Spring Boot 2 일차

면접

Spring Data JPA 에서 getOne 과 findById 차이점

https://granger.tistory.com/50

회사에[서 원하는것]

- boot 로 게시판
- Spring 으로 게시판
- react 로 게시판

Spring

Spring - MyBatisSpring - MyBatis - DB

Spring Boot

Spring - JPA - Hibernate(여러개가 있는데 요즘은 이것만 사용한다.)

ORM 페러다임 - JPA (서블릿 스펙)

EJB Entity beans

JPA 의 시초 -> HIbernate

dialect - DB 에 맞춰서 해석기를 돌려주는 역할 우리는 사용할 일 없음.

테이블 생성 전략

1. 자동 생성
2. 이미 테이블 존재 (현재 내가 사용하는 방식)
3. sql 파일 실행 (안전한 방법 - 실무에서 사용)
    - 프로젝트를 로딩하면 자동으로 이 SQL 을 실행.
    - sql 문을 미리 만들어 놓으면 촤라락 실행이 됨
    - 스프링 부트 SQL 로딩 검색 ㄱㄱ
    - data.sql 및 schema.sql 을 만들수 있음
    - https://recordsoflife.tistory.com/249?category=874063 -> (면접!!!!)
    - 위치는 application properties 랑 동일한 위치에 만든다.
    - 밖에서 부트를 사용한다고 하면 이 방식을 사용하니 꼭 알아두자.

스프링 프로필 설정

YMAL (검색 ㄱㄱ)

- 이것을 사용하면 application.properties 에 들여쓰기가 . 을 대신한다.
- 검색을 하다 보면 이런 형식으로 많이 나오는데 쫄지말고 그냥 properties 파일이랑 같은거라고 보면 됨

TodoRepository

AOP 에 Proxy 방식으로 만들어짐

JPA가 관리하는 녀석이 2녀석이 있다. - entity , value

DDD 검색해서 엔티티 찾기 - 도메인 주도 개발

엔티티의 가장 큰 특징은 식별자를 갖는것.

- 왜 식별자를 갖는가? 식별키가 있다는 얘기는 단독으로 crud를 한다는 의미이다.
- 클래스 하나일 수도 있고 여러개일 수도 있다.
- 엔티티는 pk 가 있다.
- ID 가 있으면 무조건 엔티티
- 그리고 그안에 벨튜들.

벨류타입은 엔티티의 일부인 애들

- 회원 - 주소 => 회원이 없으면 주소가 없음. 회원이 모든 주도권을 다 가지고 있음
- 영화게시물 - 이미지들 => 영화가 없으면 이미지들이 존재할 수 없음. 영화 이미지 단독으로 쓸일 없음.
- 그러면 이러한 **주소 와 이미지들은 벨류객체**, 값객체 라고 한다.
- 그리고 회원, 영화게시물은 비즈니스 단위로서 도메인이라고 한다. 즉 pk 이다. -> @Id

즉!!

- 엔티티가 있다면 반드시 @JPARepository 도 존재.
- 엔티티라는것은 하나의 단위이고 이 @JPARepository 이것들을 관리해주는 역할
- @JPARepository 이게 있으면 JPA 를 이용한다고 그리고 존재한다고 보면 되는것이다.

조회 작업 - 테스트

findById() 를 통해 조희를 한다.

save = insert, update

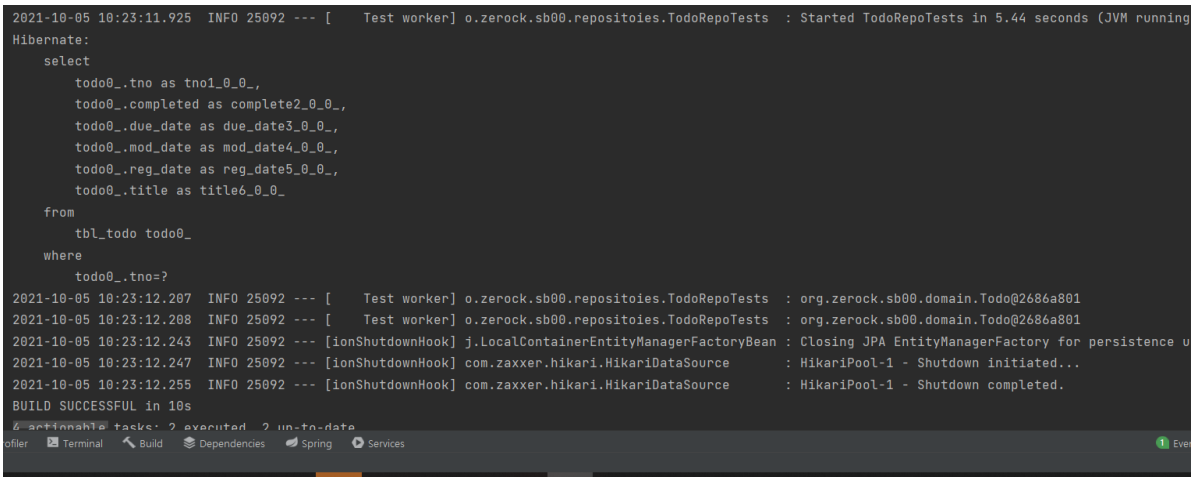
fetch = select

TodoRepoTests ㄱㄱ

```
@Test
```

```java
public void testRead() {
    Optional<Todo> result = todoRepository.findById(1L); // 거의 이 findById 만 사용한다. 이걸 사용하면 DB에서 데이터를 거ㅏ죠ㅓ온다

    result.ifPresent(todo -> log.info(todo)); // 만약에 존재한다면 이걸 해라

    if(result.isEmpty()) {
        // 만약 안에 없다면

        throw new RuntimeException("NOT FOUND");
        // 예외를 던져라
    }

    if(result.isPresent()) { // result.ifPresent(); 같은 내용
        log.info(result.get()); //  실제 내용물
    }
}
```

테스트 실행하면 결과가

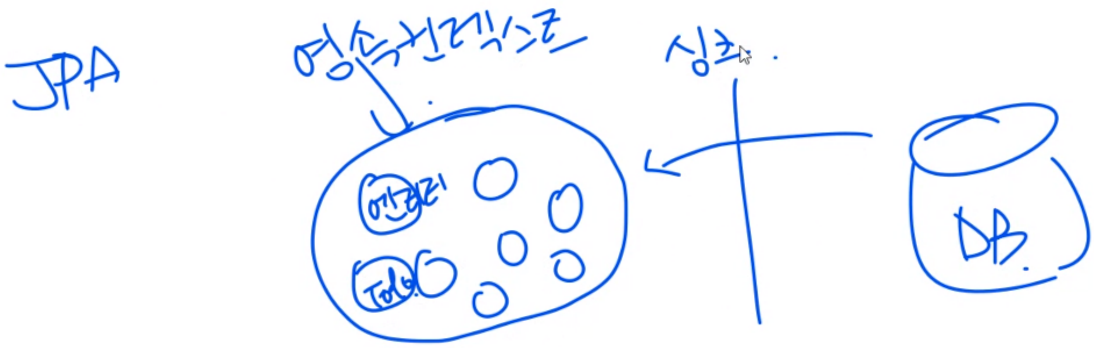


이리 나온다.

ORM 의 목적

나는 객체 지향만 코딩하겠다. 나는 영속영역은 신경 안쓰겠다.

