

스프링 부트 웹 프로젝트

chapter07

프로젝트 구조 만들기

제공된 자료는 훈련생의 수업을 돕기 위한 것으로, 타인과 공유하시면 안됩니다.

Contents

part.1

프로젝트 구성하기

part.2

게시물 목록 조회

part.3

게시물 등록

part.4

게시물 상세 조회

part.5

게시물 수정

part.6

게시물 삭제

part.7

페이징 처리

part.8

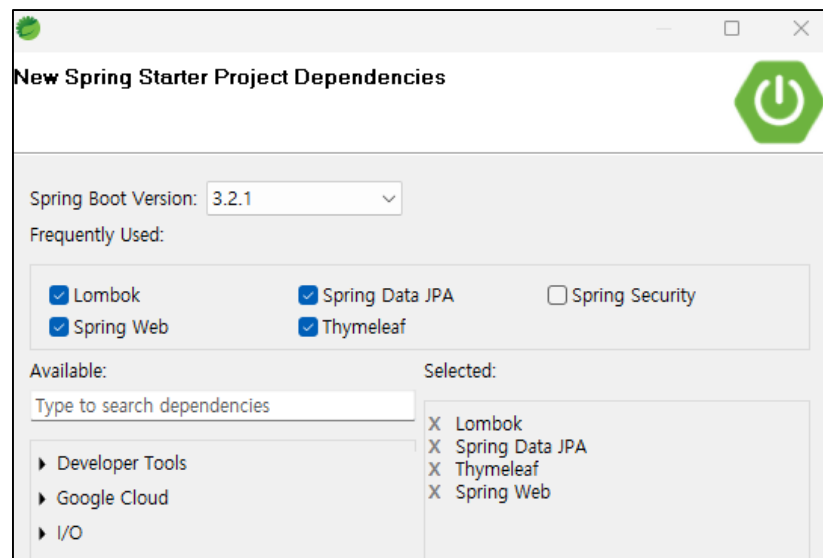
Querydsl

프로젝트 구성

프로젝트 생성하기

실습을 위해 프로젝트를 생성한다.

라이브러리는 'Lombok, Spring Data JPA, Thymeleaf, Spring Web'를 선택한다.



데이터베이스를 연결하기 위해 mariaDB 드라이버와 접속 정보를 추가한다.

```
20 }
21
22 dependencies {
23     implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
24     implementation 'org.springframework.boot:spring-boot-starter-thymeleaf'
25     implementation 'org.springframework.boot:spring-boot-starter-web'
26     compileOnly 'org.projectlombok:lombok'
27     annotationProcessor 'org.projectlombok:lombok'
28     providedRuntime 'org.springframework.boot:spring-boot-starter-tomcat'
29     testImplementation 'org.springframework.boot:spring-boot-starter-test'
30     implementation group: 'org.mariadb.jdbc', name: 'mariadb-java-client', version: '3.1.2'
31 }
```

```
1 spring.datasource.driver-class-name=org.mariadb.jdbc.Driver
2 spring.datasource.url=jdbc:mariadb://localhost:3306/bootex
3 spring.datasource.username=bootuser
4 spring.datasource.password=bootuser
5
6 spring.jpa.hibernate.ddl-auto=update
7 spring.jpa.properties.hibernate.format_sql=true
8 spring.jpa.show-sql=true
9
10 spring.thymeleaf.cache=false
```

화면 설계하기

앞에서 배운 내용을 바탕으로 게시판 기능이 있는 웹사이트는 제작한다.

웹사이트를 제작할 때는 먼저 화면 설계서를 작성하는 것이 좋다.

각 페이지의 구조와 기능을 정확하게 정의해야 하며, 이를 통해 개발의 방향성을 잡을 수 있다.

화면 url과 전달 파라미터 등을 미리 결정하면, 어떤 데이터를 주고 받아야 하는지 명확히 이해할 수 있다.

이러한 준비 작업은 개발 과정에서 발생할 수 있는 혼란을 줄여주고, 작업을 효율적으로 작업할 수 있도록 도와준다.

페이지 목록

- 게시물 등록 페이지
- 게시물 목록 페이지
- 게시물 상세 페이지
- 게시물 수정/삭제 페이지

프로젝트 구성

프로젝트 화면 구성

1. 목록 화면 - 게시물 전체 목록을 조회한다
2. 등록 화면 - 새로운 글을 등록할 수 있고, 등록 처리 후 다시 목록화면으로 이동한다.
3. 조회 화면 - 목록화면에서 특정한 글을 선택하면 상세화면으로 이동한다. 수정/삭제 화면으로 이동할 수 있다.
4. 수정/삭제 화면 - 글을 수정할 수 있고, 수정 처리 후 다시 상세화면으로 이동한다.

GuestBook List Page REGISTER 1

#	Title	Writer	Regdate
3	3번글	도우너	2023/02/21
2	2번글	또치	2023/02/21
1	1번글	들리	2023/02/21

GuestBook Register Page 2

Title
Enter Title

Content

Writer
Enter Writer

GuestBook Read Page 3

Gno
3

Title
3번글

Content
내용입니다

Writer
도우너

RegDate
2023/02/21 23:08:53

ModDate
2023/02/21 23:08:53

GuestBook Modify Page 4

Gno
3

Title
3번글

Content
내용입니다

Writer
도우너

RegDate
2023/02/21 23:08:53

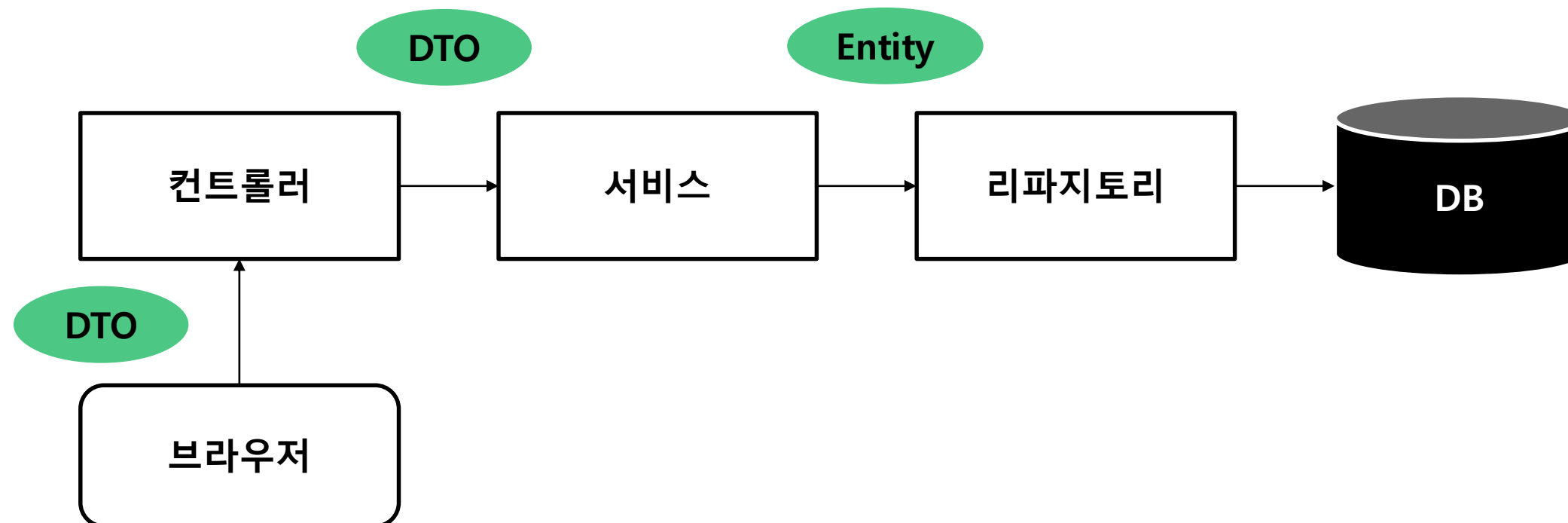
ModDate
2023/02/21 23:08:53

각 페이지의 메소드와 URL주소

기능	URL	GET/POST	기능	Redirect URL
목록조회	/board/list	GET	목록 화면	
등록	/board/register	GET	입력 화면	
		POST	등록 처리	/board/list
상세조회	/board/read	GET	상세 화면	
수정	/board/modify	GET	수정 화면	
		POST	수정 처리	/board/read
삭제	/board/remove	POST	삭제 처리	/board/list

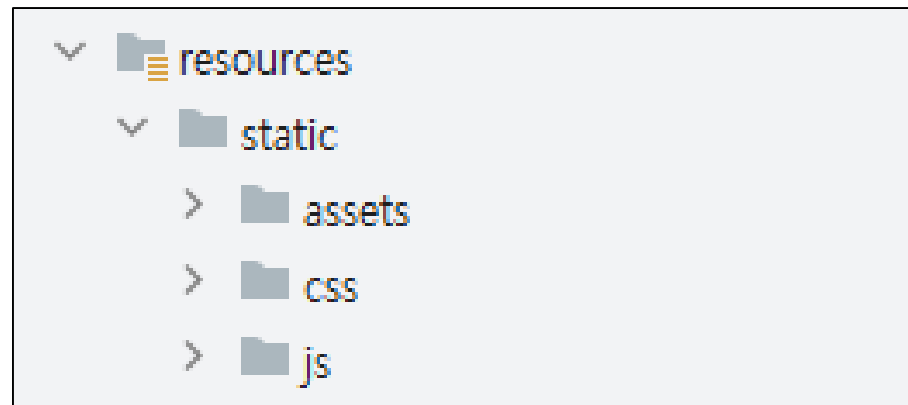
프로젝트의 기본 구조

- Controller: 사용자 요청을 처리한다.
- Service : 데이터를 가공하거나 비즈니스 로직을 처리한다.
- Repository: JPA를 사용해서 구성하고, 실제 데이터를 처리한다.
- DTO: 컨트롤러와 뷰 사이에서 데이터를 전달한다.
- Entity: JPA가 관리하는 객체이며, 컨트롤러와 디비 사이에서 데이터를 전달한다.

