQ
<code>spring-boot-starter-amqp</code>	Starter for using Spring AMQP and Rabbit MQ
<code>spring-boot-starter-aop</code>	Starter for aspect-oriented programming with Spring AOP and AspectJ
<code>spring-boot-starter-artemis</code>	Starter for JMS messaging using Apache Artemis
<code>spring-boot-starter-batch</code>	Starter for using Spring Batch
<code>spring-boot-starter-cache</code>	Starter for using Spring Framework's caching support
<code>spring-boot-starter-data-cassandra</code>	Starter for using Cassandra distributed database and Spring Data Cassandra
<code>spring-boot-starter-data-cassandra-reactive</code>	Starter for using Cassandra distributed database and Spring Data Cassandra Reactive
<code>spring-boot-starter-data-couchbase</code>	Starter for using Couchbase document-oriented database and Spring Data Couchbase

스타터명	설명
spring-boot-starter	스프링 부트의 코어. auto-configuration, logging, yaml 제공
spring-boot-starter-aop	관점지향 프로그래밍(aspect-oriented programming, AOP)을 위한 스타터
spring-boot-starter-batch	스프링 배치를 사용하는 데 필요한 스타터
spring-boot-starter-data-jpa	스프링 데이터 JPA와 하이버네이트를 사용하는데 필요한 스타터
spring-boot-starter-data-redis	메모리 저장 방식의 저장소인 레디스와 자바에서 쉽게 레디스를 사용하게끔 도와주는 제디스 설정 자동화 스타터
spring-boot-starter-data-rest	스프링 데이터 저장소 방식에 맞춘 REST API를 제공하는 데 사용하는 스타터
spring-boot-starter-thymeleaf	타임리프 템플릿 엔진을 사용하는 데 필요한 스타터
spring-boot-starter-jdbc	톰캣 JDBC 커넥션 풀에 사용하는 스타터
spring-boot-starter-security	각종 보안에 사용하는 스프링 시큐리티 스타터
spring-boot-starter-oauth2	OAuth2 인증에 사용하는 스타터
spring-boot-starter-validation	자바 빈 검증에 사용하는 스타터
spring-boot-starter-web	웹을 만드는 데 사용하는 스타터(스프링, MVC, REST형, 임베디드 톰캣, 기타 라이브러리 포함)

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

## 2. 스프링부트 프로젝트 만들기

### 가. 프로젝트 생성

**New Spring Starter Project**

Service URL: 1

Name:

☒ Use default location

Location:

Type: 2  Packaging: 3

Java Version: 4  Language:

Group:

Artifact:

Version:

Description:

Package:

Working sets

☐ Add project to working sets

Working sets:

(1) Service URL : <https://start.spring.io>

스프링 부트 프로젝트를 생성하여 zip파일로 다운로드 받을 수도 있다.

(2) type

- maven : 메이븐 설정 파일인 pom.xml은 XML 기반으로 작성되어 있어서 동적인 제약이 있다.
- gradle : 그레이들은 Ant의 기본적인 빌드 도구의 기능을, 메이븐으로부터 의존 라이브러리 관리 기능을 차용했으며 성능까지 좋다. Gradle은 그루비를 사용하기 때문에, 동적인 빌드는 Groovy 스크립트로 플러그인을 호출하거나 직접 코드를 짜면 된다.

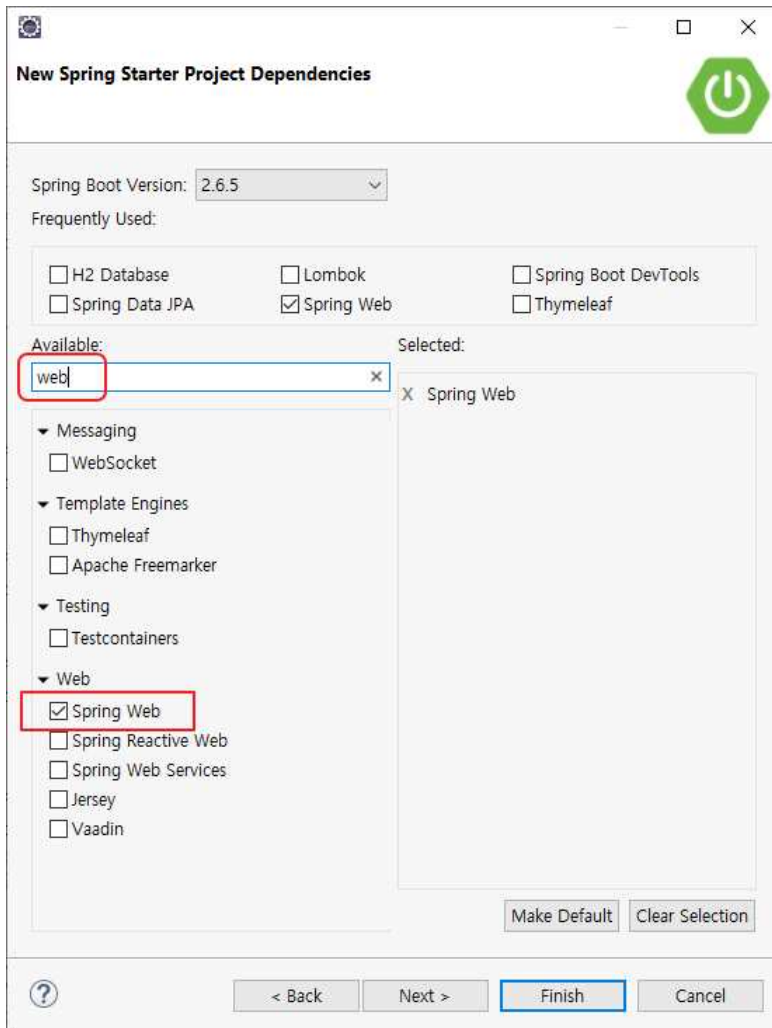
(3) Package

- jar : 자바프로젝트를 압축한 파일. 성된 jar파일은 JVM이 있는 환경이면 어디서나 실행할 수 있다.
- war : servlet/jsp 컨테이너에 배포할 수 있는 웹어플리케이션 압축파일. WEB-INF 등 사전 정의된 구조를 사용하며, 실행하기 위해서 Tomcat 같은 웹컨테이너(WAS)가 필요하다

jar 와 war 패키징 방식은 어떻게 배포하는가에 따라 달라짐. 스프링부트는 기본적으로 실행가능한 jar로 애플리케이션을 배포한다. 스프링부트는 빌드시 재포장 작업을 한다. 처음에 애플리케이션과 필요한 라이브러리를 묶고나서 내장 컨테이너를 넣고 다시 포장한다.

