Spring Boot Application

스프링 부트 응용

스프링 부트와 스프링 데이터 JPA

한국기술교육대학교
온라인평생교육원

학습내용

- 스프링 데이터 JPA 이해
- 쿼리 메소드와 @Query 사용

학습목표

- 스프링 데이터 JPA를 이용하여 CRUD 기능을 처리할 수 있다.
- 쿼리 메소드를 사용하여 다양한 JPQL을 실행할 수 있다.

스프링 데이터 JPA 이해

1 스프링 데이터 JPA 적용

〈1〉 라이브러리 추가

- 새로운 스프링 부트 프로젝트 생성

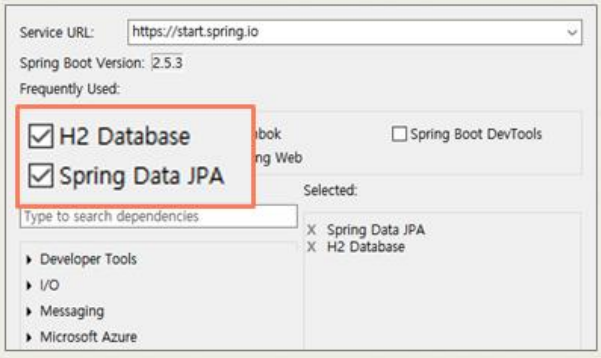
> • 스프링 데이터 JPA를 적용하기 위해서는
> 두 개의 라이브러리 의존성을 추가해야 함

1 스프링 데이터 JPA 적용

〈1〉 라이브러리 추가

- 새로운 스프링 부트 프로젝트 생성

Spring Data JPA	스프링 데이터 JPA 스타터
H2 Database	H2 데이터베이스 드라이버



스프링 데이터 JPA 이해

1 스프링 데이터 JPA 적용

2 JPA 설정

- 프로젝트에서 JPA 사용

> • application.yml 파일에 데이터 소스를 비롯한 JPA 설정을 추가해야 함

1 스프링 데이터 JPA 적용

2 JPA 설정

```yaml
# DataSource 설정
  datasource:
    driver-class-name: org.h2.Driver
    url: jdbc:h2:tcp://localhost/~/test
    username: sa
    password:

# JPA 설정
  jpa:
    database-platform: org.hibernate.dialect.H2Dialect
    show-sql: true
    generate-ddl: true
    hibernate:
      ddl-auto: update
    properties:
      hibernate:
        format_sql: true
```

스프링 데이터 JPA 이해

1 스프링 데이터 JPA 적용

〈2〉 JPA 설정

datasource, jpa 설정은 spring 하위에 위치해야 함

```
spring:
    datasource:
        driver-class-name: ...
        ...
        ...
    jpa:
        database-platform: ...
```

1 스프링 데이터 JPA 적용

〈3〉 JPA 구현체 로그 설정

- JPA 구현체인 Hibernate가 출력하는 로그 레벨을 DEBUG로 하면 많은 로그가 출력되어 로그 분석이 어려워짐
- 따라서 로그 레벨을 INFO로 지정함

```yaml
# Logging Level 설정
logging:
  level:
    org:
      hibernate: info
# 동일한 설정 : '[org.hibernate]': info
      springframework:
        security: debug
```

스프링 데이터 JPA 이해

1 스프링 데이터 JPA 적용

⟨4⟩ 엔티티 클래스 작성

- 게시판 테이블과 매핑할 Board 엔티티 클래스를 작성함
- 매핑과 관련된 JPA 어노테이션을 추가함

```
@Data
@Entity
public class Board {
    @Id
    @GeneratedValue
    private int seq;
    private String title;
    private String writer;
    private String content;
    @Temporal(value = TemporalType.TIMESTAMP)
    private Date createDate;
    private int cnt;
```

1 스프링 데이터 JPA 적용

⟨5⟩ 설정 확인

1. 메인 애플리케이션 (BoardWebApplication.java) 실행
2. 콘솔에 출력된 메시지를 확인해보면 BOARD 테이블이 생성되는 것을 확인할 수 있음