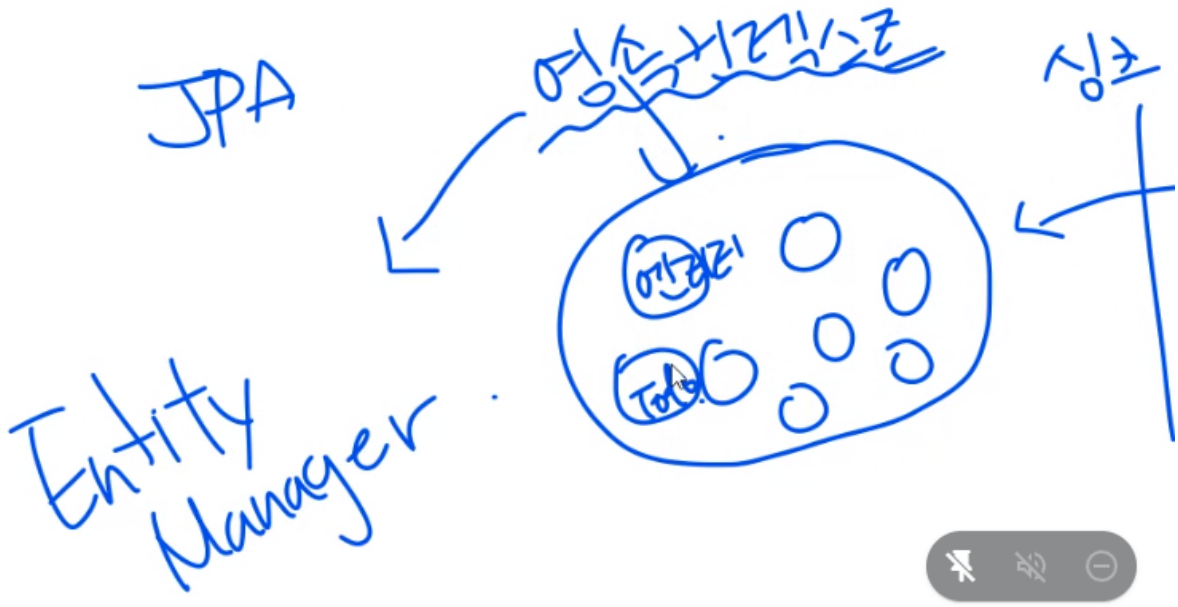
ORM 이 될라면



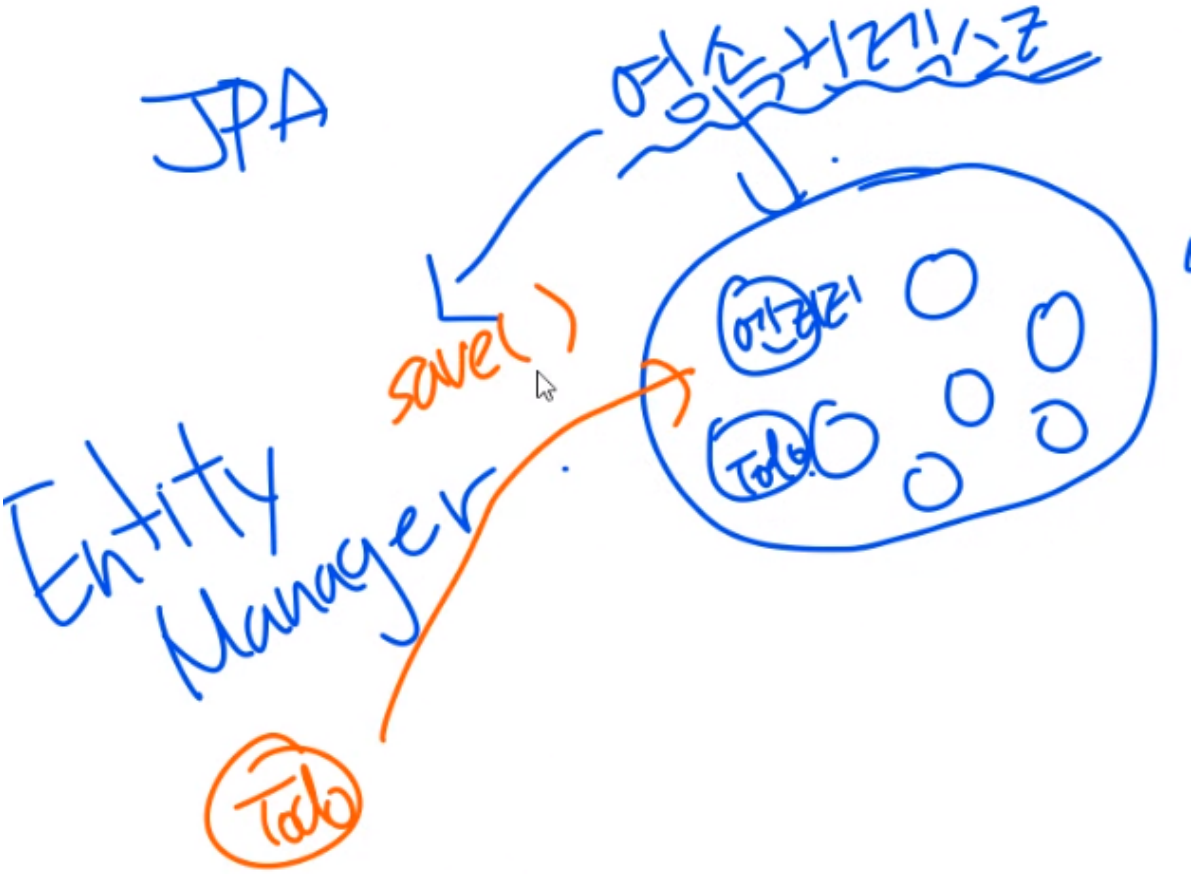
영속 컨텍스트는 DB를 미러링한다.

즉 영속 컨텍스트는 DB랑 싱크를 맞추려고 한다. ★★★

즉 DB 랑 저 영속컨텍스트를 똑같이 해주는게 ORM의 목적(?)



저 영속 컨텍스트를 관리하는 애들 - Entity Manager ★★★



이렇게 Todo를 save를 통해서 영속컨텍스트에 넣는다. 그럼 entity id 체크를 하고 만약 아이디가 똑같은 애가 존재하면 insert를 시키는게 아니라 update를 때려준다. 자동으로 insert에서 update로 변환을 해줌. ★★★

```
8344 --- [    Test worker] o.z.sb00.repositories.TodoRepoTests      : Started TodoRepoTests in 3.097 sec
8344 --- [    Test worker] o.z.sb00.repositories.TodoRepoTests      : ============================
8344 --- [    Test worker] o.z.sb00.repositories.TodoRepoTests      : ============================

'g.zerock.sb00.domain.Todo#1] - no Session
ionException: could not initialize proxy [org.zerock.sb00.domain.Todo#1] - no Session
(y.AbstractLazyInitializer.initialize(AbstractLazyInitializer.java:170)
```

저 no session 은 데이터 베이스랑 연결을 못했다는 뜻

getOne 은 지연 로딩

그래서 저 log.info 를 먼저 찍고 db 랑 연결을 시도한다.

필요할때까지 미뤘다가 필요한 순간에 select 하는 방식을 레이지 로딩 방식이라고 한다.

로딩의 방식

1. lazy 로딩 - 지연 로딩
2. eager - 즉시 로딩

수정 작업 - 테스트

domain - Todo ㄱㄱ

```
public void changeComplete() {
    this.completed = !completed; // true -> false / false -> true 로 바꿔줌
}
```

그다음

TodoRepoTests ㄱㄱ

```
@Test // (안전한 방식 UPDATE 는 그냥 이렇게 쓰면 된다!!!!)
    public void testUpdate() {
        Optional<Todo> result = todoRepository.findById(1L);
        // 일딴 todoRepository 에서 1번 게시글을 select 해서 가져온다

        if(result.isPresent()) {

            Todo todo = result.get();
            todo.changeComplete(); // 값 변경해주고
            log.info(todo);
            todoRepository.save(todo);
            // update 된 내용이 사실인지 Hibernate: 에서 select 해서 다시한번 확인해주
고 그다음 save 로 update 진행
        }

    }
```

그다음 테스트 실행

```
2021-10-05 10:54:47.792  INFO 24688 --- [    Test worker] o
Hibernate:
    select
        todo0_.tno as tno1_0_0_,
        todo0_.completed as complete2_0_0_,
        todo0_.due_date as due_date3_0_0_,
        todo0_.mod_date as mod_date4_0_0_,
        todo0_.reg_date as reg_date5_0_0_,
        todo0_.title as title6_0_0_
    from
        tbl_todo todo0_
    where
        todo0_.tno=?
Hibernate:
    update
        tbl_todo
    set
        completed=?,
        due_date=?,
        mod_date=?,
        reg_date=?,
        title=?
    where
        tno=?
2021-10-05 10:54:47.891  INFO 24688 --- [ionShutdownHook] j
```