## 나. dependency 설정

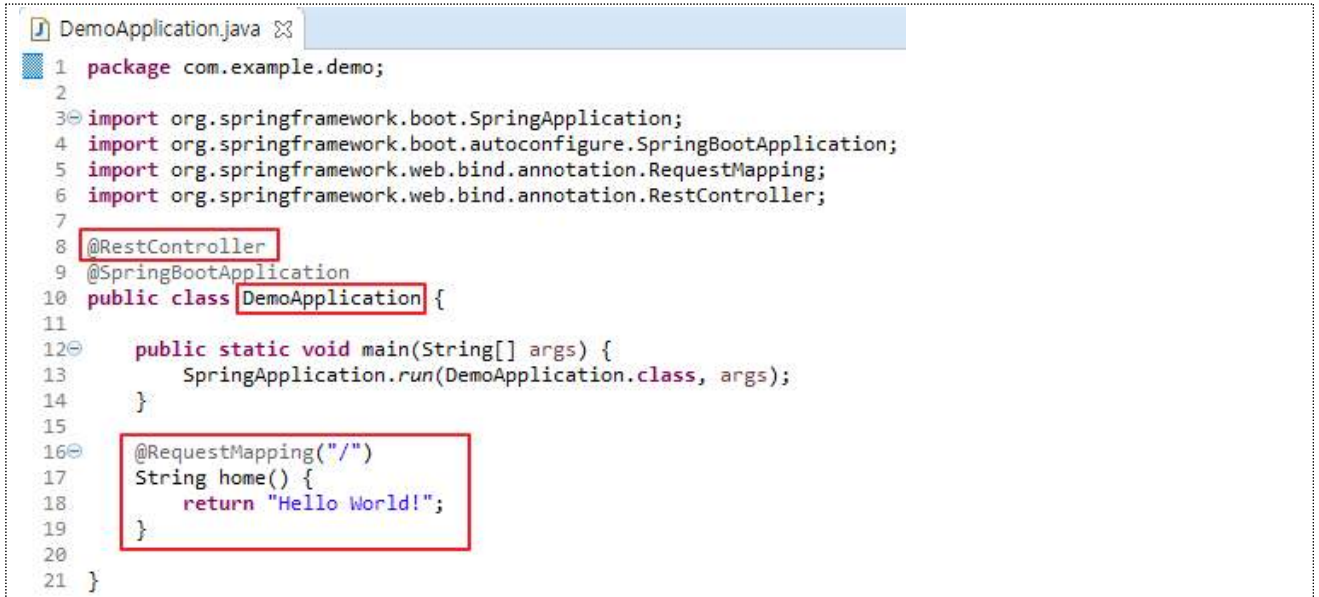
필요할 모듈을 선택하면 자동으로 pom.xml 설정함.



## 다. 프로젝트 구성

	<ul style="list-style-type: none"> <li>- src/main/java : 자바 소스 경로 <b>DemoApplication</b> : 프로젝트의 메인 클래스</li> <li>- src/main/sources : 자바 소스를 제외한 설정파일과 뷰페이지 경로 template : thymeleaf, freemarker 등 서버 사이트 템플릿 파일의 경로 static : static한 파일(css, image, js 등)의 디폴트 경로 <b>application.properties</b> : 스프링부트 설정 파일</li> <li>- src/test/java : 스프링부트의 테스트 코드 경로</li> <li>- <b>build.gradle</b> : 의존성이나 플러그인 설정, 빌드 등을 위한 설정 파일. maven의 pom.xml과 같은 역할</li> </ul>
--	--

## (1) DemoAppApplication



```

1 package com.example.demo;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5 import org.springframework.web.bind.annotation.RequestMapping;
6 import org.springframework.web.bind.annotation.RestController;
7
8 @RestController
9 @SpringBootApplication
10 public class DemoApplication {
11
12     public static void main(String[] args) {
13         SpringApplication.run(DemoApplication.class, args);
14     }
15
16     @RequestMapping("/")
17     String home() {
18         return "Hello World!";
19     }
20
21 }

```

- Application 클래스는 앞으로 만들 프로젝트의 메인 클래스로서 스프링부트가 시작되는 곳.
- @SpringBootApplication이 있는 위치부터 설정을 읽어가기 때문에 Application 클래스는 항상 프로젝트 최상단에 위치해야한다.

## (가) 실행과정

1. SpringApplication 인스턴스를 생성하고 run 실행명령을 내린다.
2. @SpringBootApplication으로 인해 스프링부트의 자동 설정 어노테이션으로 스프링 Bean 읽기, 생성, Web, H2, JDBC를 비롯해 약 100여개의 자동 설정을 제공한다.
3. main 메소드의 SpringApplication.run은 내장WAS를 실행한다.
4. DemoAppApplication을 실행하면 Spring Boot의 SpringApplication 클래스에 위임한다.
5. SpringApplication은 Spring을 시작하여 애플리케이션을 부트스트랩하고 자동으로 구성된 Tomcat 웹 서버를 시작한다.

## (나) 어노테이션

- @SpringBootApplication
  - = @SpringBootConfiguration // 스프링 부트의 설정
  - + @EnableAutoConfiguration // 자동설정. 설정값이 없으며 기본값으로 작동
  - + @ComponentScan // 경로를 지정하지 않으면 클래스 있는 패키지가 루트 경로로 설정
- @ComponentScan
- @EnableAutoConfiguration
- @Configuration
- @ConditionalOn~~
- @SpringBootConfiguration(= @Configuration)
- @EnableConfigurationProperties
- @ConfigurationProperties

## (2) application.properties

- 외부설정파일로서 애플리케이션에서 사용하는 여러가지 설정 값들을 정의
- 이 파일은 스프링부트가 애플리케이션을 구동할 때 자동으로 로딩하는 파일이다.
- key - value 형식으로 값을 정의하면 애플리케이션에서 참조하여 사용할 수 있다.

```
server.port=80    #애플리케이션 서버 포트 설정 변경
```

참고 application.yml

YAML(Ain't Markup Language): 핵심이 문서 마크업이 아닌 데이터 중심에 있다. 프로퍼티 설정값의 깊이에 따라 들여쓰기를 해서 계층 구조를 훨씬 쉽게 파악할 수 있다.

```
server.port=80
spring.datasource.driver-class-name=oracle.jdbc.driver.OracleDriver
spring.datasource.url=jdbc:oracle:thin:@127.0.0.1:1521/xe
spring.datasource.username=hr
spring.datasource.password=hr
```

```
server:
  port: 80

spring:
  datasource:
    driverClassName: oracle.jdbc.driver.OracleDriver
    url: jdbc:oracle:thin:@127.0.0.1:1521/xe
    username: hr
    password: hr
```

## (3) build.gradle

```
plugins {
    id 'org.springframework.boot' version '2.2.1.RELEASE'
    id 'io.spring.dependency-management' version '1.0.8.RELEASE'
    id 'java'
}

group = 'com.example'
version = '0.0.1-SNAPSHOT'
sourceCompatibility = '1.8'

repositories {
    mavenCentral()
}

dependencies {
    implementation 'org.springframework.boot:spring-boot-starter-web'
    testImplementation('org.springframework.boot:spring-boot-starter-test') {
        exclude group: 'org.junit.vintage', module: 'junit-vintage-engine'
    }
}

test {
    useJUnitPlatform()
}
```



- plugins : 프로젝트를 빌드하기 위해 필요하 작업들을 지원하는 플러그인을 설정
- repositoryes : 프로젝트에서 사용되는 라이브러리를 가져오기 위한 저장소를 등록
- dependencies : 프로젝트에 필요한 라이브러리를 등록
- test : 테스트 과정에서 사용할 프레임워크를 지정

● 의존성 설정과 기본 버전 확인



---

<https://github.com/spring-projects/spring-boot/blob/main/spring-boot-project/spring-boot-dependencies/build.gradle>

<https://docs.spring.io/spring-boot/docs/2.2.1.RELEASE/reference/html/appendix-dependency-versions.html#appendix-dependency-versions>