스프링 데이터 JPA 이해

1 스프링 데이터 JPA 적용

⑤ 설정 확인

```
Hibernate:

    drop table if exists board CASCADE
Hibernate:

    drop sequence if exists hibernate_sequence
Hibernate: create sequence hibernate_sequence start with 1 increment by 1
Hibernate:

    create table board (
       seq integer not null,
        cnt integer,
        content varchar(255),
        create_date timestamp,
        title varchar(255),
        writer varchar(255),
        primary key (seq)
    )
```

2 스프링 데이터 JPA

① Repository 인터페이스

- 전통적으로 데이터베이스 연동에서 작성했던 DAO(Data Access Object)와 동일한 개념의 인터페이스

스프링 데이터 JPA 이해

2 스프링 데이터 JPA

〈1〉 Repository 인터페이스

1. Repository 인터페이스를 상속하여 정의함
2. 스프링이 내부적으로 Repository 인터페이스에 대한 구현 클래스를 자동으로 생성해 줌

2 스프링 데이터 JPA

〈1〉 Repository 인터페이스

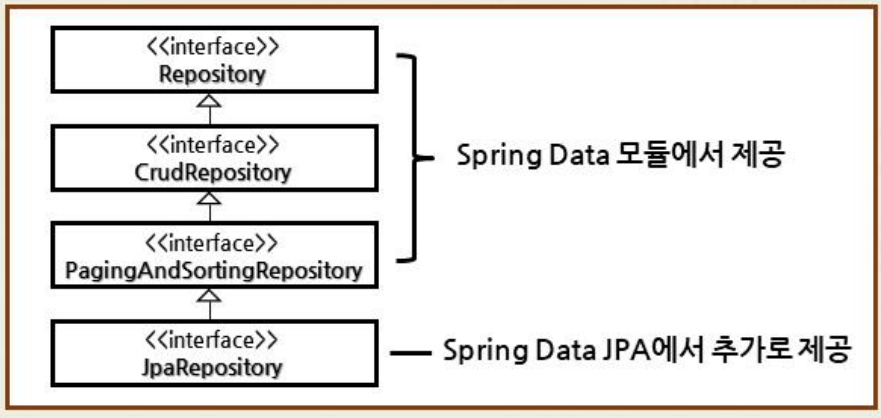
- 비즈니스 클래스에서는 Repository 객체를 이용하여 데이터베이스에 대한 CRUD 작업을 처리할 수 있음

스프링 데이터 JPA 이해

2 스프링 데이터 JPA

② Repository 인터페이스 종류

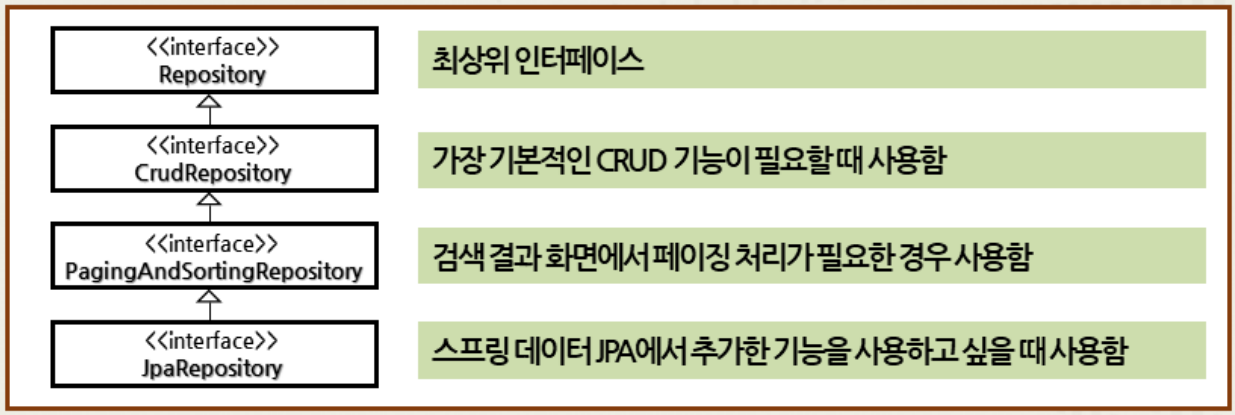
- 스프링 데이터 JPA는 Repository 인터페이스에서 파생된 다양한 Repository를 제공함