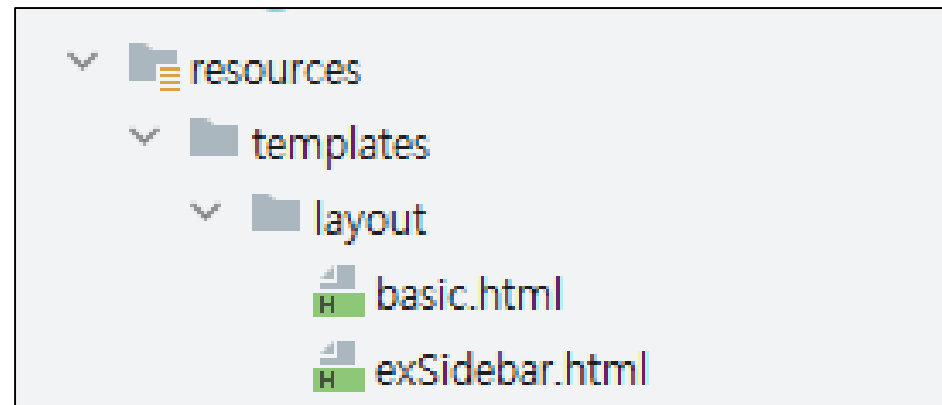


타임리프를 이용하는 설정은 "5장"에서 작성해둔 파일을 그대로 사용한다.
부트스트랩 파일과 basic.html, exSidebar.html 파일을 가져온다.

static 폴더



latout 폴더

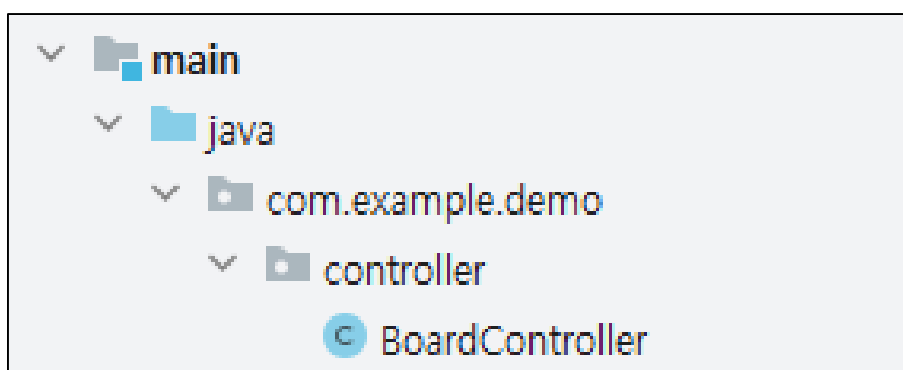


다양한 함수를 사용하기위해 "basic.html" 파일에 자바스크립트의 제이쿼리 라이브러리를 추가한다.

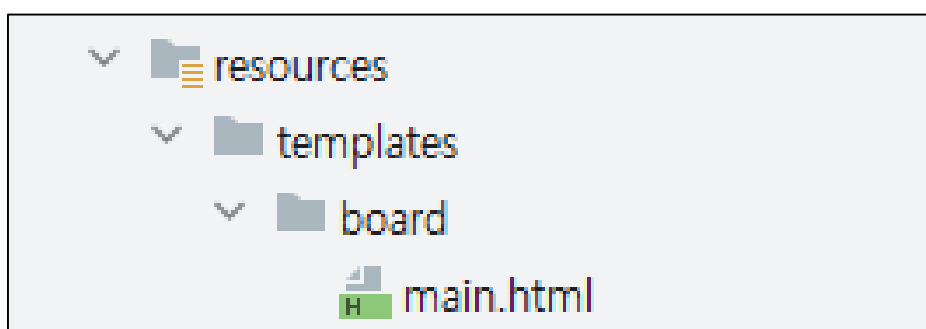
```
<head>
...
  <script src="https://code.jquery.com/jquery-3.4.1.js"></script>
</head>
```

PPT 내용을 복사하세요!

사용자 요청을 처리하기 위해 BoardController 클래스를 추가한다.
그리고 컨트롤러에 메인화면을 반환하는 메소드를 추가한다.



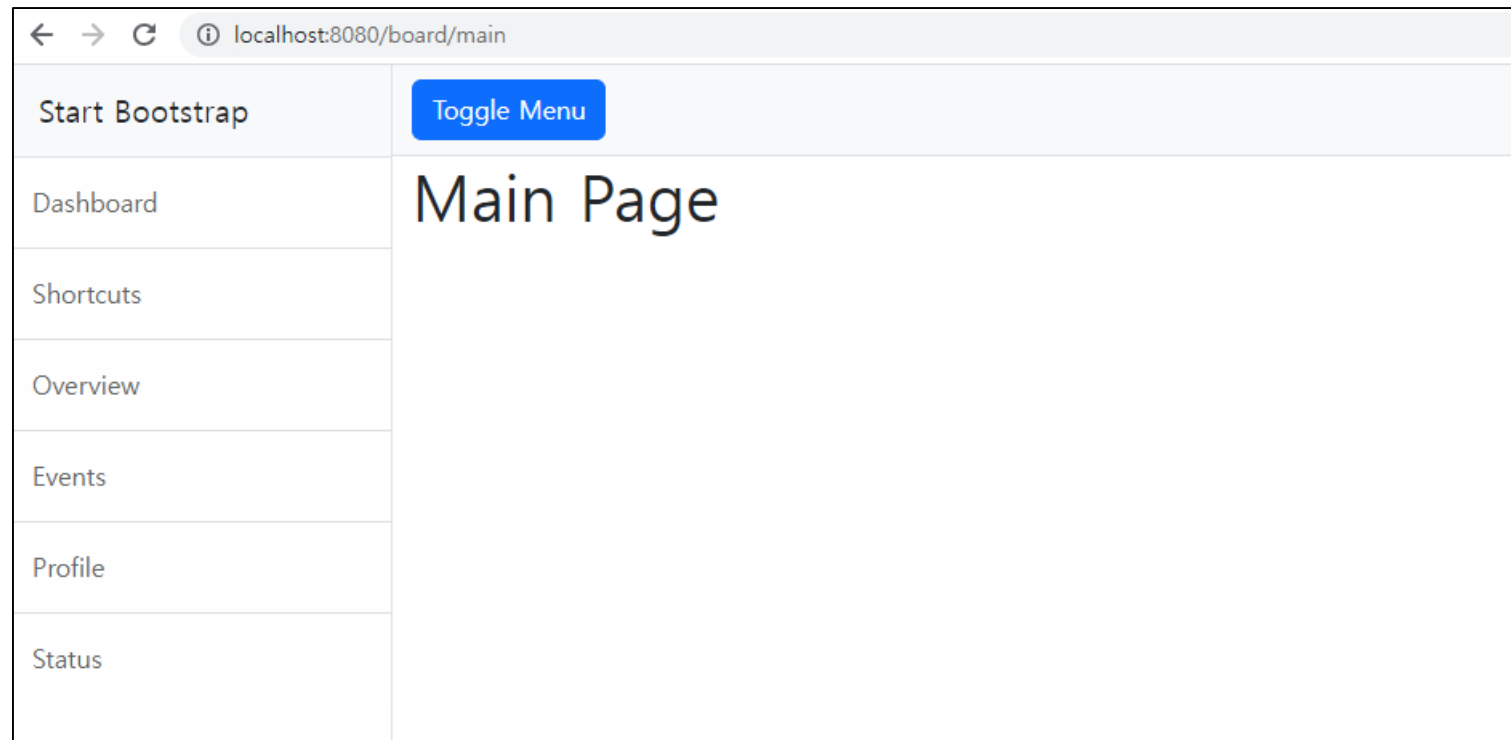
사용자에게 화면을 반환하기 위해 board 폴더를 아래 main.html 파일을 추가하고,
간단히 텍스트를 출력하는 내용을 작성한다.



localhost:8080/board/main 주소로 접속하여, 메인화면이 나타나는지 확인한다.

전체 화면은 basic.html 파일을 사용했고 중간 부분은 main.html의 코드 조각을 사용했다.

결과 화면



BaseEntity를 사용하는 이유

데이터의 등록시간과 수정시간은 자동으로 관리해야 할 때가 많다.

이를 처리하려면 어노테이션을 통해 설정해야 한다.

하지만 엔티티를 만들 때마다 처리하면 번거롭기 때문에, 공통 엔티티를 만들어 재사용하는 것이 좋다.

수정시간과 등록시간을 사용하는 엔티티들

```
public class Board {  
  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    int no;  
  
    @Column(length = 100, nullable = false)  
    String title;  
  
    @Column(length = 1500, nullable = false)  
    String content;  
  
    @ManyToOne  
    Member writer;  
  
    @CreatedDate  
    LocalDateTime regDate;  
  
    @LastModifiedDate  
    LocalDateTime modDate;  
}
```

```
public class Member {  
  
    @Id  
    @Column(length = 50)  
    private String id;  
  
    @Column(length = 200, nullable = false)  
    private String password;  
  
    @Column(length = 100, nullable = false)  
    private String name;  
  
    @CreatedDate  
    LocalDateTime regDate;  
  
    @LastModifiedDate  
    LocalDateTime modDate;  
}
```

수정시간과 등록시간을 상속받은 엔티티

```
public class Board extends BaseEntity {  
  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    int no;  
  
    @Column(length = 100, nullable = false)  
    String title;  
  
    @Column(length = 1500, nullable = false)  
    String content;  
  
    @ManyToOne  
    Member writer;  
}
```

1. BaseEntity 만들기

등록시간과 최종시간을 담당할 엔티티 클래스를 만든다.

2. Board 만들기

게시물 엔티티에서 BaseEntity를 상속받으면, 등록시간과 수정시간이 자동으로 추가된다.

BaseEntity

```
abstract class BaseEntity {  
  
    @CreatedDate  
    LocalDateTime regDate;  
  
    @LastModifiedDate  
    LocalDateTime modDate;  
  
}
```

Board 엔티티

```
public class Board extends BaseEntity {  
  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    int no;  
  
    @Column(length = 100, nullable = false)  
    String title;  
  
    @Column(length = 1500, nullable = false)  
    String content;  
  
    @ManyToOne  
    Member writer;  
  
}
```

3. 테이블 생성하기

프로젝트를 시작하면 자동으로 테이블이 생성된다.