## 라. Application 테스트

### (1) DemoAppApplication 실행

DemoAppApplication을 Run As -> Java Application 실행



```

Console Progress Problems
DemoApplication [Java Application] C:\Program Files\Java\jre1

:: Spring Boot ::
(v2.2.1.RELEASE)

main] com.example.demo.DemoApplication : Starting DemoApplication on chichi-PC with PID 8668 (C:\dev\works;
main] com.example.demo.DemoApplication : No active profile set, falling back to default profiles: default
main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 80 (http)
main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.27]
main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
main] o.s.web.context.ContextLoader : Root WebApplicationContext: initialization completed in 4644 ms
main] o.s.s.concurrent.ThreadPoolTaskExecutor : Initializing ExecutorService 'applicationTaskExecutor'
  
```

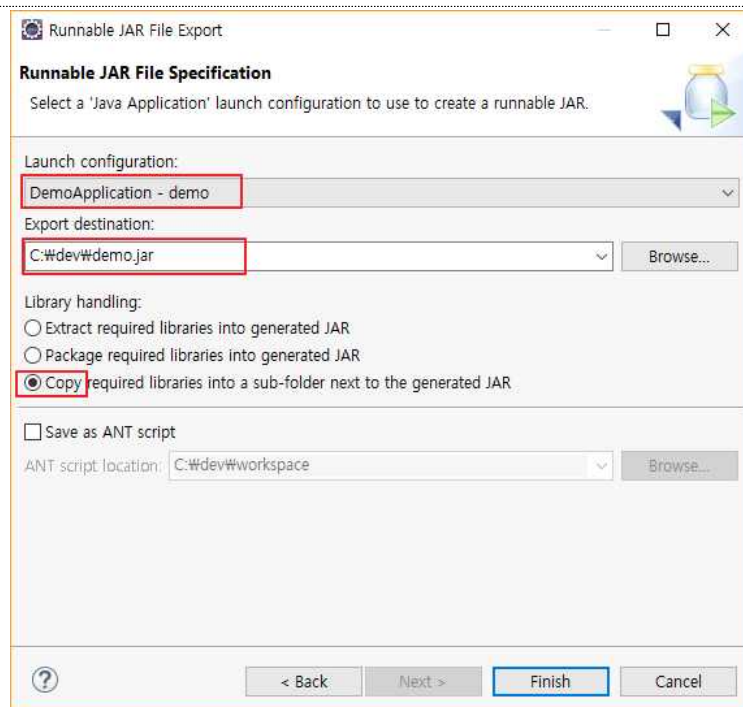
### (2) 브라우저에서 테스트



## 마. Application 배포와 실행

### (1) gradle 프로젝트인 경우 jar 배포파일 생성

export --> Runnable JAR File



- manifest.mf 파일이 추가됨.  
jar 압축시 패키지 관련 정보 및 파일 확장 관련 정보를 저장

maven 프로젝트인 경우는 "run as" -> maven install 명령 실행하면 target 폴더에 jar 파일이 만들어짐

## (2) jar 실행

```
c:\demo> java -jar demo.jar
```

C:\dev> java -jar demo.jar



### ● 포트 충돌로 서버 시작 시 에러가 나면

```
c:\demo> java -Dserver.port=80 -jar demo.jar
```

## (3) 테스트



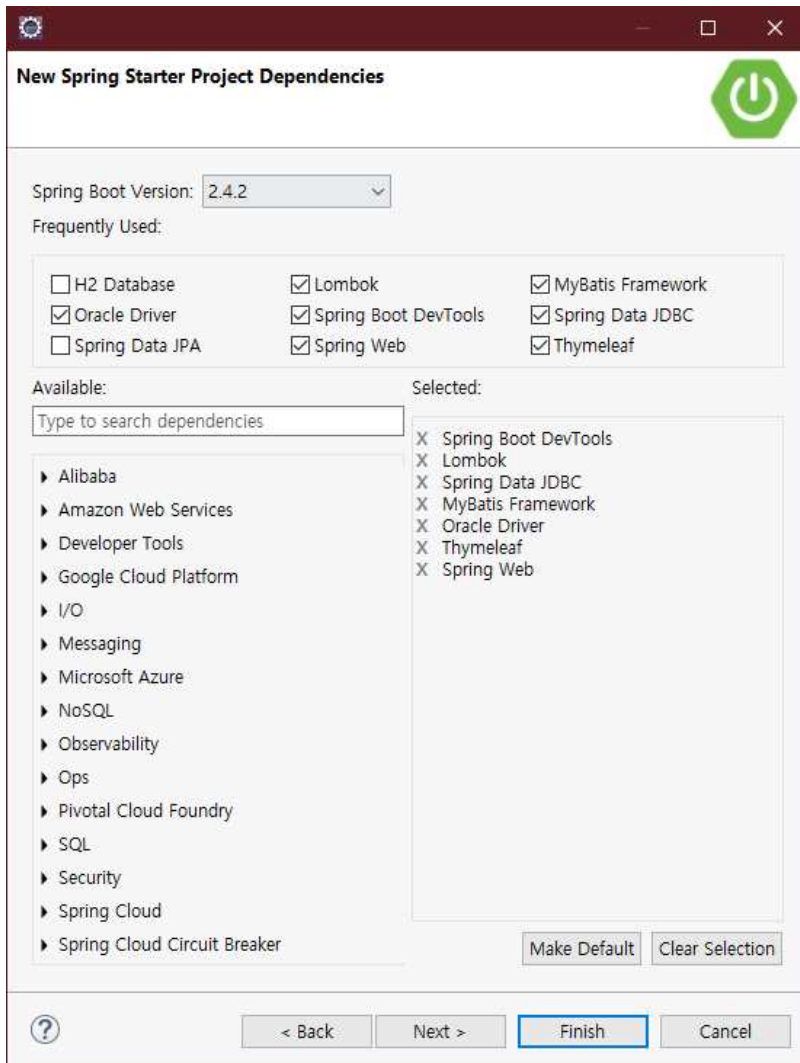
### 3. 스프링부트 웹 Maven, Mybatis, Oracle 이용

#### 가. 프로젝트 생성

maven 선택

#### 나. Dependency 설정

Devtools, Lombok, Mybatis, Oracle driver, Thymeleaf, Spring Web



#### 다. 의존성 구성 확인

- pom.xml

```
<artifactId>spring-boot-starter-thymeleaf</artifactId>
<artifactId>spring-boot-starter-web</artifactId>
<artifactId>mybatis-spring-boot-starter</artifactId>

<artifactId>lombok</artifactId>
<scope>provided</scope>

<artifactId>spring-boot-devtools</artifactId>
<scope>runtime</scope>
<optional>true</optional>

<artifactId>ojdbc8</artifactId>
<scope>runtime</scope>

<artifactId>spring-boot-starter-test</artifactId>
<scope>test</scope>
```

## 라. Mybatis 환경설정

참고사이트 : <http://mybatis.org/spring-boot-starter/mybatis-spring-boot-autoconfigure/>

### (1) MyBatis-Spring-Boot-Starter module 설치

- gradle

```
dependencies {
    implementation 'org.mybatis.spring.boot:mybatis-spring-boot-starter:2.2.2'
}
```

- maven

```
<dependency>
  <groupId>org.mybatis.spring.boot</groupId>
  <artifactId>mybatis-spring-boot-starter</artifactId>
  <version>2.2.2</version>
</dependency>
```