2 스프링 데이터 JPA

② Repository 인터페이스 종류

- 스프링 데이터 JPA는 Repository 인터페이스에서 파생된 다양한 Repository를 제공함



스프링 데이터 JPA 이해

2 스프링 데이터 JPA

〈3〉 Repository 인터페이스 작성

- Repository 인터페이스에는 T와 ID 두 개의 제네릭 타입을 지정해야 함

CrudRepository<T, ID>	
T	엔티티의 클래스 타입
ID	식별자 타입(@Id로 매핑한 식별자 변수의 타입)

2 스프링 데이터 JPA

〈3〉 Repository 인터페이스 작성

1. 개발자가 Repository 인터페이스 작성
2. 스프링이 구현 클래스를 제공
3. EntityManager 객체를 비롯한 JPA 연동에 필요한 객체들은 내부적으로 처리됨

스프링 데이터 JPA 이해

2 스프링 데이터 JPA

3 Repository 인터페이스 작성

```
package com.mycompany.persistence;

import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Repository;

import com.mycompany.entity.Board;

@Repository
public interface BoardRepository extends CrudRepository<Board, Integer> {

}
```

@Repository는 생략 가능함

2 스프링 데이터 JPA

4 CrudRepository 인터페이스 메소드

- CrudRepository 인터페이스가 제공하는 주요 메소드

메소드	설명
save(S entity)	Entity를 등록 또는 수정함(Save or Update)
delete(Object entity)	Entity를 삭제함
deleteById(Long id)	특정 id에 해당하는 엔티티를 삭제함
deleteAll()	모든 엔티티를 삭제함
findAll()	모든 엔티티를 검색함
findById(Long id)	특정 id에 해당하는 엔티티를 검색함
existsById(Long id)	특정 id에 해당하는 엔티티의 존재 여부를 확인함
count()	전체 엔티티의 개수를 리턴함

스프링 데이터 JPA 이해

2 스프링 데이터 JPA

5 Service 클래스 작성

- Service 클래스에서 Repository를 사용하기 위해서 의존성 주입(@Autowired)함
- 비즈니스 메소드는 Repository가 제공하는 메소드를 이용하여 비즈니스 로직을 처리할 수 있음

2 스프링 데이터 JPA

5 Service 클래스 작성

```
package com.mycompany.service;

import java.util.List;

@Service("boardService")
public class BoardService {

    @Autowired
    private BoardRepository boardRepositoy;

    public void insertBoard(Board board) {
        boardRepositoy.save(board);
    }

    public List<Board> getBoardList() {
        return (List<Board>) boardRepositoy.findAll();
    }
}
```

쿼리 메소드와 @Query 사용

1 쿼리 메소드

1 JPQL(Java Persistence Query Language)

- 애플리케이션에서는 다양한 조건의 복잡한 쿼리를 사용할 수 있어야 함

1 쿼리 메소드

1 JPQL(Java Persistence Query Language)

- JPA에는 복잡한 검색을 처리할 수 있도록 JPA 전용 쿼리 명령어인 JPQL(Java Persistence Query Language)을 지원함

쿼리 메소드와 @Query 사용

1 쿼리 메소드

〈1〉 JPQL(Java Persistence Query Language)

- JPQL은 검색 대상이 엔티티라는 것만 제외하면 기존의 SQL과 동일함

1 쿼리 메소드

〈1〉 JPQL(Java Persistence Query Language)

- 제목에 특정 키워드가 포함된 게시글을 검색하는 과정을 SQL과 JPQL로 비교

SQL	SELECT * FROM BOARD WHERE TITLE LIKE %검색키워드% ORDER BY SEQ DESC
JPQL	SELECT b FROM Board AS b WHERE b.title like %검색키워드% ORDER BY b.seq DESC

쿼리 메소드와 @Query 사용

1 쿼리 메소드

〈2〉 쿼리 메소드란?

- Repository 인터페이스에서 메소드 이름을 기반으로 JPQL을 생성하는 기능

1 쿼리 메소드

〈2〉 쿼리 메소드란?