	tno	completed	due_date	mo
1	1	true	2021-10-08	2021

값이 true 로 바뀜

삭제 작업 - 테스트

TodoRepoTests ㄱㄱ

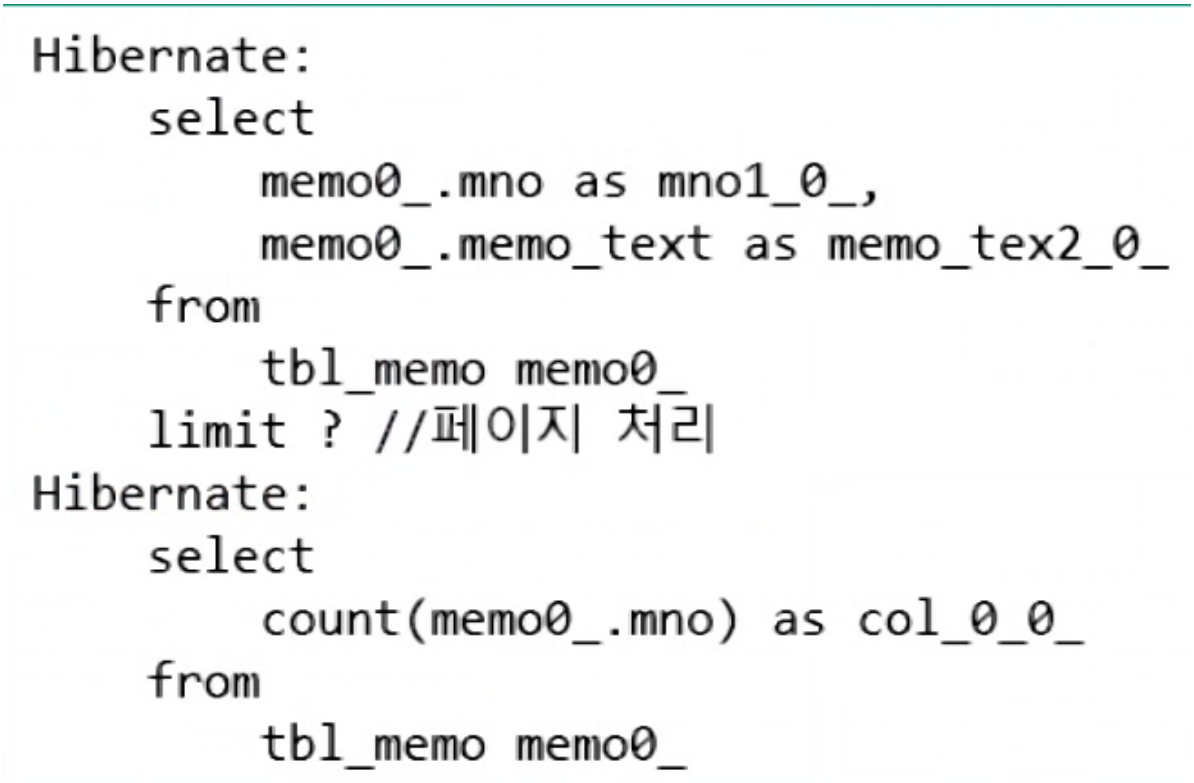
```
@Test
public void testDelete() {

    todoRepository.deleteById(1L);

}
```

페이징 작업

PageRequest.of( )

```
@Test
public void testPageDefault() {

    //1페이지 10개
    Pageable pageable = PageRequest.of(0,10);

    Page<Memo> result = memoRepository.findAll(pageable);

    System.out.println(result);

}
```

테스트 실행 결과



페이징 처리 자동

count 자동으로 select 된다.

페이징 처리 실습!!

TodoRepoTests ㄱㄱ

tbl_Todo 테이블 지우자

어차피 나중에 자동으로 생성

그다음 아래처럼 바꿔주자

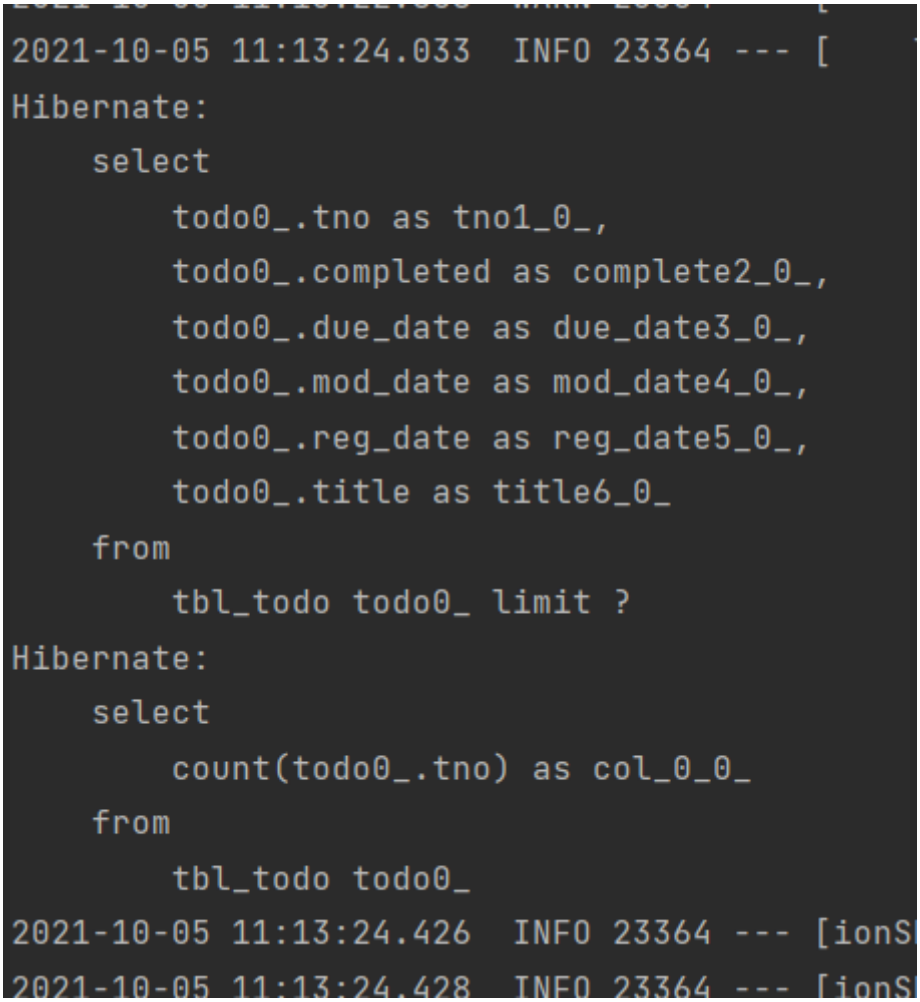
```
@Test
public void testInsert() {

    IntStream.rangeClosed(1, 300).forEach(i -> {
        Todo todo = Todo.builder()
                .title("Todo...." + i)
                .dueDate(LocalDate.now().plusDays(i % 7))
                .build();

        todoRepository.save(todo);

    });


}
```

더미데이터 이렇게 넣어주자

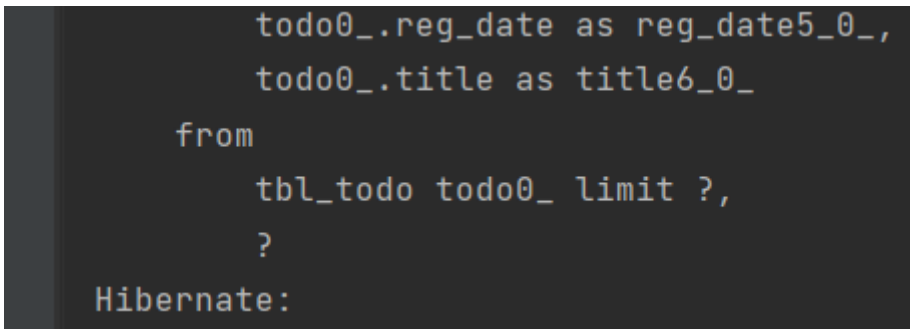
그다음

```
@Test
public void testPaging() {

    Pageable pageable = PageRequest.of(0, 10);

    //  pageable 을 파라미터로 사용하면 리턴타입은 항상 Page 고정된다.

    Page<Todo> result = todoRepository.findAll(pageable);

}
```

테스트 실행!!




```
Pageable pageable = PageRequest.of(1, 10); // 1로 바꾸면
```



limit 가 2개 생긴다.

```
System.out.println("--------------------------------------");

System.out.println("Total Pages: "+result.getTotalPages()); //총 몇 페이지

System.out.println("Total Count: "+result.getTotalElements()); //전체 개수

System.out.println("Page Number: "+result.getNumber()); //현재 페이지 번호 0부터 시

System.out.println("Page Size: "+result.getSize()); //페이지당 데이터 개수

System.out.println("has next page?: "+result.hasNext()); //다음 페이지 존재 여부

System.out.println("first page?: "+result.isFirst()); //시작 페이지(0) 여부
```

PageMaker 랑 너무다고 비슷하다 위에 저 result.기능 들이.