### (2) application.properties 파일 설정

```
server.port=80

#datasource (oracle)
spring.datasource.driver-class-name=oracle.jdbc.driver.OracleDriver
spring.datasource.url=jdbc:oracle:thin:@127.0.0.1:1521/xe
spring.datasource.username=hr
spring.datasource.password=hr

#mybatis 설정
mybatis.configuration.map-underscore-to-camel-case=true
mybatis.configuration.jdbc-type-for-null=VARCHAR
mybatis.type-aliases-package=com.example.demo
mybatis.mapper-locations=classpath:mapper/*.xml

#로그설정
logging.level.com.example.demo=DEBUG
```

### (3) Mybatis MapperScan 설정

Application 클래스에 설정을 추가하는 경우 @MapperScan 만 추가하면 됨.

```
@SpringBootApplication
@MapperScan(basePackages = "com.example.demo.**.mapper")
public class Demo1Application {
    private static final Logger log = LoggerFactory.getLogger(Demo1Application.class);
    public static void main(String[] args) {
        SpringApplication.run(Demo1Application.class, args);
    }
}
```

별도의 Configuration 클래스를 만들어서 설정을 추가할 수도 있다.

```
@Configuration
@MapperScan(basePackages = "com.example.demo.**.mapper")
public class MybatisConfig { }
```

#### (4) mybatis-config 설정

**mybatis-config.xml**

```

1 <?xml version="1.0" encoding="UTF-8" ?>
2 <!DOCTYPE configuration
3 PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
4 "http://mybatis.org/dtd/mybatis-3-config.dtd">
5 <configuration>
6   <settings>
7     <setting name="mapUnderscoreToCamelCase" value="true"/>
8     <setting name="jdbcTypeForNull" value="VARCHAR"/>
9   </settings>
10
11   <typeAliases>
12     <typeAlias type="com.yedam.lms.class.ClassVO" alias="ClassVO"/>
13     <package name="com" />
14   </typeAliases>
15 </configuration>

```

**application.properties**

```

server.port=80

#datasource (oracle)
spring.datasource.driver-class-name=oracle.jdbc.driver.OracleDriver
spring.datasource.url=jdbc:oracle:thin:@127.0.0.1:1521/xe
spring.datasource.username=hr
spring.datasource.password=hr

#mybatis 설정
mybatis.configuration.map-underscore-to-camel-case=true
mybatis.configuration.jdbc-type-for-null=VARCHAR
mybatis.type-aliases-package=com.example.demo
mybatis.mapper-locations=classpath:mapper/*.xml

#로그설정
logging.level.com.example.demo=DEBUG

```

#### (5) spring과 springboot 설정 비교

**datasource-context.xml**

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4       xmlns:context="http://www.springframework.org/schema/context"
5       xmlns:mybatis-spring="http://mybatis.org/schema/mybatis-spring"
6       xsi:schemaLocation="http://mybatis.org/schema/mybatis-spring http://mybatis.org/schema/mybatis-spring.xsd
7       http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans.xsd
8       http://www.springframework.org/schema/context http://www.springframework.org/schema/context.xsd">
9
10   <context:property-placeholder location="classpath:global.properties"/>
11
12   <!-- DBCP -->
13   <bean class="org.apache.commons.dbcp2.BasicDataSource" name="dataSource">
14     <property name="driverClassName" value="${driver}" />
15     <property name="url" value="${url}" />
16     <property name="username" value="${user}" />
17     <property name="password" value="${password}" />
18   </bean>
19
20   <!-- Mybatis -->
21   <bean class="org.mybatis.spring.SqlSessionFactoryBean" id="sqlSessionFactoryBean">
22     <property name="dataSource" ref="dataSource" />
23     <property name="configLocation" value="classpath:/mybatis-config.xml" />
24     <property name="mapperLocations" value="classpath*:com/yedam/app/**/impl/*-map.xml" />
25   </bean>
26
27   <!-- SqlSession -->
28   <bean class="org.mybatis.spring.SqlSessionTemplate" id="sqlSessionTemplate">
29     <constructor-arg ref="sqlSessionFactoryBean" />
30   </bean>
31
32   <!-- mapper scan -->
33   <mybatis-spring:scan base-package="com.yedam.app.**.impl" />

```

**application.properties**

```

server.port=80

#datasource (oracle)
spring.datasource.driver-class-name=oracle.jdbc.driver.OracleDriver
spring.datasource.url=jdbc:oracle:thin:@127.0.0.1:1521/xe
spring.datasource.username=hr
spring.datasource.password=hr

#mybatis 설정
mybatis.configuration.map-underscore-to-camel-case=true
mybatis.configuration.jdbc-type-for-null=VARCHAR
mybatis.type-aliases-package=com.example.demo
mybatis.mapper-locations=classpath:mapper/*.xml

#로그설정
logging.level.com.example.demo=DEBUG

```

**BootMybatisApplication.java**

```

1 package com.example.demo;
2
3 import org.mybatis.spring.annotation.MapperScan;
4
5
6
7 @SpringBootApplication
8 @MapperScan(basePackages = "com.example.demo.**.mapper")
9 public class BootMybatisApplication {
10
11     public static void main(String[] args) {
12         SpringApplication.run(BootMybatisApplication.class, args);
13     }
14 }

```

### 마. mapper

#### (1) EmpVO

```

@Data
@NoArgsConstructor
public class EmpVO {
    String employeeId;
    String firstName;
    String lastName;
    String email;
    Date hireDate;
    int salary;
    String jobId;
    String departmentId;
    String managerId;
}

```

#### (2) emp-mapper.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="com.example.demo.emp.mapper.EmpMapper">
    <select id="getEmpList" resultType="EmpVO">
        SELECT rownum id, e.*
        FROM employees e
    </select>
</mapper>
```

### (3) EmpMapper.java

```
//@Mapper
public interface EmpMapper {
    public EmpVO getEmp(EmpVO empVO);
    public List<EmpVO> getEmpList(EmpVO empVO);
    public void empInsert(EmpVO empVO);
}
```

### (4) Mapper 테스트

- junit5 라이브러리 추가
- test 클래스

```
@RunWith(SpringRunner.class)
@SpringBootTest
class MapperTest {

    @Autowired EmpMapper mapper;

    @Test
    void list() {
        mapper.getEmpList(null);
    }
}
```

## 바. 컨트롤러

- 컨트롤러 클래스

```
@Controller
public class EmpController {

    @Autowired EmpMapper dao;

    @RequestMapping("/empList");
    public String empList(Model model){
        model.addAttribute("empList", dao.getEmpList(null));
        return "empList";
    }
}
```

- 컨트롤러 테스트



```

import static org.junit.Assert.assertThat;
import org.junit.Test;
import org.junit.runner.RunWith;
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.test.context.junit4.SpringRunner;

@RunWith(SpringRunner.class)
@SpringBootTest(webEnvironment = RANDOM_PORT)
public class WebControllerTest {

    @Autowired
    private TestRestTemplate restTemplate;

    @Test
    public void 메인페이지_로딩() {
        //when
        String body = this.restTemplate.getForObject("/", String.class);

        //then
        assertThat(body).contains("스프링부트로 시작하는 웹 서비스");
    }
}