- Repository 인터페이스에서 메소드 이름을 기반으로 JPQL을 생성하는 기능

find + 엔티티 이름 + By + 변수 이름

예 findBoardByTitle() : Board 엔티티에서 title 변수 값에 대한 검색 쿼리를 생성함

쿼리 메소드와 @Query 사용

1 쿼리 메소드

2 쿼리 메소드란?

- 쿼리 메소드에서 엔티티 이름을 생략하면 Repository 인터페이스에 선언된 엔티티 타입을 기준으로 자동으로 결정됨

1 쿼리 메소드

2 쿼리 메소드란?

```
public interface BoardRepository extends CrudRepository<Board, Integer> {

    findBoardByTitle();
    findByTitle();

}
```

동일한 결과

쿼리 메소드와 @Query 사용

1 쿼리 메소드

〈3〉 쿼리 메소드 유형

키워드	예	생성되는 JPQL
And	findByTitleAndContent	where b.title=?1 and b.content=?2
Or	findByTitleOrContent	where b.title=?1 or b.content=?2
Between	findByCreateDateBetween	where b.createDate between ?1 and ?2
LessThan LessThanEqual	findByCntLessThan findByCntLessThanEqual	where b.cnt < ?1 where b.cnt <= ?1
GreaterThan GreaterThanEqual	findByCntGreaterThan findByCntGreaterThanEqual	where b.cnt > ?1 where b.cnt >= ?1

1 쿼리 메소드

〈3〉 쿼리 메소드 유형

키워드	예	생성되는 JPQL
IsNull IsNotNull	findByWriterIsNull findByWriterIsNotNull	where b.writer is null where b.writer is not null
Not	findByCntNot	where b.cnt <> ?1
Containing	findByContentContaining	where b.content like '%'||?1||'%'
OrderBy	findByContentOrderBySeqDesc	where b.content like '%'||?1||'%' order by b.seq desc

쿼리 메소드와 @Query 사용

1 쿼리 메소드

〈4〉 페이징과 정렬 처리

- 쿼리 메소드

페이징 처리를 위해

- 파라미터로 Pageable 인터페이스를 받음

1 쿼리 메소드

〈4〉 페이징과 정렬 처리

1 페이징 처리

- 쿼리 메소드 작성

- 쿼리 메소드에서 Pageable 타입의 매개변수를 추가함

```
import org.springframework.data.domain.Pageable;

@Repository
public interface BoardRepository extends CrudRepository<Board, Integer> {

    List<Board> findByTitleContaining(String keyword, Pageable paging);
}
```

쿼리 메소드와 @Query 사용

1 쿼리 메소드

⟨4⟩ 페이징과 정렬 처리

1 페이징 처리

- 첫 페이지를 구성할 1번부터 다섯 개의 데이터를 조회하는 경우

```
// page : 0은 1페이지를 의미
// size : 검색할 데이터 수
Pageable paging = PageRequest.of(0, 5);
```

1 쿼리 메소드

⟨4⟩ 페이징과 정렬 처리

1 페이징 처리

- 페이징 SQL은 Dialect 클래스 설정에 따라 달라질 수 있음

쿼리 메소드와 @Query 사용

1 쿼리 메소드

⟨4⟩ 페이징과 정렬 처리

1 페이징 처리

- 페이징 처리 결과 SQL은 Dialect 클래스 설정에 따라 달라질 수 있음

▪ H2 데이터베이스의 결과

```
Hibernate:
    select
        board0_.seq as seq1_0_,
        board0_.cnt as cnt2_0_,
        board0_.content as content3_0_,
        board0_.create_date as create_d4_0_,
        board0_.title as title5_0_,
        board0_.writer as writer6_0_
    from
        board board0
    where
        board0_.title like ? escape ? limit ?
```

1 쿼리 메소드

⟨4⟩ 페이징과 정렬 처리

2 정렬 처리

- 데이터 정렬을 추가하려면 Sort 클래스를 이용하여 정렬 방식과 정렬할 변수를 지정하면 됨

```
// page : 0은 1페이지를 의미
// size : 검색할 데이터 수
Pageable paging = PageRequest.of(0, 5, Sort.Direction.DESC, "seq");
```