BaseEntity를 상속받았기 때문에, 게시물 테이블에 등록일과 수정일 컬럼이 추가되었다.

콘솔 로그

Hibernate:




```
create table board (  
  id integer not null auto_increment,  
  mod_date datetime(6),  
  reg_date datetime(6),  
  content varchar(1500) not null,  
  title varchar(100) not null
```

4. 리파지토리 만들기

repository 패키지 아래 BoardRepository를 추가한다.

단위테스트를 사용하여 데이터를 추가한다.

새로 추가된 데이터의 등록시간과 수정시간이 자동으로 입력된 것을 확인한다.

	 no	 mod_date	 reg_date
1	1	2024-01-14 17:08:50.608	2024-01-14 17:08:50.608

엔티티에서 사용하는 어노테이션

어노테이션	내용	위치
@MappedSuperclass	해당클래스는 테이블로 생성되지 않는다	엔티티
@EntityListeners	데이터가 추가 또는 수정되는 것을 감지한다	엔티티
@EnableJpaAuditing	리스너를 활성화시키는 역할이다	메인클래스
@CreatedDate	데이터가 등록될 때 최초등록시간을 저장한다	엔티티
@LastModifiedDate	데이터가 수정될 때 최종수정시간을 저장한다	엔티티

Q. 단위테스트와 리파지토리를 이용하여 게시물 조회, 수정, 삭제를 처리한다.

```
@Test
void 게시물목록조회() { }

@Test
void 게시물단건조회() { }

@Test
void 게시물수정() { }

@Test
void 게시물삭제() { }
```

게시물을 수정하고 최종수정시간이 현재시간으로 변경되었는지 확인한다.

123 no	ABC TITLE	ABC writer	reg_date	mod_date
1	1번글	둘리	2023-07-20 18:26:35.023	2023-07-20 18:27:58.927

DTO와 엔티티

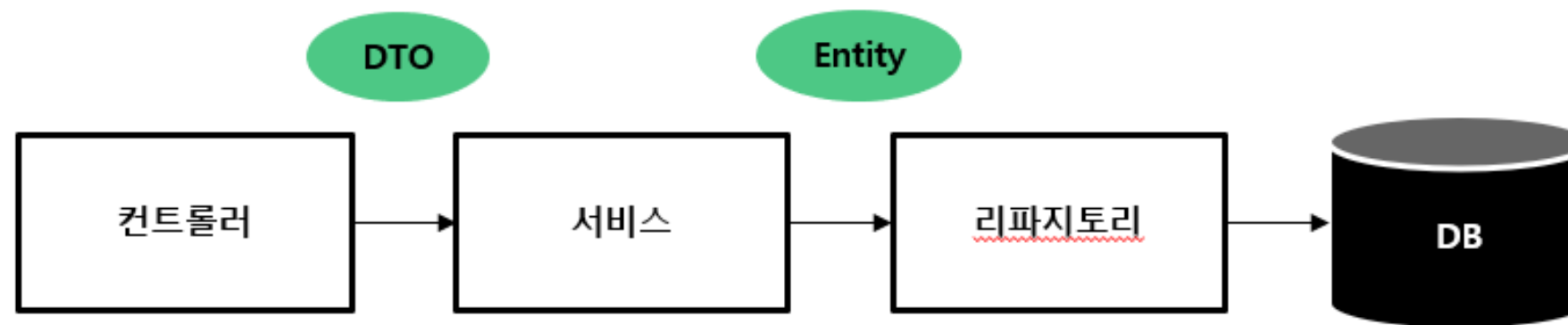
구조는 비슷하지만, 목적지에 따라 클래스를 분리해서 사용하는 것이 좋다.

DTO: 컨트롤러와 뷰 사이에서 데이터를 전달하는 객체

엔티티: 컨트롤러와 DB 사이에서 데이터를 전달하는 객체

서비스

서비스는 데이터를 DTO로 전달 받고, 리파지토리는 엔티티로 받기 때문에 중간에 DTO를 엔티티로 변환하는 작업이 필요하다.

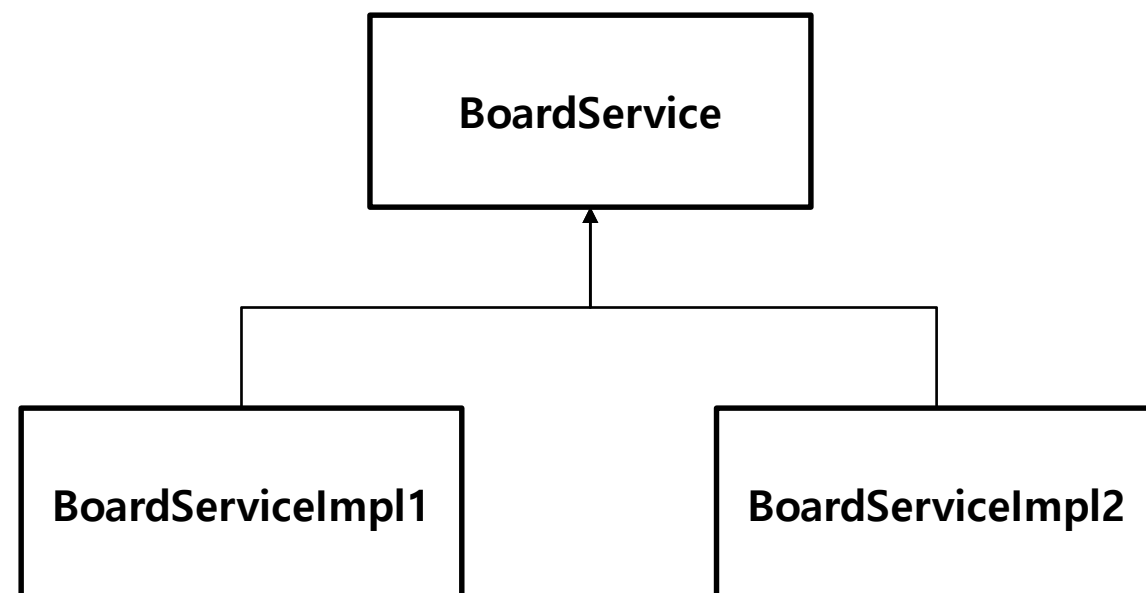
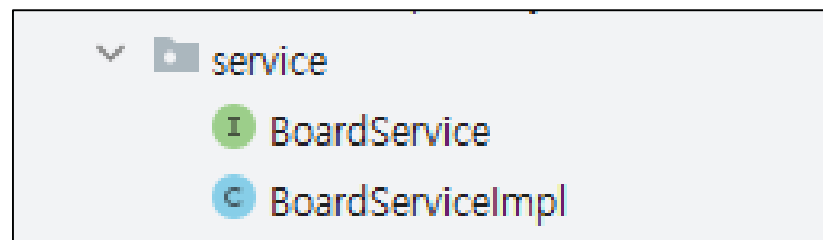


서비스의 구조

서비스는 인터페이스와 구현클래스, 계층 구조로 구성된다.

인터페이스를 사용하면 나중에 서비스의 구현 코드를 변경하거나 교체할 때, 쉽게 변경할 수 있다.

서비스 계층 구조



서비스를 사용하는 코드

```
@Controller
@RequestMapping("/board")
public class BoardController {

    @Autowired
    BoardService service;

    @GetMapping("/list")
    public void list(Model model) {

        List<BoardDTO> list = service.getList();
        model.addAttribute("list", list);
    }
}
```

서비스 구현 클래스 대신
서비스 인터페이스를 사용함

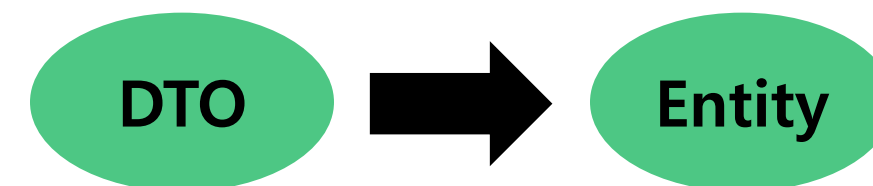
서비스 인터페이스 만들기

- 게시물을 등록하는 메소드를 추가한다.
- DTO를 엔티티로 변환하는 메소드를 추가한다.
- 엔티티를 DTO로 변환하는 메소드를 추가한다.
- 자바 8부터는 인터페이스도 default 키워드를 사용해서 일반 메소드를 추가할 수 있다.

인터페이스

```
public interface BoardService {  
    int register(BoardDTO dto);  
}
```

```
default Board dtoToEntity(BoardDTO dto) {  
    Board entity = Board.builder()  
        .no(dto.getNo())  
        .title(dto.getTitle())  
        .content(dto.getContent())  
        ...  
    return entity;  
}
```



서비스 구현 클래스 만들기

- 인터페이스에서 상속받은 등록 메소드를 구현한다.
- dtoToEntity() 메소드를 사용하여 파라미터로 전달받은 DTO를 엔티티로 변환한다.
- 데이터를 등록하고, 새로운 게시물의 번호를 반환한다.

구현 클래스

```
class BoardServiceImpl implements BoardService {  
    int register(BoardDTO dto) {  
        Board entity = dtoToEntity(dto);  
        repository.save(entity);  
        return entity.getNo();  
    }  
}
```

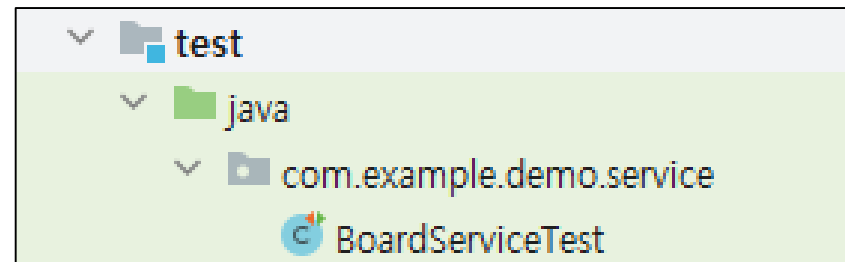
새로운 게시물

entity	Board (id=121)
> content	"내용입니다" (id=109)
> modDate	LocalDateTime (id=141)
> no	1
> regDate	LocalDateTime (id=141)
> title	"1번글" (id=116)
> writer	"둘리" (id=117)

서비스의 테스트

- 단위테스트를 통해 새로운 게시물을 등록한다.
- DTO가 엔티티로 변환되었는지 확인한다.