정렬 조건 추가하기 (order by)

db 에서 order by 부분을 추가하고 싶은 것이다.

```
@Test
public void testPaging() {

    Pageable pageable = PageRequest.of(1, 10, Sort.by("tno").descending()); // 이
부분 수정

    //  pageable 을 파라미터로 사용하면 리턴타입은 항상 Page 고정된다.

    Page<Todo> result = todoRepository.findAll(pageable);

}
```

실행

```
Hibernate:
    select
        todo0_.tno as tno1_0_,
        todo0_.completed as complete2_0
        todo0_.due_date as due_date3_0_
        todo0_.mod_date as mod_date4_0_
        todo0_.reg_date as reg_date5_0_
        todo0_.title as title6_0_
    from
        tbl_todo todo0_
    order by
        todo0_.tno desc limit ?,
        ?
Hibernate:
    select
```

그다음

아래처럼 수정

```
@Test
public void testPaging() {

    Pageable pageable = PageRequest.of(1, 10, Sort.by("tno").descending());

    //  pageable 을 파라미터로 사용하면 리턴타입은 항상 Page 고정된다.

    Page<Todo> result = todoRepository.findAll(pageable);

    result.get().forEach(todo -> log.info(todo)); // 이렇게 하면 닥 10개만  // 추가

}
```

```
    from
        tbl_todo todo0_
    order by
        todo0_.tno desc limit ?,
        ?
Hibernate:
    select
        count(todo0_.tno) as col_0_0_
    from
        tbl_todo todo0_
2021-10-05 11:20:31.293  INFO 5040 --- [    Test worker] o.zerock.sb00.repositoies.TodoRepoTests  : org.zerock.sb00.domain.Todo@52b7c1d9
2021-10-05 11:20:31.294  INFO 5040 --- [    Test worker] o.zerock.sb00.repositoies.TodoRepoTests  : org.zerock.sb00.domain.Todo@6f986031
2021-10-05 11:20:31.294  INFO 5040 --- [    Test worker] o.zerock.sb00.repositoies.TodoRepoTests  : org.zerock.sb00.domain.Todo@38326fd5
2021-10-05 11:20:31.294  INFO 5040 --- [    Test worker] o.zerock.sb00.repositoies.TodoRepoTests  : org.zerock.sb00.domain.Todo@4a467ada
2021-10-05 11:20:31.294  INFO 5040 --- [    Test worker] o.zerock.sb00.repositoies.TodoRepoTests  : org.zerock.sb00.domain.Todo@92775a3
2021-10-05 11:20:31.294  INFO 5040 --- [    Test worker] o.zerock.sb00.repositoies.TodoRepoTests  : org.zerock.sb00.domain.Todo@2512155e
2021-10-05 11:20:31.294  INFO 5040 --- [    Test worker] o.zerock.sb00.repositoies.TodoRepoTests  : org.zerock.sb00.domain.Todo@ef59aee
2021-10-05 11:20:31.294  INFO 5040 --- [    Test worker] o.zerock.sb00.repositoies.TodoRepoTests  : org.zerock.sb00.domain.Todo@779cb140
2021-10-05 11:20:31.295  INFO 5040 --- [    Test worker] o.zerock.sb00.repositoies.TodoRepoTests  : org.zerock.sb00.domain.Todo@45a4587d
2021-10-05 11:20:31.295  INFO 5040 --- [    Test worker] o.zerock.sb00.repositoies.TodoRepoTests  : org.zerock.sb00.domain.Todo@70a91d72
2021-10-05 11:20:31.317  INFO 5040 --- [ionShutdownHook] j.LocalContainerEntityManagerFactoryBean : Closing JPA EntityManagerFactory for pers
2021-10-05 11:20:31.320  INFO 5040 --- [ionShutdownHook] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Shutdown initiated...
2021-10-05 11:20:31.336  INFO 5040 --- [ionShutdownHook] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Shutdown completed.
BUILD SUCCESSFUL in 10s
```

이렇게 뜬다 이것을 고칠려면

Todo ㄱㄱ

@ToString 추가

그다음 다시 실행

제대로 뜸

검색 기능 - 하지만 이기능은 많이 사용 안한다고 한다

하지만 댓글 리스트 불러올때는 쓸만하다. 즉 쿼리문이 변경될 가능성이 없는 곳에서는 쓸모가 있다.

하지만 쿼리문이 변경될 곳에서는 쓸모가 없다.

그래서 이 기능 보다는 @Query 를 더 많이 사용한다고 한다.

TodoRepository ㄱㄱ

```
public interface TodoRepository extends JpaRepository<Todo, Long> {

    List<Todo> findTodoByTitleContains(String keyword);
}
```

TodoRepoTests ㄱㄱ

```
@Test
public void testQueryMethod() {
    List<Todo> todos = todoRepository.findTodoByTitleContains("10");

    todos.forEach(todo -> log.info(todo));
}
```

```
toies.TodoRepoTests  : Todo(tno=10, title=Todo....10, dueDate=2021-10-08, completed=false, regDate=2021-10-05T11:09:50.908706, modDate=2021-10-05T11:
toies.TodoRepoTests  : Todo(tno=100, title=Todo....100, dueDate=2021-10-07, completed=false, regDate=2021-10-05T11:09:51.356508, modDate=2021-10-05T1
toies.TodoRepoTests  : Todo(tno=101, title=Todo....101, dueDate=2021-10-08, completed=false, regDate=2021-10-05T11:09:51.361496, modDate=2021-10-05T1
toies.TodoRepoTests  : Todo(tno=102, title=Todo....102, dueDate=2021-10-09, completed=false, regDate=2021-10-05T11:09:51.365485, modDate=2021-10-05T1
toies.TodoRepoTests  : Todo(tno=103, title=Todo....103, dueDate=2021-10-10, completed=false, regDate=2021-10-05T11:09:51.368476, modDate=2021-10-05T1
toies.TodoRepoTests  : Todo(tno=104, title=Todo....104, dueDate=2021-10-11, completed=false, regDate=2021-10-05T11:09:51.372465, modDate=2021-10-05T1
toies.TodoRepoTests  : Todo(tno=105, title=Todo....105, dueDate=2021-10-05, completed=false, regDate=2021-10-05T11:09:51.376455, modDate=2021-10-05T1
toies.TodoRepoTests  : Todo(tno=106, title=Todo....106, dueDate=2021-10-06, completed=false, regDate=2021-10-05T11:09:51.379448, modDate=2021-10-05T1
toies.TodoRepoTests  : Todo(tno=107, title=Todo....107, dueDate=2021-10-07, completed=false, regDate=2021-10-05T11:09:51.383437, modDate=2021-10-05T1
toies.TodoRepoTests  : Todo(tno=108, title=Todo....108, dueDate=2021-10-08, completed=false, regDate=2021-10-05T11:09:51.387425, modDate=2021-10-05T1
toies.TodoRepoTests  : Todo(tno=109, title=Todo....109, dueDate=2021-10-09, completed=false, regDate=2021-10-05T11:09:51.391414, modDate=2021-10-05T1
toies.TodoRepoTests  : Todo(tno=110, title=Todo....110, dueDate=2021-10-10, completed=false, regDate=2021-10-05T11:09:51.395404, modDate=2021-10-05T1
toies.TodoRepoTests  : Todo(tno=210, title=Todo....210, dueDate=2021-10-05, completed=false, regDate=2021-10-05T11:09:51.747462, modDate=2021-10-05T1
tyManagerFactoryBean : Closing JPA EntityManagerFactory for persistence unit 'default'
kariDataSource       : HikariPool-1 - Shutdown initiated...
```

그럼 title에 10이 들어가있는 애들만 싹 조회되어 나옴

페이징 처리한 검색 기능 - 하지만 이기능은 많이 사용 안한다고 한다.

TodoRepository ㄱㄱ

```
public interface TodoRepository extends JpaRepository<Todo, Long> {

    List<Todo> findTodoByTitleContains(String keyword);

    Page<Todo> findTodoByTitleContains(String keyword, Pageable pageable);
}
```

그다음 테스트 ㄱㄱ

```
@Test
public void testQueryMethodPaging() {

    Pageable pageable = PageRequest.of(0, 10, Sort.by("tno").descending()); // 역순 정렬

    Page<Todo> result = todoRepository.findTodoByTitleContains("10", pageable);

    log.info("total" + result.getTotalElements());
    log.info("end page" + result.getTotalPages());
    result.get().forEach(todo -> log.info(todo));
}
```

테스트 실행

```
ests  : total13
ests  : end page2
ests  : Todo(tno=210, title=Todo....210, dueDate=2021-10-05, completed=false, regDate=2021-10-05T11:09:51.747462, modDate=2021-10-05T11:09:51.747462)
ests  : Todo(tno=110, title=Todo....110, dueDate=2021-10-10, completed=false, regDate=2021-10-05T11:09:51.395404, modDate=2021-10-05T11:09:51.395404)
ests  : Todo(tno=109, title=Todo....109, dueDate=2021-10-09, completed=false, regDate=2021-10-05T11:09:51.391414, modDate=2021-10-05T11:09:51.391414)
ests  : Todo(tno=108, title=Todo....108, dueDate=2021-10-08, completed=false, regDate=2021-10-05T11:09:51.387425, modDate=2021-10-05T11:09:51.387425)
ests  : Todo(tno=107, title=Todo....107, dueDate=2021-10-07, completed=false, regDate=2021-10-05T11:09:51.383437, modDate=2021-10-05T11:09:51.383437)
ests  : Todo(tno=106, title=Todo....106, dueDate=2021-10-06, completed=false, regDate=2021-10-05T11:09:51.379448, modDate=2021-10-05T11:09:51.379448)
ests  : Todo(tno=105, title=Todo....105, dueDate=2021-10-05, completed=false, regDate=2021-10-05T11:09:51.376455, modDate=2021-10-05T11:09:51.376455)
ests  : Todo(tno=104, title=Todo....104, dueDate=2021-10-11, completed=false, regDate=2021-10-05T11:09:51.372465, modDate=2021-10-05T11:09:51.372465)
ests  : Todo(tno=103, title=Todo....103, dueDate=2021-10-10, completed=false, regDate=2021-10-05T11:09:51.368476, modDate=2021-10-05T11:09:51.368476)
ests  : Todo(tno=102, title=Todo....102, dueDate=2021-10-09, completed=false, regDate=2021-10-05T11:09:51.365485, modDate=2021-10-05T11:09:51.365485)
yBean : Closing JPA EntityManagerFactory for persistence unit 'default'
      : HikariPool-1 - Shutdown initiated...
      : HikariPool-1 - Shutdown completed.
```

@Query

얘를 쓰는 경우가 더 많다!!