쿼리 메소드와 @Query 사용

1 쿼리 메소드

⟨4⟩ 페이징과 정렬 처리

2 정렬 처리

- 정렬이 추가됐을 때 생성된 쿼리

```
Hibernate:
    select
        board0_.seq as seq1_0_,
        board0_.cnt as cnt2_0_,
        board0_.content as content3_0_,
        board0_.create_date as create_d4_0_,
        board0_.title as title5_0_,
        board0_.writer as writer6_0_
    from
        board board0_
    where
        board0_.title like ? escape ?
    order by
        board0_.seq desc limit ?
```

1 쿼리 메소드

⟨5⟩ Page 타입 사용

1 Page 객체

- 검색 결과를 웹 페이지에서 사용하는 경우라면 List가 아닌 Page 객체를 리턴하는 것이 효율적임

쿼리 메소드와 @Query 사용

1 쿼리 메소드

⑤ Page 타입 사용

1 Page 객체

- Page는 검색 결과를 기반으로 웹 페이지 구성에 필요한 다양한 정보를 추가로 제공함

```
import org.springframework.data.domain.Page;

@Repository
public interface BoardRepository extends CrudRepository<Board, Integer> {

    Page<Board> findByTitleContaining(String keyword, Pageable paging);
}
```

1 쿼리 메소드

⑤ Page 타입 사용

2 Page 사용 결과

```
Hibernate:
    select
        board0_.seq as seq1_0_,
        board0_.cnt as cnt2_0_,
        board0_.content as content3_0_,
        board0_.create_date as create_d4_0_,
        board0_.title as title5_0_,
        board0_.writer as writer6_0_
    from
        board board0_
    where
        board0_.title like ? escape ?
    order by
        board0_.seq desc limit ?
Hibernate:
    select
        count(board0_.seq) as col_0_0_
    from
        board board0_
    where
        board0_.title like ? escape ?
PAGE SIZE : 5
TOTAL PAGES : 3
TOTAL COUNT : 11
NEXT : Page request [number: 1, size 5, sort: seq: DESC]
BOARD LIST : [Board(seq=11, title=Sprnig Data JPA 테스트, writer=테스터, content=Sp
```

쿼리 메소드와 @Query 사용

1 쿼리 메소드

```
Hibernate:
    select
        board0_.seq as seq1_0_,
        board0_.cnt as cnt2_0_,
        board0_.content as content3_0_,
        board0_.create_date as create_d4_0_,
        board0_.title as title5_0_,
        board0_.writer as writer6_0_
    from
        board board0_
    where
        board0_.title like ? escape ?
    order by
        board0_.seq desc limit ?
Hibernate:
    select
        count(board0_.seq) as col_0_0_
    from
        board board0_
    where
        board0_.title like ? escape ?
PAGE SIZE : 5
TOTAL PAGES : 3
TOTAL COUNT : 11
NEXT : Page request [number: 1, size 5, sort: seq: DESC]
BOARD LIST : [Board(seq=11, title=Sprnig Data JPA 테스트, writer=테스터, content=Sp
```

1차 검색
(실질적인 검색)

2차 검색
(데이터 수 검색)

1 쿼리 메소드

〈5〉 Page 타입 사용

3 Page에서 제공하는 메소드

메소드	기능
int getNumber()	현재 페이지 번호를 리턴함
int getSize()	한 페이지의 크기를 리턴함
int getTotalPages()	전체 페이지의 수를 리턴함
int getNumberOfElements()	결과 데이터 수를 리턴함
boolean hasPreviousPage()	이전 페이지의 존재 여부를 확인함
boolean hasNextPage()	다음 페이지의 존재 여부를 확인함

쿼리 메소드와 @Query 사용

1 쿼리 메소드

⑤ Page 타입 사용

3 Page에서 제공하는 메소드

메소드	기능
boolean isLastPage()	마지막 페이지 존재 여부를 확인함
Pageable nextPageable()	다음 페이지 객체를 리턴함
Pageable previeousPageable()	이전 페이지 객체를 리턴함
List<T> getContent()	실질적인 검색 결과 데이터 목록을 리턴함
boolean hasContent()	결과 존재 여부를 확인함
Sort getSort()	검색 시 사용된 Sort 정보를 리턴함

2 @Query 어노테이션

① @Query 어노테이션이란?