단위 테스트



```
@Test  
void 게시물등록() { }
```

dto객체를 엔티티 객체로 변환 확인

```
2024-01-14 18:18:25.152+09:00 INFO 58784 --- [Test worker] c.example.demo.service.BoardService :  
BoardDTO(no=0, title=2번글, content=내용입니다, writer=또치, regDate=null, modDate=null)  
Board(no=4, title=2번글, content=내용입니다, writer=또치)
```

목록 조회 기능

- 모든 게시물 목록을 출력한다.
- 게시물이 가지고 있는 모든 정보를 출력할 필요는 없다.

결과 화면

Board List Page		게시물 등록	
#	제목	작성자	등록일
1	1번글	둘리	2024/01/15
2	2번글	또치	2024/01/15
3	3번글	도우너	2024/01/15

DTO 리스트 출력

서비스에서 목록 처리

- 게시물 목록을 조회하는 메소드를 추가한다.
- 리파지토리를 통해서 화면에서 필요한 게시물 목록 데이터를 꺼내온다.
- entityToDTO() 메소드를 사용하여 엔티티 리스트를 DTO 리스트로 변환한다

인터페이스

```
List<BoardDTO> getList();

default BoardDTO entityToDto(Board entity){
    BoardDTO dto = BoardDTO.builder()
                            .no(entity.getNo()) ...

    return dto;
}
```

구현 클래스

```
@Override
public List<BoardDTO> getList() {
    List<Board> result = repository.findAll();
    list = result.stream()
                .map(entity -> entityToDto(entity))
                ...

    return list;
}
```

서비스의 테스트

- 단위테스트를 통해 게시물 목록을 조회한다.
- 엔티티 리스트가 DTO 리스트로 변환되었는지 확인한다.

단위테스트

```
@SpringBootTest
public class BoardServiceTest {

    @Autowired
    BoardService service;
```

```
@Test
public void 게시물목록조회() {

}
```

엔티티 객체를 DTO 객체로 변환 확인

```
Board(no=1, title=1번글, content=내용입니다, writer=둘리)
Board(no=2, title=2번글, content=내용입니다, writer=또치)
Board(no=3, title=3번글, content=내용입니다, writer=도우너)
BoardDTO(no=1, title=1번글, content=내용입니다, writer=둘리, regDate=
BoardDTO(no=2, title=2번글, content=내용입니다, writer=또치, regDate=
BoardDTO(no=3, title=3번글, content=내용입니다, writer=도우너, regDat
```


컨트롤러에서 목록 처리

- 게시물 목록 화면을 반환하는 메소드를 추가한다.
- 서비스를 사용하여 게시물 리스트를 가져온다.
- 화면에 결과 데이터를 전달한다.

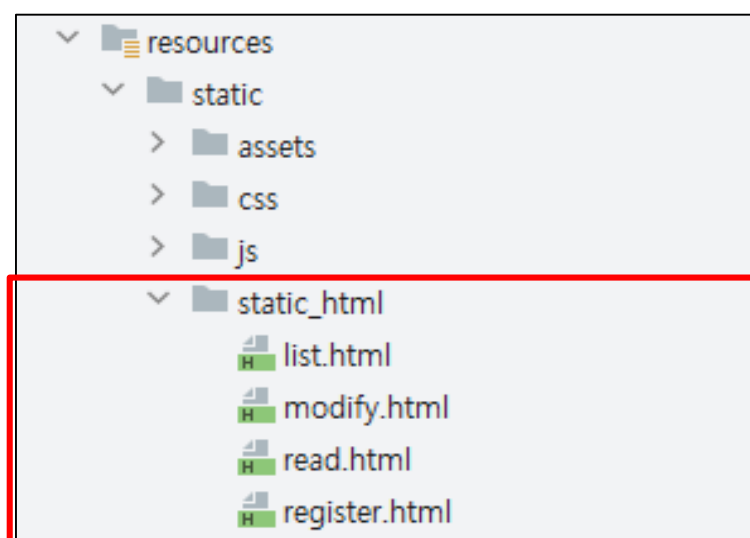
BoardController

```
@GetMapping("/list")
public void list(Model model) {
    List<BoardDTO> list = service.getList();
    model.addAttribute("list", list);
}
```

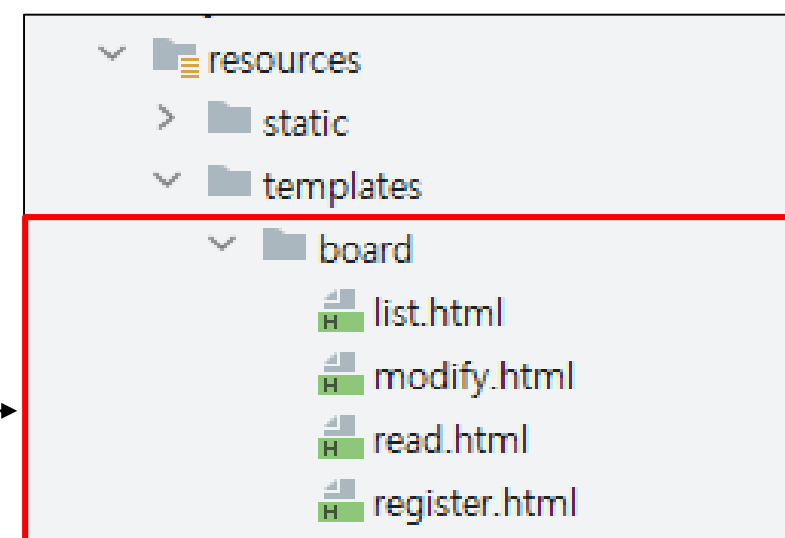
정적 페이지 활용하기

- 해당 프로젝트는 만들어진 정적 페이지를 사용한다.
- 정적 파일은 이미 기본적인 구조를 가지고 있다.
- 정적 파일에서 동적인 부분만 타임리프로 처리한다.
- spring study의 Project07에 있는 HTML 파일을 자신의 프로젝트로 가져간다.
(/resources/static/static_html/아래파일 → 복사 → resource/templates/board/)

예제 프로젝트



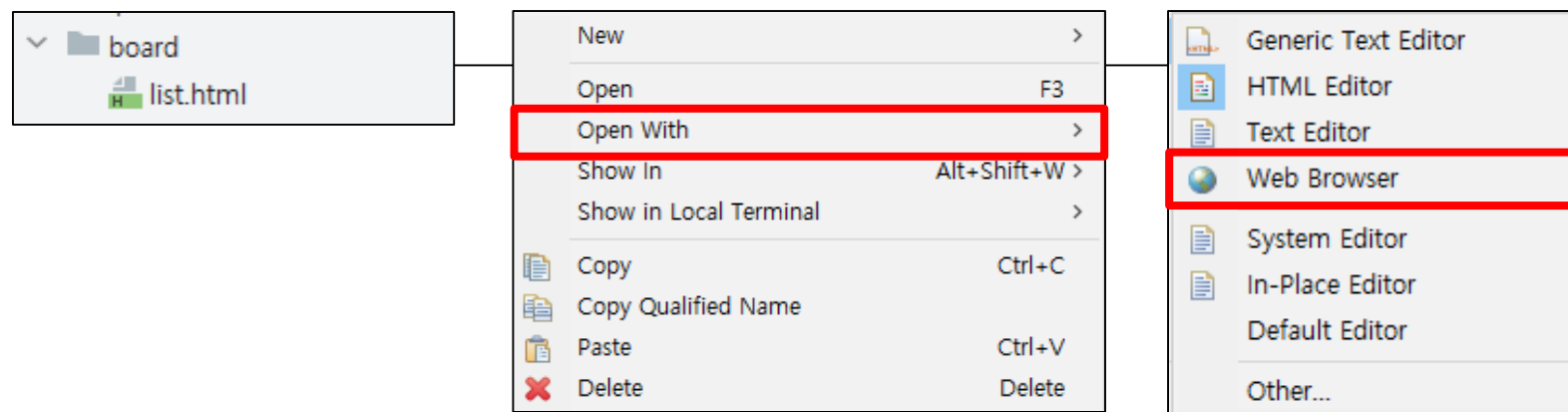
내 프로젝트



정적 페이지 활용하기

- list.html을 브라우저에서 열어서 내용을 확인한다.
- 파일 선택 > 우클릭 > Open With 메뉴 > Web Brower 메뉴

브라우저에서 열기



결과 화면



레이아웃 적용

- 목록 화면을 조각으로 만들고, 레이아웃을 적용한다.
- basic.html 파일에 list.html의 코드 조각을 전달한다.

list.html

```
<th:block th:replace="~/layout/basic :: setContent(~{this::content} )">

  <th:block th:fragment="content">

기존 HTML 내용



  </th:block>

</th:block>
```

Simple Sidebar, 부트 스트랩 사이트, <https://startbootstrap.com/template/simple-sidebar>

리스트 출력

- "방명록 등록" 버튼을 클릭하면 등록화면으로 이동한다.
- 전달받은 게시물 리스트를 테이블을 이용해서 출력한다.
- 반복문을 사용해서 리스트를 처리한다.
- "게시물 번호"를 클릭하면 상세화면으로 이동한다.
- 날짜는 "년/월/일" 포맷으로 출력한다.

list.html

```
<a th:href="@{/board/register}"></a>
```

```
<tbody>  
  <tr th:each="dto : ${list}">  
    <td>[[${dto.title}]]</td>  
    <td>[[${dto.writer}]]</td>  
    ...  
  </tr>  
</tbody>
```

[게시물 등록](#)

#	제목	작성자	등록일
1	1번글	둘리	2023/02/26

테스트

localhost:8080/board/list 주소로 접속한다.

게시물 목록이 정상적으로 출력됐는지 확인한다.