```

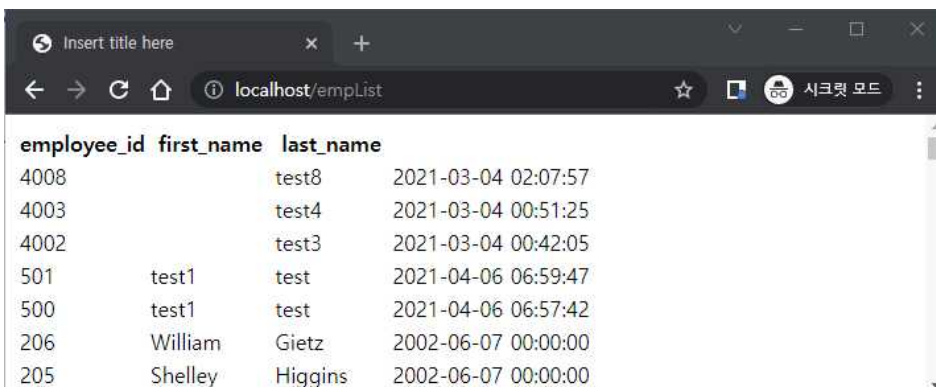
## 사. 뷰 페이지

```

<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
<table>
<tr>
<th>employee_id</th>
<th>first_name</th>
<th>last_name</th>
</tr>
<tr th:each="emp : ${empList}">
<td th:text="${emp.employeeId}"></td>
<td th:text="${emp.firstName}"></td>
<td th:text="${emp.lastName}"></td>
<td th:text="${emp.hireDate}"></td>
</tr>
</table>
</body>
</html>

```

## 아. 브라우저에서 테스트



The screenshot shows a web browser window with the address bar displaying 'localhost/empList'. The page content is a table with four columns: 'employee\_id', 'first\_name', 'last\_name', and an unlabeled column for dates. The table contains seven rows of data.

employee_id	first_name	last_name	
4008		test8	2021-03-04 02:07:57
4003		test4	2021-03-04 00:51:25
4002		test3	2021-03-04 00:42:05
501	test1	test	2021-04-06 06:59:47
500	test1	test	2021-04-06 06:57:42
206	William	Gietz	2002-06-07 00:00:00
205	Shelley	Higgins	2002-06-07 00:00:00



## 4. thymeleaf Thymeleaf

참고사이트 : <https://dev-gorany.tistory.com/302>

### 가. 타임리프 소개

- (1) 자바 기반의 서버 사이드 뷰 **템플릿 엔진**으로 동적 콘텐츠를 생성한다.
- (2) 스프링부트가 자동설정을 지원하는 템플릿 중에 하나이다.
  - FreeMaker, Groovy, Thymeleaf, Mustache
  - JSP는 자동설정을 지원하지 않는다. 의존성 문제 발생하고 jar 패키징을 할 수 없음.
- (3) **Natural Templates** : 타임리프의 주 목표는 유지관리가 쉬운 템플릿 생성 방법을 제공하는 것이며, 실제로 템플릿에 영향을 주지 않는(HTML의 구조를 깨지 않는, 기존 HTML 코드를 변경하지 않고 덧붙이는 코드) 방식을 사용한다. 이를 통해 디자인 팀과 개발 팀간 갈등, 격차 해소.  
 디자이너와 개발자가 동일한 템플릿 파일에서 작업하고 정적인 HTML을 동적인 HTML로 변환하는데 필요한 노력을 줄일 수 있다.
- (4) standard Dialect  
 Attribute Processor : 브라우저는 단순히 추가 속성을 무시하기 때문에 처리되기 전에도 HTML 템플릿 파일을 올바르게 표시할 수 있다.

### 나. using text

- Thymeleaf 사용하기 위해 th 네임스페이스 추가

thymeleaf	jsp
<pre>&lt;!DOCTYPE html&gt; &lt;html lang="ko" xmlns:th="http://www.thymeleaf.org"&gt; &lt;head&gt;   &lt;meta charset="UTF-8"&gt;   &lt;title&gt;Title&lt;/title&gt; &lt;/head&gt; &lt;body&gt;   &lt;h1 th:text="\${name}"&gt;Name&lt;/h1&gt; &lt;/body&gt; &lt;/html&gt;</pre>	<pre>&lt;%@ page language="java" contentType="text/html; charset=UTF-8"     pageEncoding="UTF-8"%&gt; &lt;%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %&gt; &lt;!DOCTYPE html&gt;  &lt;h1&gt;\${name}&lt;/h1&gt;</pre>

### 다. 표준표현식( [thymeleaf tutorial](#) )

- **\${ }** : 변수 표현. Controller로부터 받은 model에서 값을 꺼내 사용
- **#{ }** : 메시지 표현. properties 파일에서 메시지 값을 꺼내 사용
- **\*{ }** : 객체(선택)변수 표현식
- **@{ }** : 링크표현식
- **~{ }** : Fragment 표현식

#### (1) variable

(가) ○○○○

```
// 해당 변수의 값이 text로 출력
<p th:text="${test}"></p>
```

```
// 해당 변수의 값이 href 링크로 출력
<p th:text="${test}"></p>
```

#### (나) object

#ctx	the context object
#vars	the context variables
#locale	the context locale.
#request	(only in Web Contexts) the HttpServletRequest object.
#response	(only in Web Contexts) the HttpServletResponse object.
#session	(only in Web Contexts) the HttpSession object.
#servletContext	(only in Web Contexts) the ServletContext object.
#execInfo	information about the template being processed.
#messages	methods for obtaining externalized messages inside variables expressions, in the same way as they would be obtained using #{...} syntax.
#uris	methods for escaping parts of URLs/URIs
#conversions:	methods for executing the configured conversion service (if any).
#dates	methods for java.util.Date objects: formatting, component extraction, etc.
#calendars	analogous to #dates, but for java.util.Calendar objects.
#numbers	methods for formatting numeric objects.
#strings	methods for String objects: contains, startsWith, prepending/appending, etc.
#objects:	methods for objects in general.
#bools	methods for boolean evaluation.
#arrays:	methods for arrays.
#lists	methods for lists.
#sets	methods for sets.
#maps	methods for maps.
#aggregates	methods for creating aggregates on arrays or collections.
#ids	methods for dealing with id attributes that might be repeated (for example, as a result of an iteration).

Strings : String객체 메서드

```
<td th:text="${#strings.substring(emp.hireDate,0,4)}">2000년</td>
```

#### (2) message

#### (3) Expression on selection(asterisk)

#### (4) Link url

#### (5) Fragments

#### (6) Literals

#### (7) 연산자

##### (가) 삼항연산자

- ? :

```
<td th:text="${emp.commissionPct} ? ${emp.commissionPct*100} : 'no' ">없음</td>
```

- ?:

```
<td th:text="${emp.commissionPct} ?: 'no' ">없음 </td>
```

- ? : \_ (No-Operation)

토큰은 밑줄 기호(\_)로 표시. 타임리프가 실행되지 않는 것처럼 동작되며 프로토타이핑 텍스트를 기본값으로 사용

```
<td th:text="${emp.commissionPct}${emp.commissionPct*100}:_ ">없음 </td>
```

## 라. 속성값 지정

(가) 속성값 변경 : th:attr

```
<input value="hong" th:attr="value=#{vo.name}"/>
```

```

```

(나) 특정 속성 변경 : th:class, th:style, th:id, th:method, th:checked 등

```

```

```

```

(다) 속성값 추가 : th:attrappend, th:attrprepend

```
<input type="button" value="Do it!" class="btn" th:attrappend="class=${' ' + cssStyle}" />
```