- 간단한 쿼리는 쿼리 메소드로 처리할 수 있지만 조인 같은 복잡한 쿼리는 JPQL을 사용해야 함

쿼리 메소드와 @Query 사용

2 @Query 어노테이션

〈1〉 @Query 어노테이션이란?

- 성능을 위해 데이터베이스 독립성을 포기하고 특정 데이터베이스에 종속적인 쿼리를 사용하는 경우도 있음

2 @Query 어노테이션

〈1〉 @Query 어노테이션이란?

- @Query는 직접 JPQL을 사용하거나 네이티브 쿼리를 사용할 때 사용하는 어노테이션

쿼리 메소드와 @Query 사용

2 @Query 어노테이션

⟨2⟩ 파라미터 설정

- @Query에 등록된 쿼리에 사용자가 입력한 파라미터를 설정하는 방법은 두 가지가 있음

2 @Query 어노테이션

⟨2⟩ 파라미터 설정

1 위치 기반 파라미터

```
@Query("SELECT b FROM Board b WHERE b.title like %?1% ORDER BY b.seq DESC")
List<Board> jpqlMethod(String keyword);
```

2 이름 기반 파라미터

```
@Query("SELECT b FROM Board b WHERE b.title like %:keyword% ORDER BY b.seq DESC")
List<Board> jpqlMethod(@Param("keyword") String keyword);
```

쿼리 메소드와 @Query 사용

2 @Query 어노테이션

〈3〉 네이티브 쿼리

- @Query를 통해 특정 데이터베이스에 종속적인 네이티브 쿼리를 사용할 수도 있음

2 @Query 어노테이션

〈3〉 네이티브 쿼리

- 특정 데이터베이스에 종속되는 문제가 있지만 성능상 최적화된 쿼리를 사용해야 하는 경우에는 네이티브 쿼리를 사용함

```
@Query(value="select seq, title, writer, createdate "
          + "from board where title like '%'||?1||'%' "
          + "order by seq desc", nativeQuery = true)
List<Object[]> nativeMethod(String keyword);
```

쿼리 메소드와 @Query 사용

2 @Query 어노테이션

〈3〉 네이티브 쿼리

주의

- @Query로 등록한 SQL에서 문제가 발생하면 애플리케이션 자체가 실행되지 않음
- @Query로 등록한 SQL은 애플리케이션이 실행되는 시점에 일괄적으로 메모리에 로딩되기 때문