결과 화면

Board List Page 게시물 등록			
#	제목	작성자	등록일
1	1번글	둘리	2023/07/23
2	2번글	또치	2023/07/23

목록조회 기능의 동작 과정을 확인하기 위해 다음과 같은 단계를 수행한다.

1. 프로젝트 실행

- 클래스마다 메소드에 브레이크 포인트를 건다.
- 디버깅 모드로 프로젝트를 실행한다.

컨트롤러

```
25 @GetMapping("/list")
26 public String list(Model model){
27     List<GuestbookDTO> list = service.ge
28     model.addAttribute("list", list);
29 }
```

서비스 인터페이스

```
25 default GuestbookDTO entityToDto(Gues
26
27     GuestbookDTO dto = GuestbookDTO.
```

서비스 구현 클래스

```
33 public List<GuestbookDTO> getList() {
34     List<Guestbook> result = repository.f
35     List<GuestbookDTO> list = new ArrayLi
```

2. 목록 페이지 호출

- localhost:8080/board/list

3. 코드 디버깅

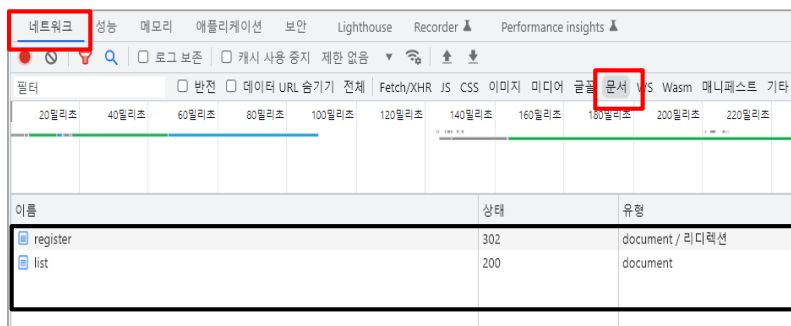
- 코드를 한 줄씩 실행하면서, 변수에 저장된 값을 확인한다.

(x) Variables × Breakpoints Expressions	
Name	Value
<init>() returned	(No explicit return value)
> this	GuestbookServiceImpl
> result	ArrayList<E> (id=208)
list	ArrayList<E> (id=209)

이번에는 브라우저에서 동작과정을 확인한다.

1. 개발자도구 설정

- request만 확인 할 수 있도록 필터링을 건다. (안하면 리소스 요청까지 다 보임)
- F12 키 > 네트워크 탭 선택 > 문서 필터링 선택



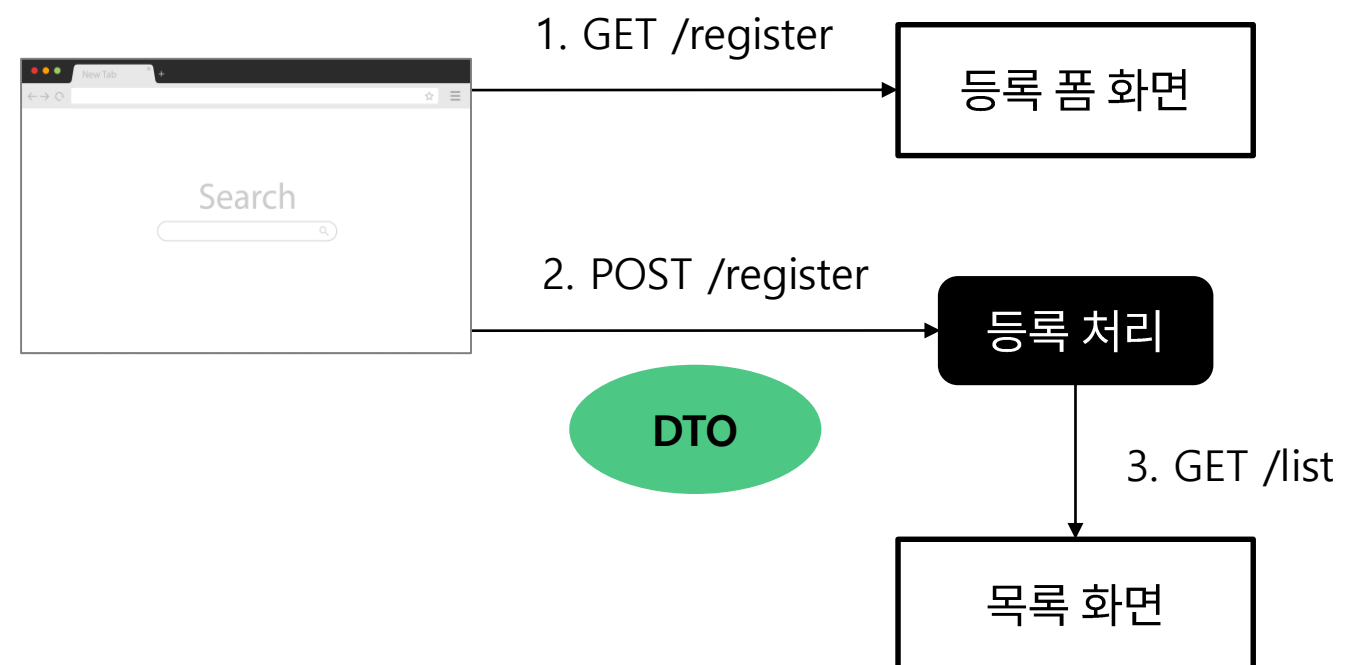
2. 브라우저에서 요청/응답메세지, HTML코드 확인

Name	Status
list	200

```
<table class="table table-striped">
  <thead>
    <tr>
      <th scope="col">#</th>
      <th scope="col">제목</th>
      <th scope="col">작성자</th>
      <th scope="col">등록일자</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <th>1</th>
      <td>1번글</td>
      <td>관리</td>
      <td>2023/02/22</td>
    </tr>
    <tr>
      <th>2</th>
      <td>2번글</td>
      <td>도치</td>
      <td>2023/02/22</td>
    </tr>
  </tbody>
</table>
```


등록 기능

- 사용자에게 게시물 정보를 입력 받는다.
- 게시물 등록 후 다시 목록화면으로 돌아간다.
- 처리되었다는 결과를 보여주기 위해 모달창을 띄운다.



Board Register Page

제목

내용

작성자

등록

Board List Page

게시물 등록

#	제목	작성자
1	1번글	둘리
2	2번글	또치
3	localhost:8080 내용: 3번 글이 등록되었습니다.	

확인

컨트롤러에서 등록 처리

- GET방식으로 화면을 반환하는 메소드를 추가한다.
- post방식으로 등록을 처리하는 메소드를 추가한다.
- 등록폼에서 전달한 데이터를 파라미터로 수집한다.
- 등록 후 목록화면으로 리다이렉트한다.
- 이때, 새로운 게시물 번호를 화면에 전달한다.

BoardController

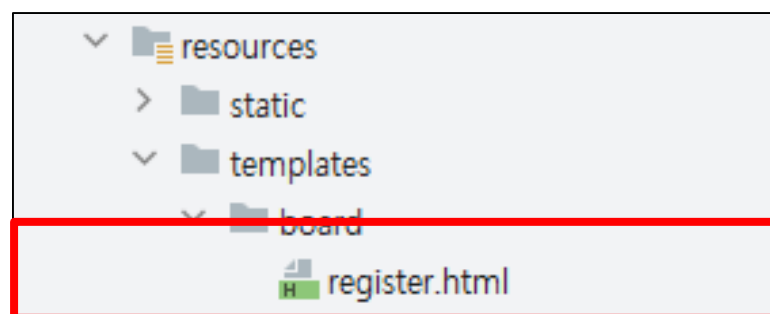
```
@GetMapping("/register")
public void register() { }

@PostMapping("/register")
public String registerPost(BoardDTO dto, RedirectAttributes redirectAttributes) {
    int no = service.register(dto)
    redirectAttributes.addFlashAttribute("msg", no);
    return "redirect:/board/list";
}
```

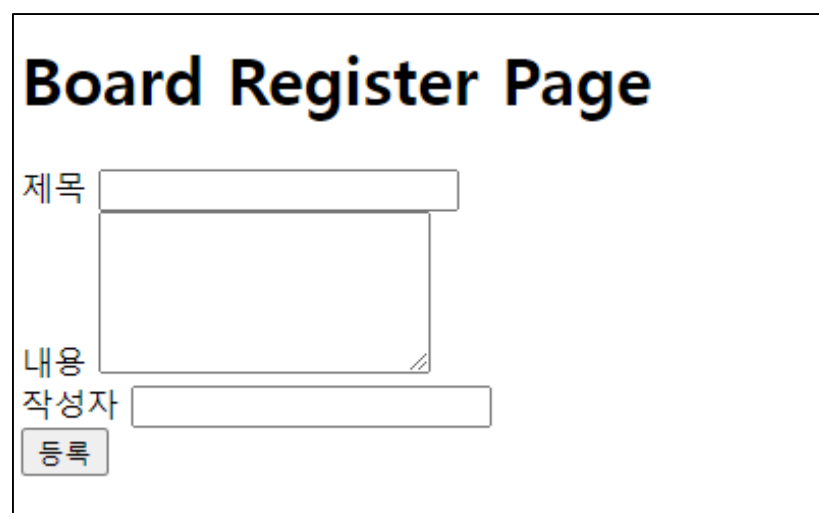
정적 페이지 활용하기

- register.html을 브라우저에서 열어서 내용을 확인한다.
- 파일 선택 > 우클릭 > Open With 메뉴 > Web Brower 메뉴

정적 파일



결과 화면

A screenshot of a web form titled "Board Register Page". The form has three input fields: "제목" (Title), "내용" (Content), and "작성자" (Author). Below the "작성자" field is a "등록" (Register) button.