(라) 고정값 불린 속성

속성 자체가 true를 나타내고 하나의 값만 가짐

```
<input type="checkbox" name="option2" checked /> <!-- HTML -->
<input type="checkbox" name="option1" checked="checked" /> <!-- XHTML -->
```

th:async	th:autofocus	th:autoplay	th:checked	th:controls	th:declare
th:default	th:defer	th:disabled	th:formnovalidate	th:hidden	th:ismap
th:loop	th:multiple	th:novalidate	th:nowrap	th:open	th:pubdate
th:readonly	th:required	th:reversed	th:scoped	th:seamless	th:selected

```
<input type="checkbox" name="active" th:checked="${user.active}" />
```

## 마. 반복문 ( th:each )

```
<span th:each="emp, iterStat : ${empList}" th:text="${emp.firstName}"> </span>
```

```
<tr th:each="emp : ${empList}">
  <td th:text="${emp.employeeid}">100</td>
</tr>
```

- 반복 상태

두 번째 파라미터를 지정하여 반복문 실행 중에 상태를 확인

상태 변수를 명시적으로 설정하지 않으면 Thymeleaf는 항상 반복 변수 이름에 Stat을 접미사로 만들어 생성 index, count, size, even, odd, first, last, current

```
<tr th:each="emp, iterStat : ${empList}" th:class="${iterStat.odd}? 'odd'" >
```

```
<tr th:each="emp : ${empList}" th:class="${empStat.odd}? 'odd'">
```

- th:block

HTML 태그가 아닌 타임리프의 유일한 자체 태그이다. 가능하면 태그에 for loop를 넣는것을 추천

```
<th:block th:each="emp : ${empList}" >
```

반복할 내용

```
</th:block>
```

## 바. 조건문

```
<a href="comments.html"
  th:href="@{/product/comments(prodId=${prod.id})}"
  th:if="${not #lists.isEmpty(prod.comments)}">view</a>
```

```
<div th:switch="${user.role}">
  <p th:case="'admin'">User is an administrator</p>
  <p th:case="#{roles.manager}">User is a manager</p>
</div>
```

### (1) 객체

```
// Calendar -> String 으로 변환
// #Strings : String객체 메서드
<td th:text="${#strings.substring(emp.hireDate,0,4)}">2000년</td>
```

## 사. 주석

```
<!-- html 주석 -->
```

타임리프 렌더링에서 주석 부분을 제거

정적인 페이지에서는 주석으로 보이다가 thymeleaf 처리가 될때 제거되는 주석으로 불필요한 주석을 클라이언트에 노출시키지 않게 처리가 가능

```
<!--/* 타임리프 파서 주석 */-->
```

HTML 파일을 그대로 열어보면 주석처리되고 타임리프를 통해 렌더링 한 경우에만 출력.

parser-level 주석과 반대되는 개념으로 정적페이지에서는 주석으로 처리가 되고 thymeleaf 처리후 나타나게 되는 주석

```
<!--*/ 타임리프 프로토타입 주석 */-->
```

<pre>&lt;p th:utext="\${name}"&gt;utext연습입니다.&lt;/p&gt; &lt;hr&gt; &lt;p&gt;html 주석: &lt;!-- &lt;b&gt; [[\${name}]] &lt;/b&gt; --&gt; &lt;/p&gt; &lt;hr&gt; &lt;p&gt;타임리프 파서 주석: &lt;!--*/ &lt;b&gt; [[\${name}]] &lt;/b&gt; &lt;hr&gt; &lt;p&gt;타임리프 프로토타입 주석: &lt;!--*/ &lt;b&gt; [[\${name}]] &lt;/b&gt;</pre>	<pre>홍길동 html 주석: 타임리프 파서 주석: 타임리프 프로토타입 주석: 홍길동</pre>	<pre>&lt;p&gt;홍길동&lt;/p&gt; &lt;hr&gt; &lt;p&gt;html 주석: &lt;!-- &lt;b&gt; 홍길동 &lt;/b&gt; --&gt; &lt;/p&gt; &lt;hr&gt; &lt;p&gt;타임리프 파서 주석: &lt;/p&gt; &lt;hr&gt; &lt;p&gt;타임리프 프로토타입 주석: &lt;b&gt; 홍길동</pre>
---	--	---

## 아. 인라인

### (1) Expression inlining

- 비활성화. 내용을 표현식으로 처리하지 않는다.
- Standard Dialect를 사용하면 태그 속성을 사용하여 거의 모든 작업을 수행할 수 있지만 HTML 텍스트에 직접 표현식을 작성하려면 데이터 변환을 적용할 수 있는 변수(\${...}) 및 선택(\*{...}) 표현식에 대한 이중 중괄호를 사용
- [[...]] or [{...}] 이중괄호로 표시

```
<b> [[${name}]] </b>
```

```
<p th:text="${name}">김유신</p>
```

### (2) Inlining vs natural templates

인라인 표현식은 HTML 파일을 정적으로 열 때 그대로 표시된다는 점을 항상 기억해야 합니다. 따라서 디자인 프로토타입으로 사용할 수는 없다.

#### 1) 자바스크립트 inline

```
<script th:inline="javascript"> </script>
```

JavaScript serialization

### (3) css inline

## 자. Thymeleaf Layout Dialect

참고사이트 : <https://ultraq.github.io/thymeleaf-layout-dialect/>

A dialect for Thymeleaf that lets you build layouts and reusable templates in order to improve code reuse.

### (1)

```
<dependency>
  <groupId>nz.net.ultraq.thymeleaf</groupId>
  <artifactId>thymeleaf-layout-dialect</artifactId>
</dependency>
```

### (2) layout template 페이지 작성

- layout.html

```
<!DOCTYPE html>
<html xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout">
<head>
  <title>Layout page</title>
  <script src="common-script.js"> </script>
</head>
<body>
  <header>
    <h1>My website</h1>
  </header>
  <section layout:fragment="content">
    <p>Page content goes here</p>
  </section>
  <footer>
    <p>My footer</p>
    <p layout:fragment="custom-footer">Custom footer here</p>
  </footer>
</body>
</html>
```

layout:fragment : 일치하는 content teplate fragement를 찾아서 대체되는 지점

(3) content template 페이지 작성

- content.html

```
<!DOCTYPE html>
<html xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
      layout:decorate="~{layout}">
<head>
  <title>Content page 1</title>
  <script src="content-script.js"> </script>
</head>
<body>
  <section layout:fragment="content">
    <p>This is a paragraph from content page 1</p>
  </section>
  <footer>
    <p layout:fragment="custom-footer">This is some footer content from content page 1</p>
  </footer>
</body>
</html>
```

```
<p layout:decorate="~{layout}" layout:fragment="custom-footer">
  This is some footer text from content page 2.
</p>
```