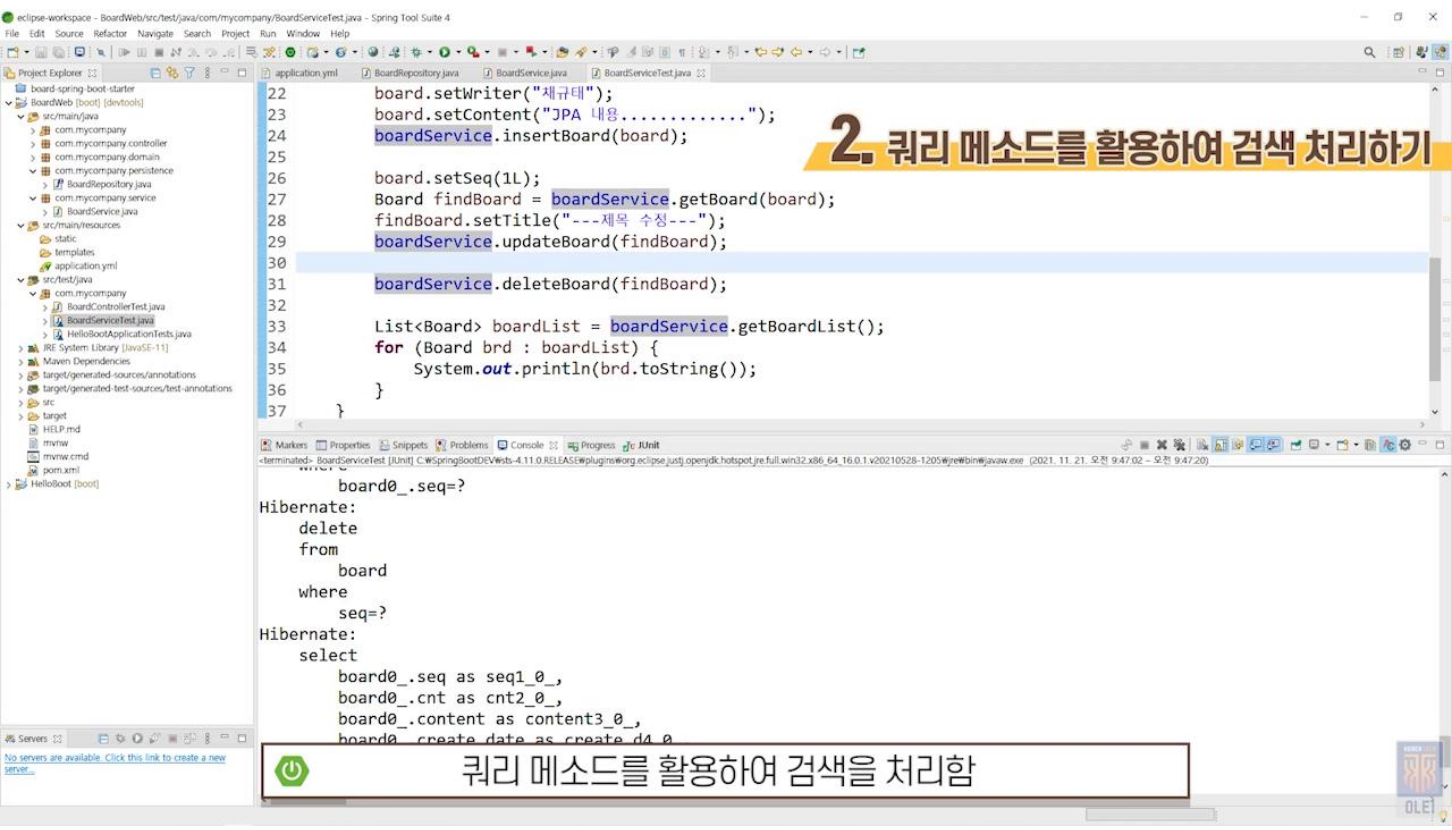
적용 스프링 부트

실습단계
CrudRepository 상속, CRUD 기능 JPA 기반으로 구현
쿼리 메소드를 기반으로 JPQL을 이용하여 목록 검색 기능 구현
특정 DBMS에 최적화 된 네이티브 쿼리 이용
더 이상 사용하지 않을 것이기 때문에 주석으로 처리함
yml 파일 수정
persistence.xml 파일을 작성할 필요가 없음
들여쓰기 주의하여 작성
persistence.xml 설정과 큰 차이 없음
인터페이스만 정의하면 됨
CRUD 기능을 포함하게 하려면 CrudRepository 상속
엔티티 클래스, 식별자 필드에 대한 타입 지정
인터페이스만 선언하면 구현 클래스는 자동으로 제공됨
의존성 주입