레이아웃 적용

- 목록 화면을 조각으로 만들고, 레이아웃을 적용한다.
- basic.html 파일에 register.html의 코드 조각을 전달한다.

register.html

```
<th:block th:replace="~/layout/basic :: setContent(~{this::content} )">

  <th:block th:fragment="content">

기존 HTML 내용



  </th:block>

</th:block>
```

입력양식 만들기

- Form 태그를 사용하여 게시물 정보를 입력 받는다.
- action속성에 주소를 입력하고, POST방식으로 처리한다.
- 데이터가 DTO로 수집되므로 입력필드에 적절한 name값을 지정한다.
- 등록 버튼을 클릭하면, 등록처리가 호출되고 사용자가 입력한 데이터가 전달된다.

register.html

```
<form th:action="@{/board/register}" th:method="post">
```

```
<div>  
  <label >제목</label>  
  <input type="text" name="title">  
</div>
```

등록처리와 목록화면의 모달창

- 등록 처리 후 사용자에게 등록 처리 결과를 보여준다.
- 목록화면에서 자바스크립트를 이용해서 전달받은 msg를 출력한다.
- document 객체: 해당 html 문서를 담고 있는 객체
- ready() 메소드 : 문서가 만들어질 때 동작하는 함수, 수행할 코드를 블록 안에 작성

list.html

```
<script th:inline="javascript">
$(document).ready(function(){
    var msg = [{msg}];
    if(msg != null){
        alert(msg + "번 글이 등록되었습니다.");
    }
});
</script>
```

테스트

1. 목록화면에서 [게시물 등록] 버튼을 클릭한다.

게시물 등록

2. 등록화면에서 게시물 정보를 입력하고 [등록] 버튼을 클릭한다.

Board Register Page

제목

3번글

내용

내용입니다

작성자

도우너

등록

3. 목록화면에서 게시물이 정상적으로 등록되었는지 확인한다.

Board List Page			
#	제목	작성자	등록일
1	1번글	둘리	2023/07/23
2	2번글	또치	2023/07/23
3	3번글	도우너	2023/07/23

localhost:8080 내용:
3번 글이 등록되었습니다.

확인

이제 31~32 페이지를 보고
다시 테스트 해보세요!

상세 조회 기능

- 특정 게시물 정보를 출력한다.
- 게시물이 가지고 있는 모든 정보를 출력한다.
- 정보를 읽기만 하고, 수정할 수 없다.
- 수정 버튼을 클릭하면 수정페이지로 이동한다

결과 화면

Board Read Page	
번호	1
제목	1번글
내용	내용입니다
작성자	둘리
등록일	2024/01/16 18:29:38
수정일	2024/01/16 18:29:38
수정	목록으로

DTO 출력

서비스에서 조회 처리

- 특정 게시물을 조회하는 메소드를 추가한다.
- 구현 클래스에서는 상속받은 조회 메소드를 구현한다.
- entityToDTO() 메소드를 사용하여 엔티티 객체를 DTO 객체로 변환한다.

인터페이스

```
BoardDTO read(int no);
```

구현 클래스

```
@Override  
public BoardDTO read(int no) {  
    Optional<Board> result = repository.findById(no);  
    Board board = result.get();  
    return entityToDto(board);  
}
```

서비스의 테스트

- 단위테스트를 통해 특정 게시물 정보를 조회한다.
- 엔티티 객체가 DTO 객체로 변환되었는지 확인한다.

단위테스트

```
@Test
public void 게시물단건조회() {

}
```

스스로 진행해보세요~

엔티티 객체를 DTO 객체로 변환 확인

✓ 테스트 통과: 1 / 1개 테스트 - 236ms

b1_0.no=?

Board(no=1, title=1번글, content=내용입니다, writer=둘리)

BoardDTO(no=1, title=1번글, content=내용입니다, writer=둘리, reg

컨트롤러에서 조회 처리

- 게시물 상세 화면을 반환하는 메소드를 추가한다.
- 파라미터로 글번호를 수집한다.
- 서비스를 사용하여 특정 게시물 정보를 가져온다.
- 화면에 결과 데이터를 전달한다.

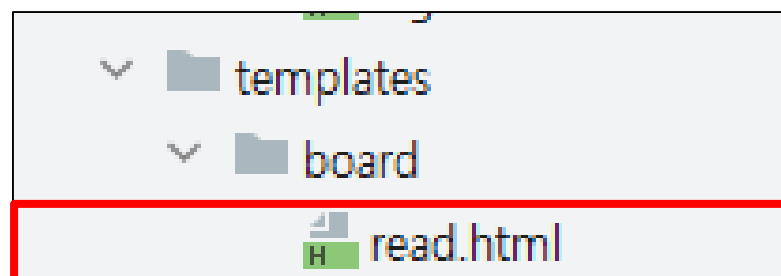
BoardController

```
@GetMapping("/read")
void read(int no) {
    BoardDTO dto = service.read(no);
    model.addAttribute("dto", dto);
}
```

정적 페이지 활용하기

- read.html을 브라우저에서 열어서 내용을 확인한다.
- 파일 선택 > 우클릭 > Open With 메뉴 > Web Brower 메뉴

정적 파일



결과 화면

Board Read Page

번호

1

제목

1번글

내용

내용입니다

작성자

둘리

등록일

2023/02/21 10:00:00

수정일

2023/02/21 10:00:00

수정

목록으로

레이아웃 적용

- 목록 화면을 조각으로 만들고, 레이아웃을 적용한다.
- basic.html 파일에 read.html의 코드 조각을 전달한다.

read.html

```
<th:block th:replace="~/layout/basic :: setContent(~{this::content} )">

  <th:block th:fragment="content">

기존 HTML 내용



  </th:block>

</th:block>
```

조회 화면 만들기

- 전달받은 게시물 정보를 입력필드를 이용하여 출력한다.
- 값을 변경할 수 없도록 읽기모드로 설정한다.
- 날짜는 "년/월/일 시:분:초" 패턴으로 출력한다.
- 수정 버튼을 클릭하면 수정화면으로 이동한다.
- 목록으로 버튼을 클릭하면 목록화면으로 이동한다.

read.html

```
<label>번호</label>
<input type="text" th:value="${dto.no}" readonly>
<label>제목</label>
<input type="text" th:value="${dto.title}" readonly>

<a th:href="@{/board/modify(no = ${dto.no})}">
    <button>수정</button>
</a>
<a th:href="@{/board/list}">
    <button>목록으로</button>
</a>
```

테스트

1. 목록화면에서 [글번호]를 클릭한다.
2. 상세화면에서 게시물 정보가 정상적으로 출력되는지 확인한다.

목록 화면

Board List Page			게시물 등록
#	제목	작성자	
1	1번글	둘리	
2	2번글	또치	

상세 화면

Board Read Page	
번호	1
제목	1번글
내용	내용입니다

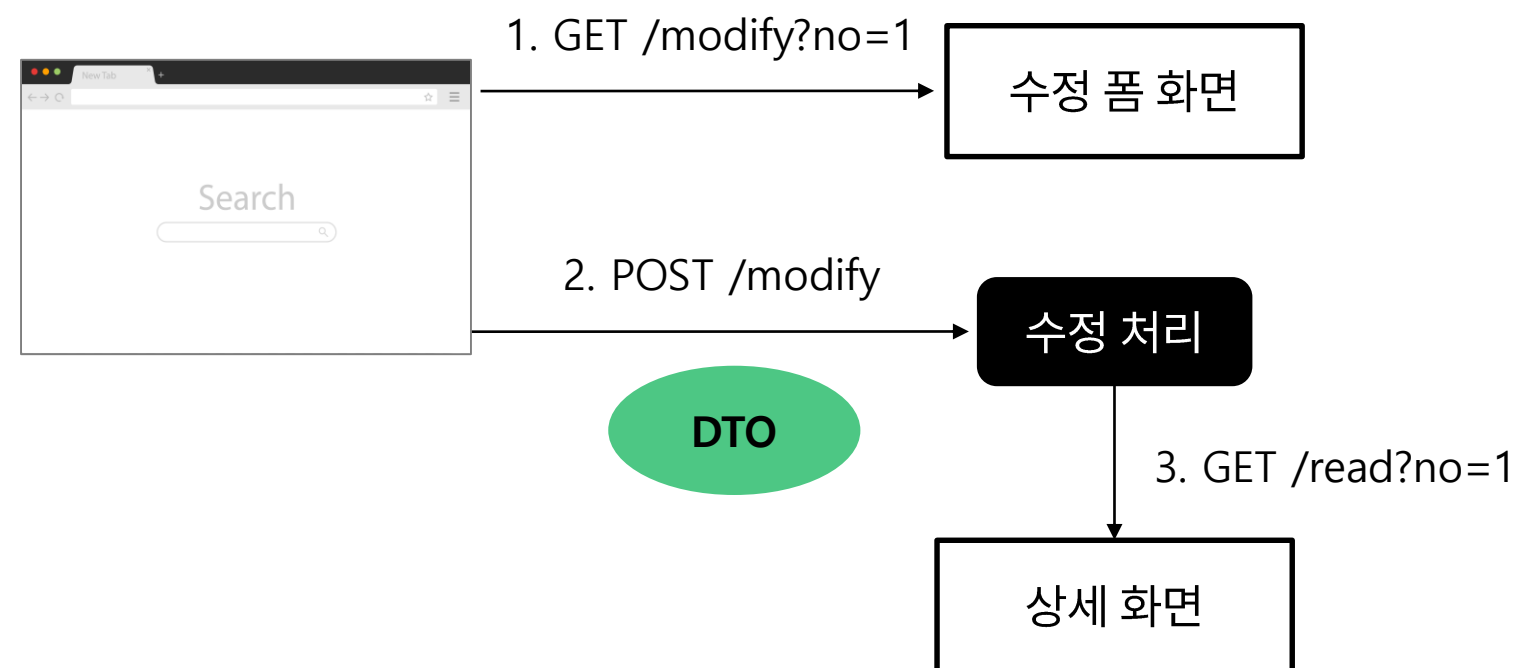
이제 31~32 페이지를 보고
다시 테스트 해보세요!

게시물 수정

수정 처리

수정 기능

- 기존 게시물 정보를 출력한다.
- 제목과 내용만 수정할 수 있다.
- 게시물 수정 후 상세화면으로 이동한다.