더 간결하게 처리한다는 장점

쿼리 메소드는 단위가 엔티티인데 @Query 이것을 사용하면 여러가지 다 사용 가능

TodoRepository ㄱㄱ

```
public interface TodoRepository extends JpaRepository<Todo, Long> {

    List<Todo> findTodoByTitleContains(String keyword);

    //JPQL - JPA의 쿼리 언어이다. -> 엔티티를 기준으로 쿼리를 작성
    @Query("select t from  Todo t where t.title like concat('%', : keyword, '%')
") // 추가
    List<Todo> getListTitle(String keyword);

    Page<Todo> findTodoByTitleContains(String keyword, Pageable pageable);
}
```

일딴은 테스트 안한다

나중에 예제 만들때 사용할꺼라 나중에 하면 된다.

```
@Transactional
@Modifying
@Query("update Memo m set m.memoText = :memoText where m.mno = :mno ")
int updateMemoText(@Param("mno") Long mno, @Param("memoText") String memoText );
```

@Param 을 사용한다.

```
@Query(value = "select m from Memo m where m.mno > :mno",
      countQuery = "select count(m) from Memo m where m.mno > :mno" )
Page<Memo> getListWithQuery(Long mno, Pageable pageable);
```

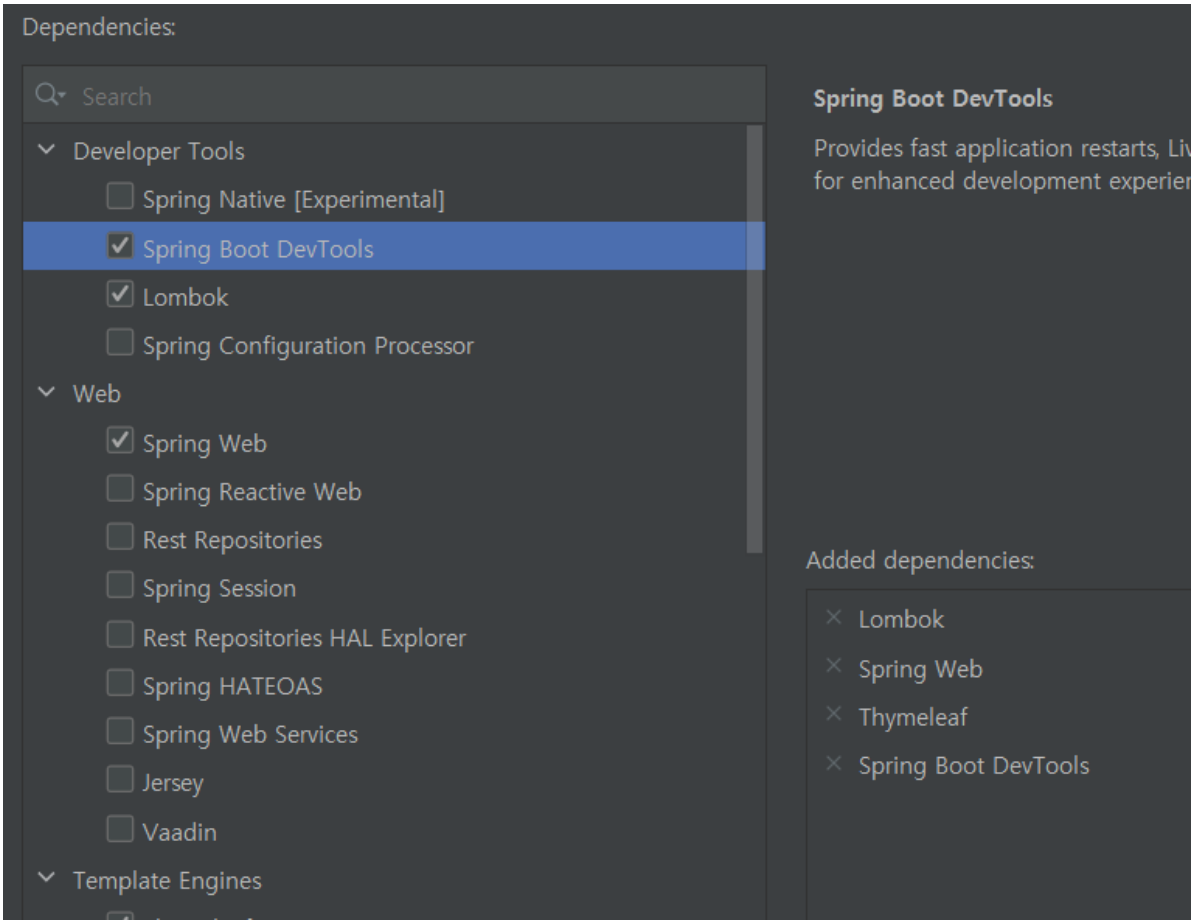
뒤에다가 저렇게 pageable 페이징 처리만 주면 자동으로 다 페이징 처리가 된다.

게시판 만들기 with Spring Boot!!!!!!!!!!

association?

★★ Thymeleaf 를 사용!! ★★

프로젝트 새로 생성



이렇게 설정

controller 패키지 생성

SampleController 생성

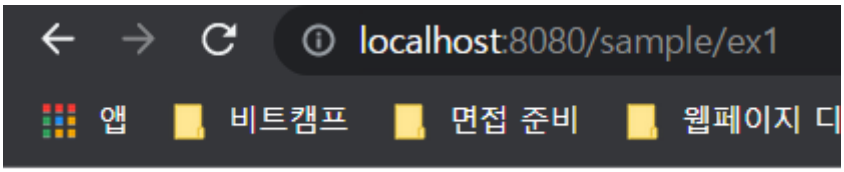
```
@Controller
@RequestMapping("/sample")
@Log4j2
public class SampleController {

    @GetMapping("/ex1")
    public void ex1() {
        log.info("ex1.....................");
    }
}
```

그다음

resources - templates - sample - ex01.html 생성 ㄱㄱ

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<h1>ex1</h1>
</body>
</html>
```

여기서는 webapp 이 아닌 resources 에서 페이지를 다룬다.



그다음 맨위에

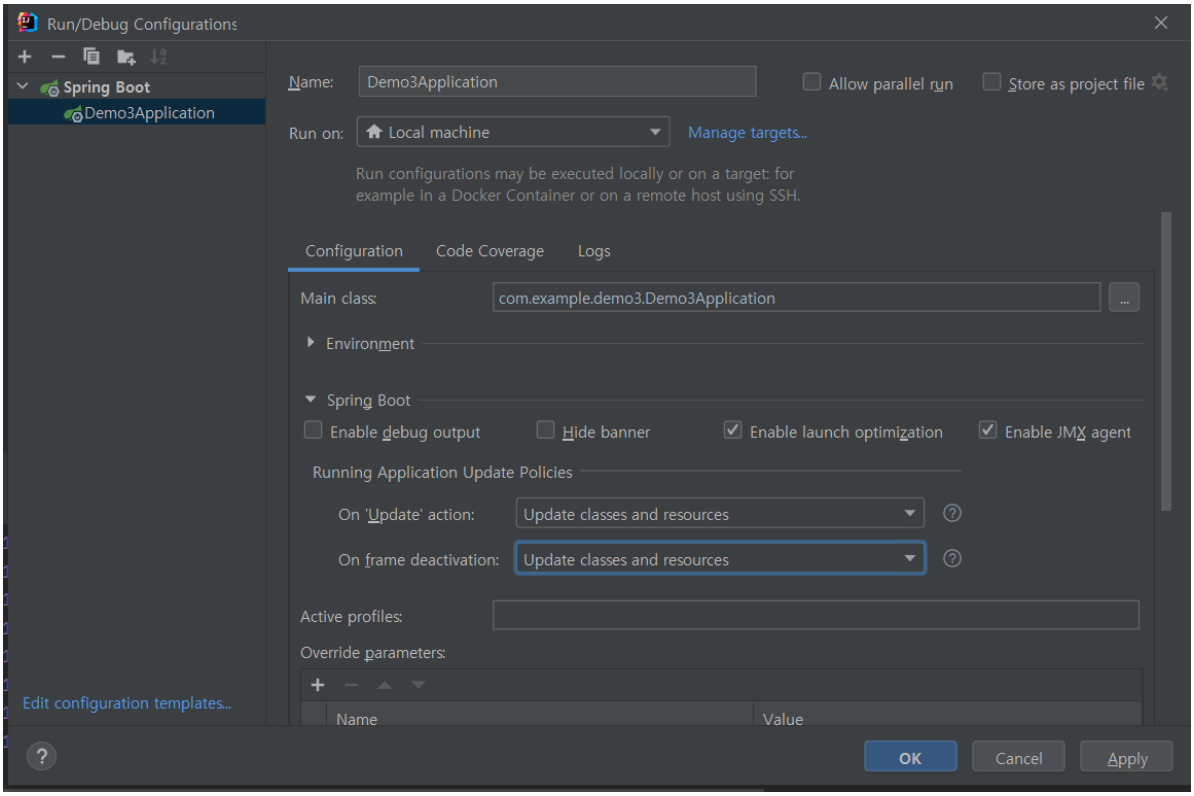
```
<html xmlns:th="http://www.thymeleaf.org">
```

이거 추가

application.properties ㄱㄱ