적용 스프링 부트

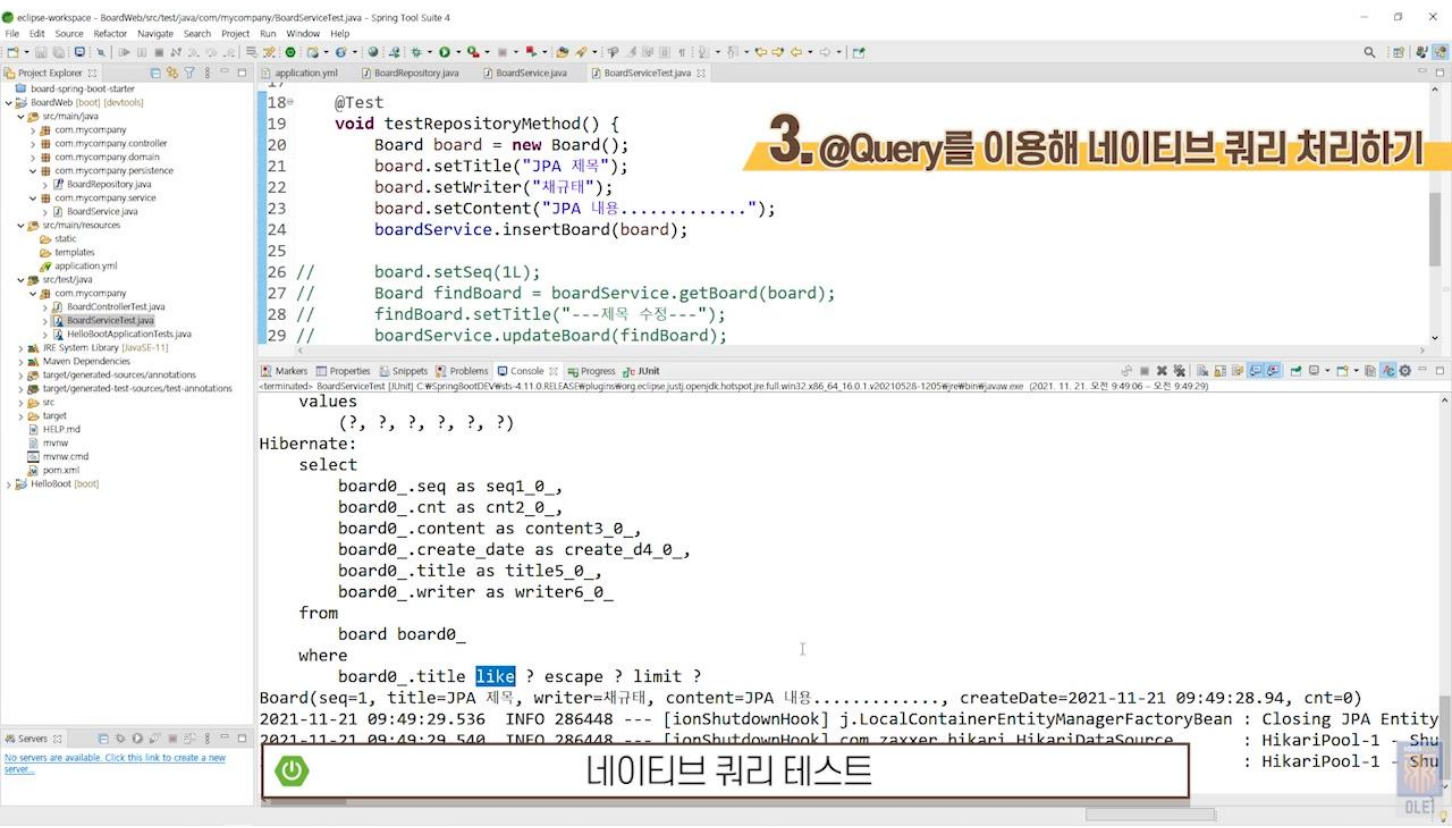
실습단계
save method가 insert, update 할 때 똑같이 사용됨
업데이트할 엔티티를 상세조회
모든 엔티티 셀렉트
의존성 주입
글등록
Insert 된 데이터를 상세조회
수정 가능
삭제 가능
H2 데이터베이스 구동 상태 확인
셀렉트 된 상세 정보를 업데이트
주석 막은 것을 해지하고 실행하면 delete sql도 실행됨

적용 스프링 부트



실습단계
쿼리 메소드 : 메소드 이름을 기준으로 JPQL을 제너레이션 해주는 기능
제목에 특정 키워드가 포함된 게시글을 like 검색
BoardService 클래스 수정
수정과 삭제는 주석으로 막고 테스트 진행
중요! Pagable 객체를 이용해 페이징 처리했기 때문에 실행된 쿼리에서도 관련 where 절 확인 가능

적용 스프링 부트



실습단계
네이티브 쿼리 테스트
중요! nativeQuery 속성이 true로 설정되어야 함
이름은 마음대로 결정해도 됨
검색 키워드는 빈 문자열을 세팅해 모든 글을 검색하도록 설정
페이징 관련 코드는 주석으로 막음
일반적인 SQL 실행된 것 확인 가능