## 차. jsp 와 thymeleaf 비교

구분	JSP	thymeleaf
EL상수	<code>\${true} \${123} \${'nava'}</code>	
EL 연산자	<code>+ - * / % mod</code>	
	<code>&amp;&amp;    and or not</code>	
	<code>== &gt; &lt; &lt;= &gt;= eq ne lt gt le ge</code>	
	<code>empty</code>	
내장 객체	<code>\${pageContext.request.contextPath}</code> <code>\${pageContext.session}</code>	<code>#session</code> <code>#servletContext</code> <code>#request #locale #vars #ctx #response #application</code>
	<code>\${requestScope.x} \${x}</code>	<code>\${x}</code> <code>&lt;- context or as a request attribute.</code>
	<code>\${sessionScope.x}</code>	<code>\${session.x}</code>
	<code>\${applicationScope.x}</code>	<code>\${application.x}</code>
	<code>\${param.x}</code> <code>\${paramValues.x[0]}</code>	<code>\${param.x}</code>
	<code>\${header.x}, \${headValues.x}</code>	
	<code>\${cookie.X}</code> <code>\${initParam.x}</code>	
조건 처리	<code>&lt;c:if&gt;</code>	<code>th:if</code> <code>&lt;- true이면 실행</code> <code>th:unless</code> <code>&lt;- false이면 실행</code>
	<code>&lt;c:choose&gt;</code>	
반복 처리 컬렉션 없이 반복처리	<code>&lt;c:forEach items="\${empList}" var="emp"&gt;</code> <code>&lt;c:forEach items="\${empList}" var="emp"</code> <code>  varStatus="status"&gt;</code>	<code>&lt;tr th:each="emp: \${empList}" &gt;</code> <code>&lt;tr th:each="emp, status: \${empList}" &gt; &lt;- empStat</code>
	<code>  \${status.begin} &lt;- 시작값</code> <code>  \${status.end} &lt;- 끝값</code> <code>  \${status.step} &lt;- 증가값</code>	<code>  \${empStat.even} &lt;- 현재 반복이 짝수인지 여부 (boolean)</code> <code>  \${status.odd} &lt;- 현재 반복이 홀수인지 여부 (boolean)</code> <code>  \${status.size} &lt;- 총 요소 수</code>
	<code>  \${status.current}</code> <code>  \${status.index}</code> <code>  \${status.count}</code> <code>  \${status.first}</code> <code>  \${status.last}</code>	<code>  \${status.current} &lt;- 현재 요소</code> <code>  \${empStat.index} &lt;- 현재 반복 인덱스 (0부터시작)</code> <code>  \${status.count} &lt;- 현재 반복 인덱스 (1부터 시작)</code> <code>  \${status.first} &lt;- 현재 반복이 첫번째인지 여부 (boolean)</code> <code>  \${status.last} &lt;- 현재 반복이 마지막인지 여부 (boolean)</code>
	<code>&lt;c:forEach var="item" begin="1" end="5"&gt;</code> <code>&lt;c:fortokens&gt;</code>	<code>&lt;th:block th:each="num : \${#numbers.sequence(1,5)}"&gt;</code>
	<code>&lt;c:set&gt;</code>	<code>&lt;th:block th:with="temp = \${data}, example = \${foo}"&gt;</code> <code>  &lt;span th:text="\${temp}"&gt;text&lt;/span&gt;</code> <code>&lt;/th:block&gt;</code>
	<code>&lt;c:url value="/"&gt;</code>	<code>@{/}</code> <code>&lt;- ContextPath</code>
	<code>&lt;c:import&gt;</code> <code>&lt;c:redirect&gt;</code> <code>&lt;c:out&gt;</code> <code>&lt;c:out escapeXml="false" value="&lt;u&gt;태그&lt;/u&gt;" /&gt;</code>	<code>th:text</code> <code>&lt;- HTML로 이스케이프됨 (&amp;lt;로 변환)</code> <code>th:utext</code> <code>&lt;- unescaped (태그적용)</code>
JSTL fmt	<code>&lt;fmt:formatDate&gt;</code> <code>&lt;fmt:timezone&gt;</code> <code>&lt;fmt:parseDate&gt;</code> <code>&lt;fmt:formatNumber&gt;</code>	<code>\${#dates.format(date, 'yyyy-MM-dd')}</code> , <code>#calendars</code>  <code>\${#numbers.formatInteger(num,3)}</code>
	<code>\${fn:length(empList)}</code> <code>\${fn:join(arr[], str)}</code>	<code>\${#lists.size(empList)}</code> <code>#arrays</code>
	<code>\${fn:contains(str, str)}</code> <code>\${fn:replace( str, str, str)}</code> <code>\${fn:startsWith(str, str)}</code> <code>\${fn:split(str, str)}</code>	<code>\${#strings.contains(str, 'search')}</code>
	<code>\${fn:escapeXml(str)}</code>	<code>th:utext</code> <code>&lt;- HTML로 이스케이프 수행안함</code>
주석	<code>&lt;%-- --%&gt;</code>	<code>&lt;!--/* 타임리프 파서 주석 */--&gt;</code> <code>&lt;!--/* 타임리프 프로토타입 주석 */--&gt;</code>

## 5. 부록

### 가. junit

참고사이트 : <https://junit.org/junit5/docs/snapshot/user-guide/>  
<https://m.blog.naver.com/simpolor/221327833587>

#### ■ imprt

```
import static org.hamcrest.CoreMatchers.is;
import static org.hamcrest.MatcherAssert.assertThat;
import org.junit.jupiter.api.Test;
```

#### ■ Matcher is : s는 첫번째 파라미터와 자기 자신의 파라미터가 동일한지 여부를 체크

```
Assert.assertThat(result, CoreMatchers.is(10));
```

#### ■ junit 기본 지원 매치

```
assertThat("Sample string.", is(not(startsWith("Test"))));
```

JUnit에서 제공하는 매치는 org.hamcrest.CoreMatchers 클래스에 선언된 메서드를 통해 사용할 수 있다.

### 나. 스프링부트에서 JSP 사용

#### (1) 의존성 추가

Maven이면 pom.xml에 Gradle이면 build.gradle에 추가

#### ■ pom.xml

```
<dependencies>
  <dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>jstl</artifactId>
  </dependency>
  <dependency>
    <groupId>org.apache.tomcat.embed</groupId>
    <artifactId>tomcat-embed-jasper</artifactId>
  </dependency>
</dependencies>
```

#### ■ build.gradle

```
dependencies {
    compile('javax.servlet:jstl')
    compile("org.apache.tomcat.embed:tomcat-embed-jasper")
}
```

이클립스 컨텍스트 메뉴 -> Gradle -> Refresh Gradle Project 실행

#### (2) jsp 파일 폴더 지정

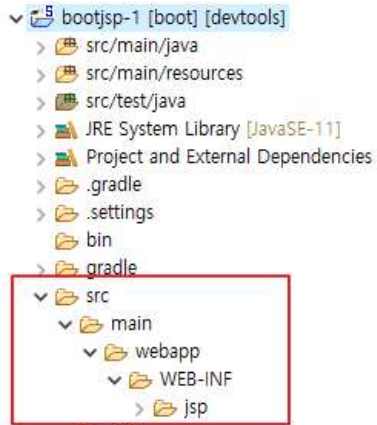
#### ■ application.properties



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

### ■ 폴더 생성

src/main/webapp/jsp



### ■ jsp 파일 생성

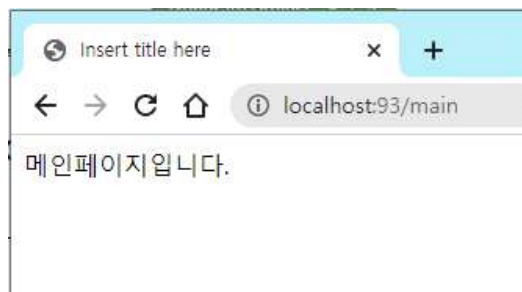
src/main/webapp/jsp/main.jsp

```
<body> 메인페이지입니다.</body>
```