```
spring.thymeleaf.cache=false
```

추가

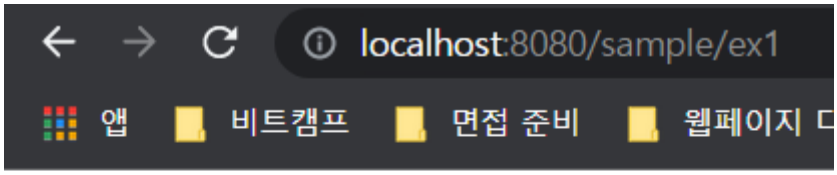


자동 tomcat 실행같은 느낌?

<th:> 사용해보기

방법 1

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<h1>ex1</h1>
<h2 th:text="${'Hello1'}"></h2>
</body>
</html>
```

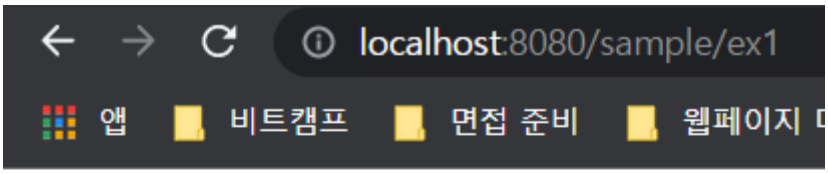


ex1

Hello1

방법2

```
</head>
<body>
<h1>ex1</h1>
<h2 th:text="${'Hello1'}"></h2>
<h2>[[${'Hello2'}]]</h2>
</body>
</html>
```



ex1

Hello1

Hello2

---

th 로 반복문 쓰기

dto package 만들기

SampleDTO ㄱㄱ

```
@Builder
@AllArgsConstructor
@NoArgsConstructor
@Data
public class SampleDTO {

    private Long sno;
    private String title;
    private LocalDateTime regdate;
}
```

SampleController ㄱㄱ

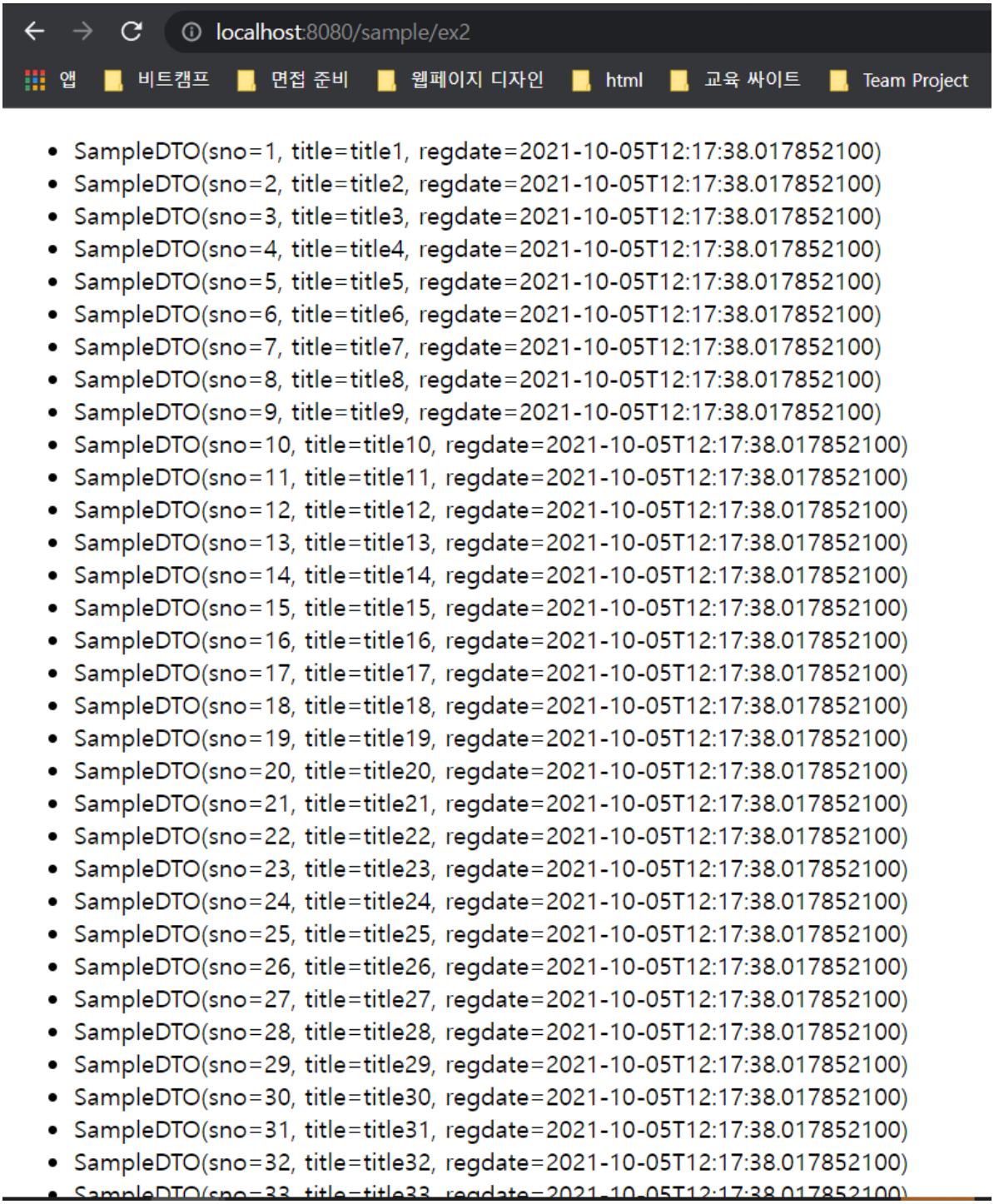
```
@GetMapping("/ex2")
public void ex1(Model model) {
    log.info("ex2.....................");

    // mapToObj() 는 int 를 obj 객체로 바꿔줌
    List<SampleDTO> dtoList =
    IntStream.rangeClosed(1, 100)
            .mapToObj(i -> SampleDTO.builder().sno((long)i).title("title" +
i).regdate(LocalDateTime.now()).build())
                    .collect(Collectors.toList());

    model.addAttribute("list", dtoList);
}
```

resources - ex2.html 생성

thymleaf 는 li 태그 자체를 반복시킴

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>

<ul>
    <li th:each="dto: ${list}">
        [[${dto}]]
    </li>
</ul>

</body>
</html>
```

새로고침 ㄱㄱ



이렇게 출력된다.

status 반복 상태 변수 - 1. 게시글, 2. 게시글 ...

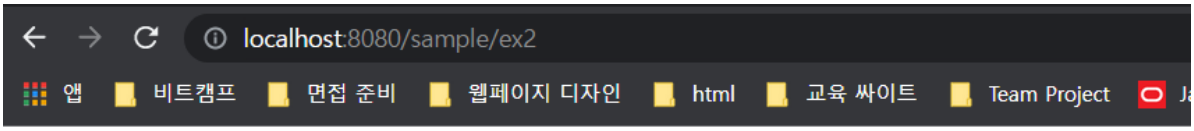
반복 상태 변수 (status)

Thymeleaf에서 th:each 사용하면 반복 상태를 추적할 수 있는 status 변수를 제공해 주는데 이를 이용하여 index, count 등의 값을 쉽게 추출 할 수 있습니다.