Board Modify Page	
번호	1
제목	
1번글	
내용	내용입니다
작성자	둘리
등록일	2024/01/16 18:29:38
수정일	2024/01/16 18:29:38
수정 목록으로 삭제	

Board Read Page	
번호	1
제목	
1번글	
내용	수정되었습니다~
작성자	둘리
등록일	2024/01/16 18:29:38
수정일	2024/01/16 20:15:33
수정 목록으로	

게시물 수정

수정 처리

서비스에서 수정 처리

- 게시물 정보를 수정하는 메소드를 추가한다.
- 구현 클래스에서는 상속받은 수정 메소드를 구현한다.
- 기존의 엔티티에서 제목과 내용만 수정하고 다시 저장한다.

인터페이스

```
void modify(BoardDTO dto);
```

구현 클래스

```
public void modify(BoardDTO dto) {  
    Optional<Board> result = repository.findById(dto.getNo());  
    Board entity = result.get();  
    entity.setTitle(dto.getTitle());  
    entity.setContent(dto.getContent());  
    repository.save(entity);  
}
```

게시물 수정

수정 처리

서비스의 테스트

- 단위테스트를 통해 특정 게시물 정보를 수정한다.
- 제목 또는 내용이 변경되었는지 확인한다.
- 최종 수정 시간이 현재시간으로 변경되었는지 확인한다.

단위테스트

```
@Test
public void 게시물수정() {

}
```

스스로 진행해보세요~

게시물 테이블

<small>123</small> no	<small>🕒</small> mod_date	<small>🕒</small> reg_date	<small>ABC</small> content	<small>ABC</small> title	<small>ABC</small> writer
1	2024-01-16 20:31:55.992	2024-01-16 18:29:38.454	내용이수정되었습니다~	1번글	둘리

컨트롤러에서 등록 처리

- GET방식으로 화면을 반환하는 메소드를 추가한다.
- 글번호를 파라미터로 수집한다.
- POST방식으로 수정을 처리하는 메소드를 추가한다.
- 게시물 정보를 파라미터로 수집한다.
- 수정 후 상세화면으로 리다이렉트한다.
- `addAttribute()` 메소드를 이용하여 URL에 파라미터를 추가한다. (`/board/read?no=1`)

BoardController

```
@GetMapping("/modify")  
void modify(int no) { }
```

```
@PostMapping("/modify")  
public String modifyPost(BoardDTO dto) { }
```

정적 페이지 활용하기

- modify.html을 브라우저에서 열어서 내용을 확인한다.
- 파일 선택 > 우클릭 > Open With 메뉴 > Web Brower 메뉴

정적 파일



결과 화면

A screenshot of a web browser displaying the 'Board Modify Page'. The browser's address bar shows the file path: D:/0.tool/spring-workspace/project06/src/main/resources/static/static_html/modify.html. The page title is 'Board Modify Page'. The form contains the following fields:

- 번호 (Number): 1
- 제목 (Title): 1번
- 내용 (Content): 내용입니다 (highlighted with a red box)
- 작성자 (Author): 돌리
- 등록일 (Registration Date): 2023/02/21 10:00:00
- 수정일 (Modification Date): 2023/02/21 10:00:00

At the bottom of the form, there are three buttons: '수정' (Modify), '목록으로' (Back to List), and '삭제' (Delete).

레이아웃 적용

- 목록 화면을 조각으로 만들고, 레이아웃을 적용한다.
- basic.html 파일에 modify.html의 코드를 조각을 전달한다.

modify.html

```
<th:block th:replace="~/layout/basic :: setContent(~{this::content} )">

  <th:block th:fragment="content">

기존 HTML 내용



  </th:block>

</th:block>
```

수정폼 만들기

- 전달받은 게시물 정보를 입력필드를 이용하여 출력한다.
- 제목과 내용만 수정할 수 있다.
- 데이터가 DTO로 수집되므로 입력필드에 적절한 name값을 지정한다.
- 수정 버튼을 클릭하면, 수정처리가 호출되고 사용자가 변경한 데이터가 전달된다.

modify.html

```
<label>번호</label>
<input type="text" name="no" th:value="${dto.no}" readonly >
<label>제목</label>
<input type="text" name="title" th:value="${dto.title}" >
```

버튼 이벤트 처리

- 수정 및 삭제를 처리하기 위해, 자바스크립트로 요청을 보낸다.
- 수정 버튼을 클릭하면, 서버에 수정 요청을 보낸다.
- 목록 버튼을 클릭하면, 목록 화면으로 이동한다.
- 삭제 버튼을 클릭하면, 서버에 삭제 요청을 보낸다.

modify.html

```
<script th:inline="javascript">
    $(".modifyBtn").click(function() {
        if(!confirm("수정하시겠습니까?")){
            return ;
        }
    });
```

```
$(".listBtn").click(function() {
    ...
});
```

게시물 수정

수정화면에서 이벤트 처리

테스트

1. 상세화면에서 [수정] 버튼을 클릭한다.

Board Read Page	
번호	1
제목	1번글
내용	내용입니다

수정

2. 수정화면에서 게시물 정보를 변경하고, [수정] 버튼을 클릭한다.

Board Modify Page	
번호	1
제목	1번글
내용	내용을 수정합니다
작성자	관리

수정

게시물 수정

수정화면에서 이벤트 처리

3. 수정여부 확인창에서 [확인] 을 클릭한다.

localhost:8080 내용:
수정하시겠습니까?

확인

취소

4. 상세화면에서 정보가 정상적으로 변경되었는지 확인한다.

Board Read Page

번호

1

제목

1번글

내용

내용을 수정합니다

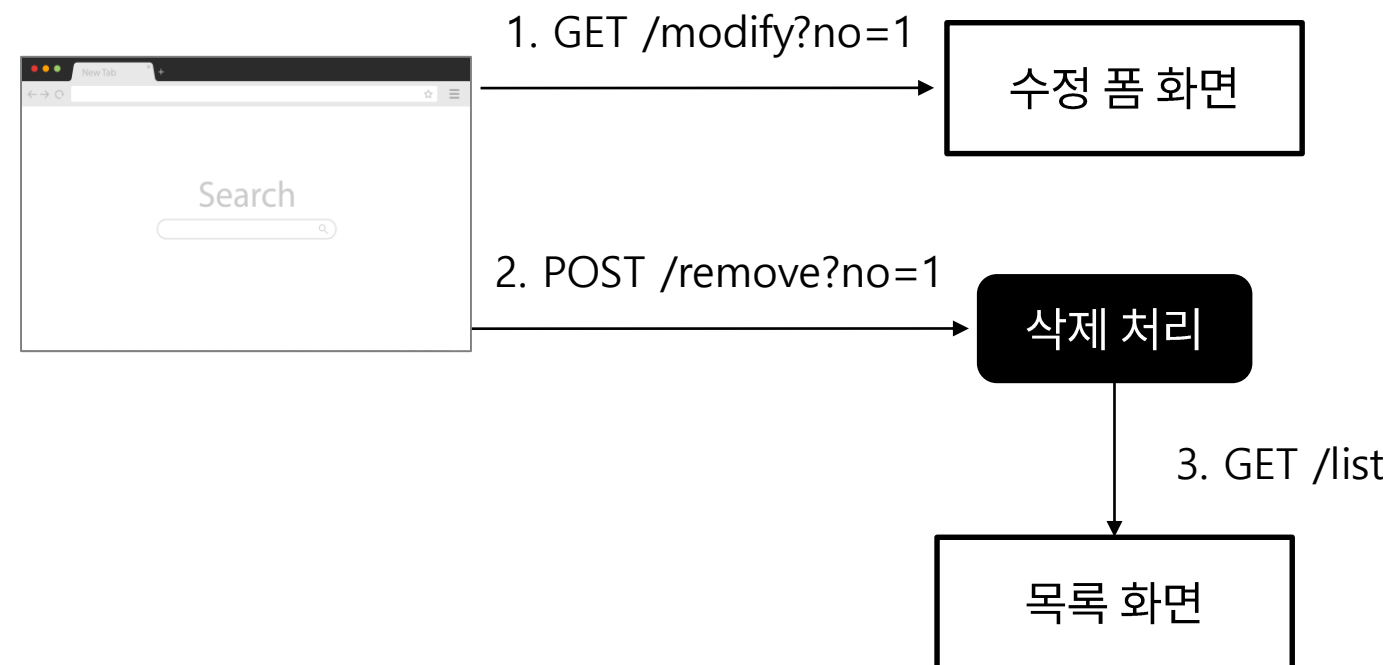
이제 31~32 페이지를 보고
다시 테스트 해보세요!

게시물 삭제

삭제 처리

삭제 기능

- 기존 게시물 정보를 삭제한다.
- 게시물 삭제 후 목록화면으로 이동한다.



Board Modify Page	
번호	1
제목	1번글
내용	내용을 수정합니다
작성자	관리

삭제

Board List Page		게시물 등록
#	제목	
2	2번글	

게시물 삭제

삭제 처리

서비스에서 삭제 처리

- 특정 게시물을 삭제하는 메소드를 추가한다.
- 구현 클래스에서는 상속받은 삭제 메소드를 구현한다.

인터페이스

```
void remove(int no);
```

구현 클래스

```
public void remove(int no) {  
    repository.deleteById(no);  
}
```

게시물 삭제

삭제 처리

서비스의 테스트

- 단위테스트를 통해 특정 게시물을 삭제한다.
- 게시물이 삭제되었는지 확인한다.

단위테스트

```
@Test  
public void 게시물삭제() {  
  
}
```

스스로 진행해보세요~

게시물 테이블

tbl_board ×				
Properties		Data	엔티티 관계도	
tbl_board Enter a SQL expression to filter results (use Ctrl+Space)				
UI	123 no ▾	🕒 mod_date ▾	🕒 reg_date ▾	ABC content

컨트롤러에서 삭제 처리

- POST방식으로 게시물을 삭제하는 메소드를 추가한다.
- 글번호를 파라미터로 수집한다.
- 수집한 글번호로 게시물을 삭제한다.
- 삭제 처리 후 목록화면으로 이동하도록 리다이렉트를 호출한다.