### (3) controller 작성

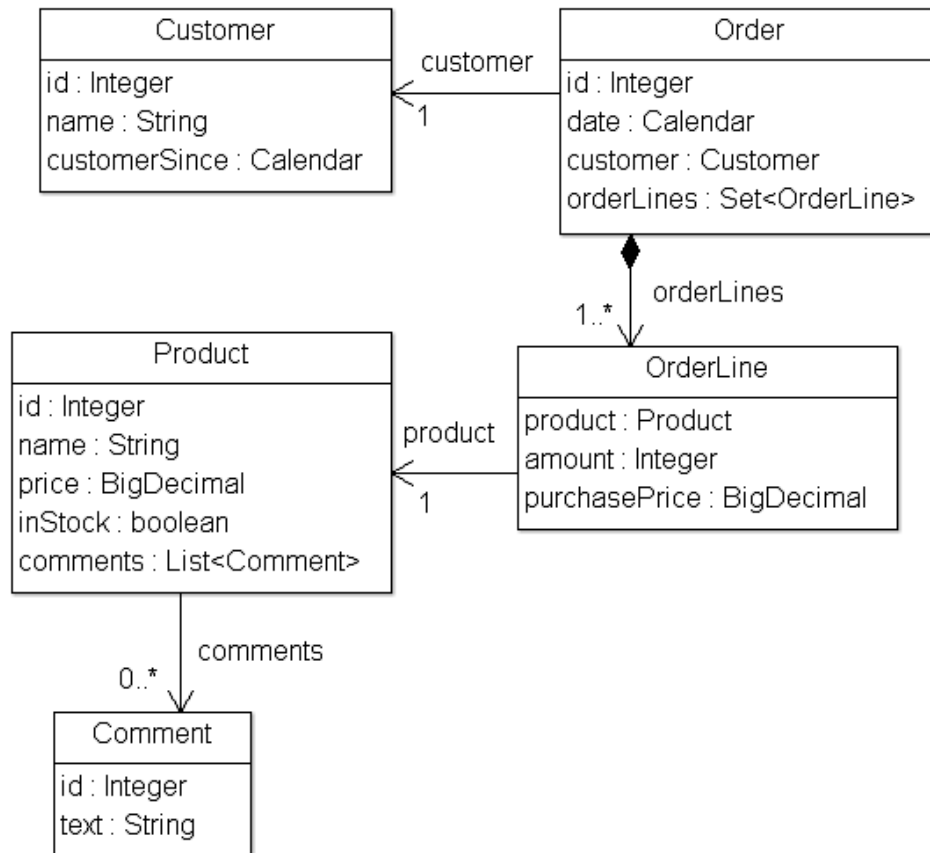
```
@Controller
@RequestMapping("/main")
public class TestController {
    @GetMapping
    public String main() {
        return "main";
    }
}
```

### (4) 테스트



## 다. 웹프로젝트 예제

참고사이트: <https://www.thymeleaf.org/doc/tutorials/3.0/usingthymeleaf.html#using-the-empty-fragment>



## 라. Lombok 라이브러리

- Lombok 은 에디터(IntelliJ, Eclipse...)와 빌드 도구(Maven, Gradle..)에 추가하여 사용하는 라이브러리이다.
- DTO(VO) Class 작성할 때 필수적으로 작성해야 하는 getter, setter, toString, 생성자 등을 직접 작성하지 않아도 된다.
- 코드를 간결하게 사용할 수 있다.
- 테이블 설계 시 코드 변경량을 최소화할 수 있다.

참고사이트 : <https://hyoj.github.io/blog/java/basic/lombok/#lombok-이론>

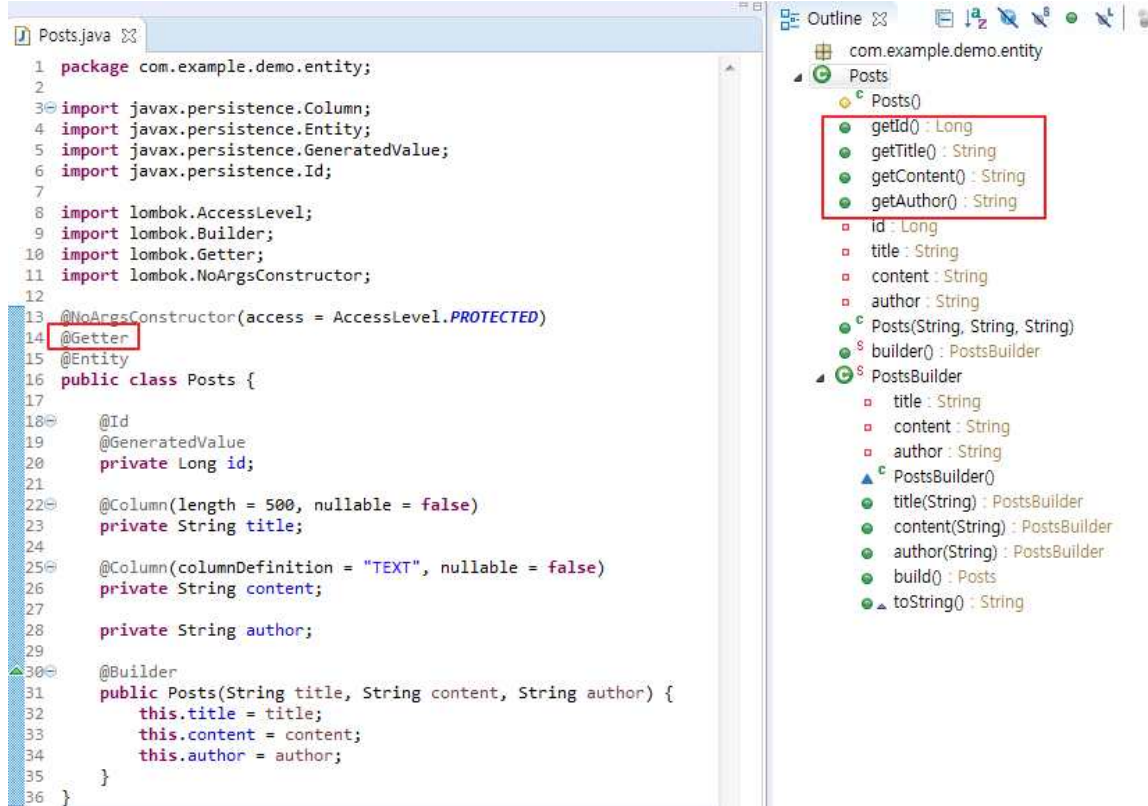
### (1) Lombok 설치

설치가이드 <https://medium.com/@dongchimi/이클립스에-lombok-설치-및-사용하기-b3489875780b>

- lombok.jar 파일 실행



## (2) Lombok 사용

Lombok features <https://projectlombok.org/features/all>

(가) @NoArgsConstructor

(나) @AllArgsConstructor

(다) @Data : getter, setter, toString

(라) @Builder

//생성자

new Example(b,a)

//빌더:어느 필드에 값을 넣을지 명확하기 인지

Example.builder().a(a).b(b).build();

## 마. postman

다운로드: <https://www.getpostman.com/downloads/>

설치파일: Postman-win64-7.2.0-Setup.exe