```
index    현재 반복 인덱스 (0부터 시작)
count    현재 반복 인덱스 (1부터 시작)
size     총 요소 수
current  현재 요소
even     현재 반복이 짝수인지 여부 (boolean)
odd      현재 반복이 홀수인지 여부 (boolean)
first    현재 반복이 첫번째인지 여부 (boolean)
last     현재 반복이 마지막인지 여부 (boolean)
```

https://ifuwanna.tistory.com/200

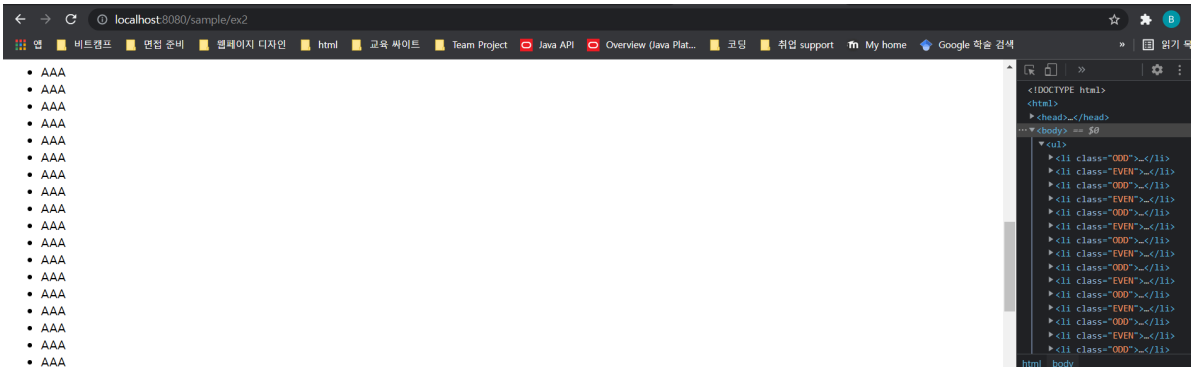
```
<body>

<ul>
    <li th:each="dto, status: ${list}">
        [[${status.index}]]--- [[${dto}]]
    </li>
</ul>

</body>
```



- 0--- SampleDTO(sno=1, title=title1, regdate=2021-10-05T12:20:45.118218400)
- 1--- SampleDTO(sno=2, title=title2, regdate=2021-10-05T12:20:45.118218400)
- 2--- SampleDTO(sno=3, title=title3, regdate=2021-10-05T12:20:45.118218400)
- 3--- SampleDTO(sno=4, title=title4, regdate=2021-10-05T12:20:45.118218400)
- 4--- SampleDTO(sno=5, title=title5, regdate=2021-10-05T12:20:45.118218400)
- 5--- SampleDTO(sno=6, title=title6, regdate=2021-10-05T12:20:45.118218400)
- 6--- SampleDTO(sno=7, title=title7, regdate=2021-10-05T12:20:45.118218400)
- 7--- SampleDTO(sno=8, title=title8, regdate=2021-10-05T12:20:45.118218400)
- 8--- SampleDTO(sno=9, title=title9, regdate=2021-10-05T12:20:45.118218400)
- 9--- SampleDTO(sno=10, title=title10, regdate=2021-10-05T12:20:45.118218400)
- 10--- SampleDTO(sno=11, title=title11, regdate=2021-10-05T12:20:45.118218400)
- 11--- SampleDTO(sno=12, title=title12, regdate=2021-10-05T12:20:45.118218400)
- 12--- SampleDTO(sno=13, title=title13, regdate=2021-10-05T12:20:45.118218400)
- 13--- SampleDTO(sno=14, title=title14, regdate=2021-10-05T12:20:45.118218400)
- 14--- SampleDTO(sno=15, title=title15, regdate=2021-10-05T12:20:45.118218400)
- 15--- SampleDTO(sno=16, title=title16, regdate=2021-10-05T12:20:45.118218400)
- 16--- SampleDTO(sno=17, title=title17, regdate=2021-10-05T12:20:45.118218400)
- 17--- SampleDTO(sno=18, title=title18, regdate=2021-10-05T12:20:45.118218400)
- 18--- SampleDTO(sno=19, title=title19, regdate=2021-10-05T12:20:45.118218400)
- 19--- SampleDTO(sno=20, title=title20, regdate=2021-10-05T12:20:45.118218400)
- 20--- SampleDTO(sno=21, title=title21, regdate=2021-10-05T12:20:45.118218400)
- 21--- SampleDTO(sno=22, title=title22, regdate=2021-10-05T12:20:45.118218400)
- 22--- SampleDTO(sno=23, title=title23, regdate=2021-10-05T12:20:45.118218400)
- 23--- SampleDTO(sno=24, title=title24, regdate=2021-10-05T12:20:45.118218400)
- 24--- SampleDTO(sno=25, title=title25, regdate=2021-10-05T12:20:45.118218400)
- 25--- SampleDTO(sno=26, title=title26, regdate=2021-10-05T12:20:45.118218400)
- 26--- SampleDTO(sno=27, title=title27, regdate=2021-10-05T12:20:45.118218400)
- 27--- SampleDTO(sno=28, title=title28, regdate=2021-10-05T12:20:45.118218400)
- 28--- SampleDTO(sno=29, title=title29, regdate=2021-10-05T12:20:45.118218400)
- 29--- SampleDTO(sno=30, title=title30, regdate=2021-10-05T12:20:45.118218400)
- 30--- SampleDTO(sno=31, title=title31, regdate=2021-10-05T12:20:45.118218400)
- 31--- SampleDTO(sno=32, title=title32, regdate=2021-10-05T12:20:45.118218400)
- 32--- SampleDTO(sno=33, title=title33, regdate=2021-10-05T12:20:45.118218400)

이렇게 처리된다.

```
<ul>

    <th:block th:each="dto, status:${list}">
        <li th:class="${status.even ? 'EVEN': 'ODD'}">AAA</li>

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



Thymeleaf 조건문

th:if, th:unless 가 있지만

삼한연산자를 더 선호하신다 강사님은.

```
<th:block th:switch="${data}">
    <span th:case="a">aa</span>
    <span th:case="b">bb</span>
</th:block>
```

뭐 이렇게 switch 문도 있다고 한다.

링크 주기

링크를 줄때는 @ 를 항상 붙여줘야 하고 th:href 를 사용한다.

```
<ul>
    <li th:each="dto : ${list}" >
        <a th:href="@{/sample/exView(sno=${dto.sno})}">[[
    </li>
</ul>
```

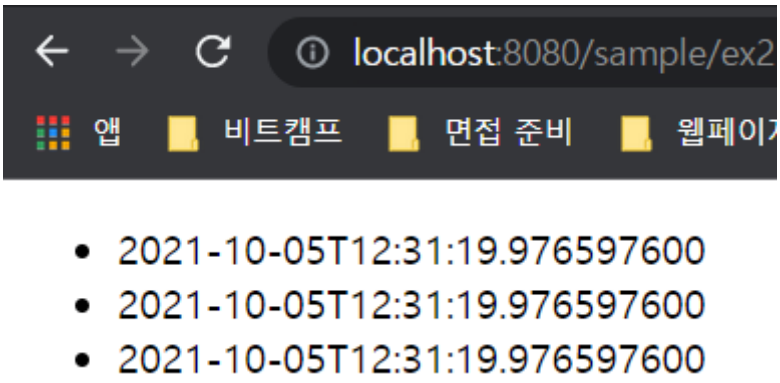

날짜 처리 ★★★★★★

ex2.html ㄱㄱ

다 지우고

```
<body>

<ul>
    <li th:each="dto, status: ${list}">
        [[${dto.regdate}]]
    </li>
</ul>

</body>
```

이렇게 추가

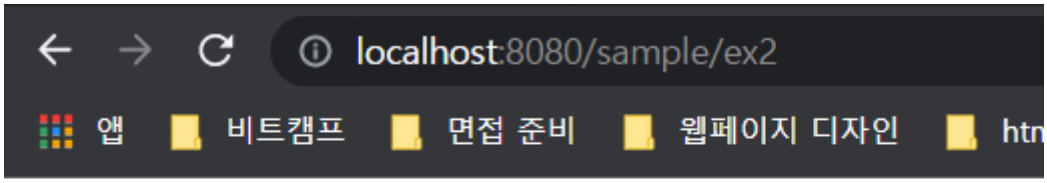


새로 고침하면 날짜가 이렇게 찍힌다. 이쁘게 바꿔주자

```
<body>

<ul>
    <li th:each="dto, status: ${list}">
        [[${dto.regdate}]] --- [[${#temporals.format(dto.regdate,
'yyyy/MM/dd')}]]
    </li>
</ul>

</body>
```

이렇게 한다음

새로고침 ㄱㄱ



- 2021-10-05T12:32:43.229191100 --- 2021/10/05
- 2021-10-05T12:32:43.229191100 --- 2021/10/05
- 2021-10-05T12:32:43.229191100 --- 2021/10/05
- 2021-10-05T12:32:43.229191100 --- 2021/10/05
- 2021-10-05T12:32:43.229191100 --- 2021/10/05
- 2021-10-05T12:32:43.229191100 --- 2021/10/05
- 2021-10-05T12:32:43.229191100 --- 2021/10/05
- 2021-10-05T12:32:43.229191100 --- 2021/10/05
- 2021-10-05T12:32:43.229191100 --- 2021/10/05
- 2021-10-05T12:32:43.229191100 --- 2021/10/05
- 2021-10-05T12:32:43.229191100 --- 2021/10/05
- 2021-10-05T12:32:43.229191100 --- 2021/10/05

왼쪽이 전 오른쪽이 후

참 이쁘게 바뀌었다.

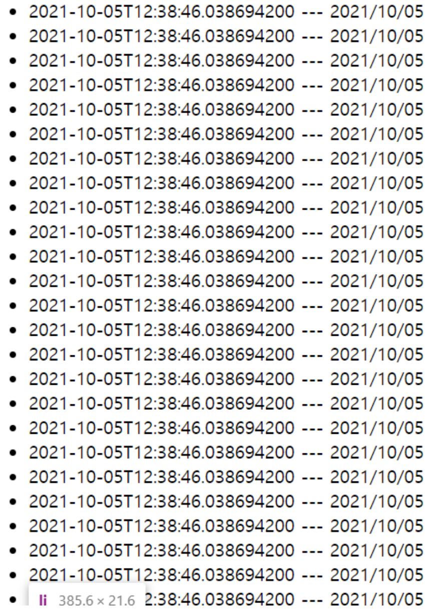
자바스크립트 inline 하기 - Gson, Json 도움 없이

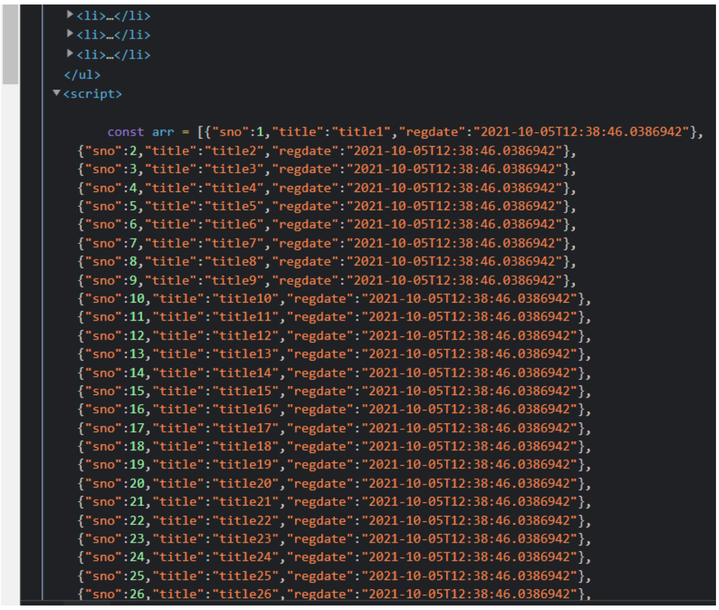
자바스크립트를 직접 안에 넣는 방식(?)

ex2.html

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

    const arr = [[${list}]]

    console.log(arr[0].sno)

</script>
```

새로고침 하면



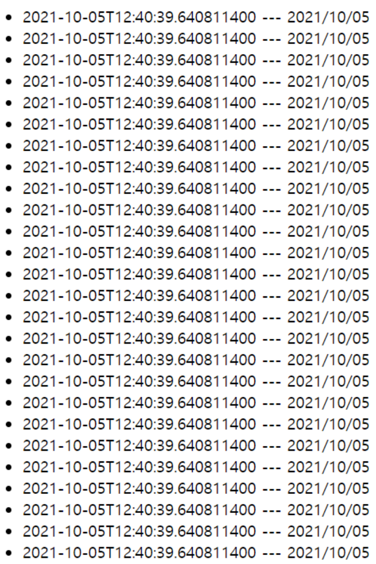


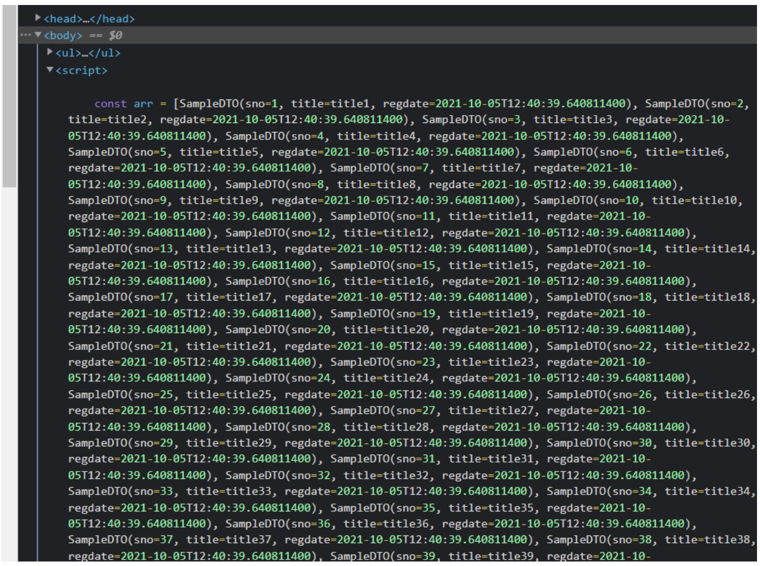
이렇게 나온다.

d3.js 할때의 개고생이 이렇게 간단하게...

```
<script>

    const arr = [[${list}]]

    console.log(arr[0].sno)

</script>
```

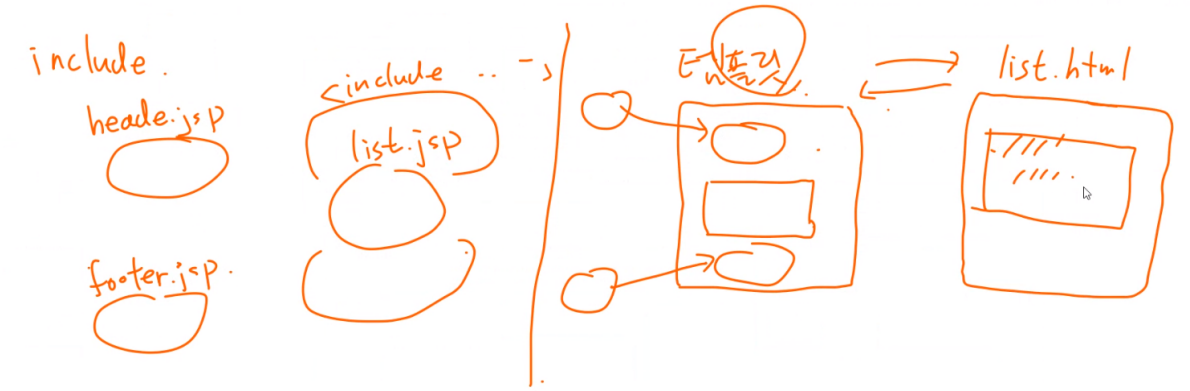
그냥 이렇게 해버리면





이렇게 날라오는데 저거를 예전에는 Gson - Json 처리해서 js 로 들고왔는데 th:inline 을 사용하면 그럴 필요가 없어지는 것이다.

Layout - header, footer 개선 버전



왼쪽이 구방식 오른쪽이 앞으로 배울 내용

[Thymeleaf Layout Dialect]

build.gradle 추가

```
// https://mvnrepository.com/artifact/nz.net.ultraq.thymeleaf/thymeleaf-layout-dialect
implementation group: 'nz.net.ultraq.thymeleaf', name: 'thymeleaf-layout-dialect', version: '2.5.3'
```

그다음

resources - layout 다이렉토리 생성

그다음

layout1.html 생성

이렇게 추가

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org"
xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>

<div>
    <h1>Header....</h1>
</div>

<!--전체 layout 의 1 조각-->
<div layout:fragment="content" >

</div>

<div>
    <h1>Footer.......</h1>
</div>
```

```
</body>
</html>
```

그다음

ex1.html ㄱㄱ

layout:decorator="layout/layout1" 이부분 layout1 로 꼭 바꿔주기

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org"
      xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
      layout:decorator="layout/layout1">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>



<div layout:fragment="content">
<h1>ex1</h1>
<h2 th:text="${'Hello1'}"></h2>
<h2>[[${'Hello2'}]]</h2>
</div>

</body>
</html>
```

이렇게 수정