BoardController

```
@PostMapping("/remove")
    public String removePost(int no) {
        service.remove(no);
        return "redirect:/board/list";
    }
```

버튼 이벤트 처리

- 삭제 버튼을 클릭하면, 서버에 게시물 삭제를 요청한다.

modify.html

```
$(".removeBtn").click(function(){  
    form  
        .attr("action", "/board/remove")  
        .attr("method","post")  
        .submit();  
});
```

게시물 삭제

수정화면에서 이벤트 처리

테스트

1. 수정화면에서 [삭제] 버튼을 클릭한다.

Board Modify Page

번호
1

제목
1번글

내용
내용을 수정합니다

작성자
도치

삭제

2. 목록화면에서 게시물이 삭제되었는지 확인한다.

Board List Page 게시물 등록		
#	제목	작성자
2	2번글	도치

이제 31~32 페이지를 보고
다시 테스트 해보세요!

페이징 처리

페이징 처리란?

페이징이란 게시물 목록을 한번에 모두 보여주지 않고, 페이지를 나누어 출력하는 방법이다.
페이징은 정렬 방식, 페이지 크기, 현재 페이지 번호에 따라 출력하는 것이 방식이다.

#	제목	작성자	등록일
30	30번글	둘리	2023/08/03
29	29번글	둘리	2023/08/03
28	28번글	둘리	2023/08/03
27	27번글	둘리	2023/08/03
26	26번글	둘리	2023/08/03
25	25번글	둘리	2023/08/03
24	24번글	둘리	2023/08/03
23	23번글	둘리	2023/08/03
22	22번글	둘리	2023/08/03
21	21번글	둘리	2023/08/03

1

2

3

- Pageable는 페이지 처리에 필요한 정보를 담는 인터페이스이다.
- 객체를 생성할 때는 PageRequest라는 클래스를 사용한다.
- of(int page, int size, Sort sort) 메소드: 페이지번호, 데이터개수, 정렬 관련 정보
- 결과는 Page 타입으로 반환되며, 페이지정보와 게시물 리스트가 담겨있다.
- 아래는 1페이지의 데이터 10개를 가져오는 코드이다.

단위테스트

```
public void 페이지테스트() {
    Pageable pageable = PageRequest.of(0,10);
    Page<Board> result = repository.findAll(pageable);
}
```

SQL

```
b1_0.writer
from
board b1_0 limit ?,
```

반환값

```
result = {PageImpl@11162} "Page 1 of 3 containing com.example.demo.entity.B
  total = 30
  content = {ArrayList@11175} size = 10
    0 = {Board@11178} "Board(no=1, title=1번글, content=안녕하세요)"
    1 = {Board@11179} "Board(no=2, title=2번글, content=안녕하세요)"
    2 = {Board@11180} "Board(no=3, title=3번글, content=안녕하세요)"
    8 = {Board@11186} "Board(no=9, title=9번글, content=안녕하세요)"
    9 = {Board@11187} "Board(no=10, title=10번글, content=안녕하세요)"
  pageable = {PageRequest@11161} "Page request [number: 0, size 10, sor
    sort = {Sort@11208} "UNSORTED"
    page = 0
    size = 10
```

- Sort는 정렬 정보를 담는 클래스이다.
- 특정 필드를 기준으로 순정렬 또는 역정렬 할 수 있다.
- 페이지 처리를 담당하는 PageRequest에 정렬정보를 담는다.

단위테스트

```
public void 정렬조건추가하기() {  
    Sort sort = Sort.by("no").descending();  
    Pageable pageable = PageRequest.of(0,10, sort);  
    Page<Board> result = repository.findAll(pageable);  
}
```

SQL

```
from  
    board b1_0  
order by  
    b1_0.no desc limit ?,
```

쿼리에 order by가 적용됨

- 서비스에서 목록 조회 메소드를 수정한다.
- 매개변수로 페이지번호를 수집하고, Page 객체를 반환한다.
- 페이지번호, 개수, 정렬방식을 담아서 페이지 정보를 생성한다.
- 페이지 정보를 전달하여 게시물 목록을 조회한다.
- 결과값에는 게시물 목록과 페이지정보가 담겨있다.

인터페이스

```
Page<BoardDTO> getList(int pageNumber);
```

구현 클래스

```
@Override
public Page<BoardDTO> getList(int page) {
    Pageable pageable = PageRequest.of(page, 10, Sort.by("no").descending());
    Page<Board> entityPage = repository.findAll(pageable);
    ...
}
```

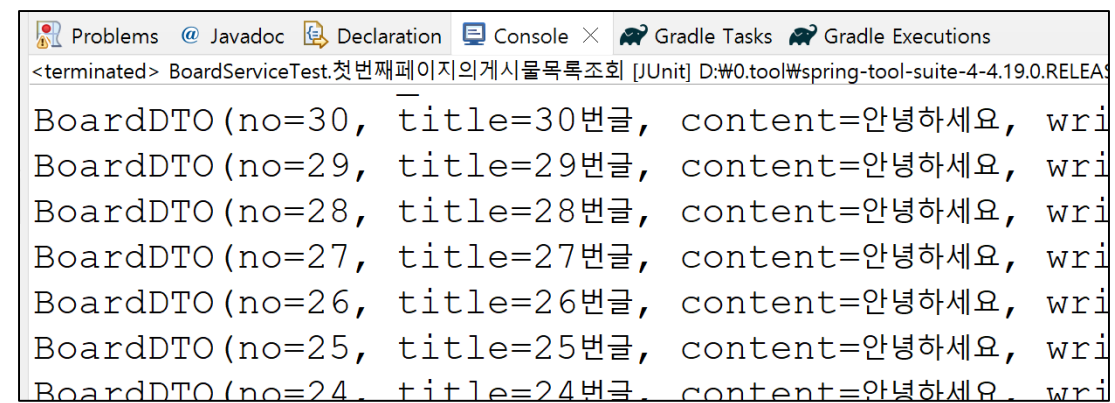
단위테스트를 통해 첫번째 페이지의 게시물 목록을 조회한다.

단위테스트

```
@Test
public void 첫번째페이지_게시물목록조회 () {

}
```

결과 화면



```
Problems @ Javadoc Declaration Console × Gradle Tasks Gradle Executions
<terminated> BoardServiceTest.첫번째페이지의게시물목록조회 [JUnit] D:\W00\tool\spring-tool-suite-4-4.19.0.RELEASE
BoardDTO (no=30, title=30번글, content=안녕하세요, writer=홍길동)
BoardDTO (no=29, title=29번글, content=안녕하세요, writer=홍길동)
BoardDTO (no=28, title=28번글, content=안녕하세요, writer=홍길동)
BoardDTO (no=27, title=27번글, content=안녕하세요, writer=홍길동)
BoardDTO (no=26, title=26번글, content=안녕하세요, writer=홍길동)
BoardDTO (no=25, title=25번글, content=안녕하세요, writer=홍길동)
BoardDTO (no=24, title=24번글, content=안녕하세요, writer=홍길동)
```

컨트롤러에서 목록 페이지 처리

- 컨트롤러에서 목록 조회 메소드를 수정한다.
- 페이지번호를 수집하기 위한 파라미터를 추가한다.
- 페이지번호를 이용하여 게시물 목록을 조회한다.
- 결과 데이터를 화면에 전달한다.
- 결과 데이터에는 게시물 목록과 페이지 처리에 필요한 정보가 담겨있다.

BoardController

```
public void list( int page, Model model ) {  
    Page<DTO> result = service.getList(page);  
    model.addAttribute("result", result);  
}
```

결과 데이터 구성 요소

- 총 페이지 번호 (totalPages)
- 현재 페이지 번호 (number)
- 게시물 리스트

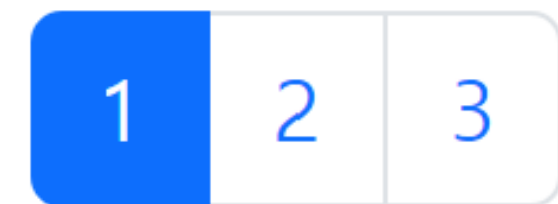
목록화면에서 페이지 처리

- 목록 화면의 하단에 페이지번호를 추가한다.
- 목록 태그를 사용하여, 1부터 3까지 페이지 번호 목록을 생성한다.
- "active" 라는 클래스 속성을 사용하여, 1번 페이지를 현재 페이지로 지정한다.
- CSS를 사용하여 "active" 클래스를 가진 페이지 번호를 파란색으로 표시한다.

list.html

```
44 <!-- 주석 해제: ctrl + shift + \ -->
45
46 <!-- 1. 페이지 번호 목록 생성 -->
47 <!--<ul class="pagination h-100 justify-content:
48     <li class="page-item active">
49     <a class="page-link" href="">1</a>
50     </li>
51     <li class="page-item">
52     <a class="page-link" href="">2</a>
53     </li>
```

실습 코드에서 이 부분을 복사하세요!
static_html 폴더 아래 list.html 파일
line: 44 ~ 100



페이징 처리

컨트롤러와 화면에서 페이징 처리

테스트

1. localhost:8080/board/list 주소로 접속한다.
2. 목록화면에 1부터 3까지 페이지 번호가 나타나는지 확인한다.
3. 현재 페이지가 표시되는지 확인한다.

결과 화면

Board List Page 게시물 등록			
#	제목	작성자	등록일
30	30번글	둘리	2023/08/06
29	29번글	둘리	2023/08/06
28	28번글	둘리	2023/08/06
27	27번글	둘리	2023/08/06
26	26번글	둘리	2023/08/06
25	25번글	둘리	2023/08/06
24	24번글	둘리	2023/08/06
23	23번글	둘리	2023/08/06
22	22번글	둘리	2023/08/06
21	21번글	둘리	2023/08/06

1

2

3

목록화면에서 페이지 처리

- 전달받은 페이지 정보를 사용하여 번호 목록을 출력한다.
- 숫자 객체의 `sequence()` 함수로 1부터 마지막 페이지 번호까지 반복한다. (1,2,3)

```
<th:block th:each="page : ${#numbers.sequence(1, list.totalPages)}">
    [[${page}]]
</th:block>
```



- 현재 페이지 여부를 확인해서 'active' 클래스를 추가한다.

```
<th:block th:if="${list.number+1 == page}">
    <li class="page-item active">
        <a class="page-link">[[${page}]]</a>
    </li>
</th:block>
```

현재 페이지인 경우



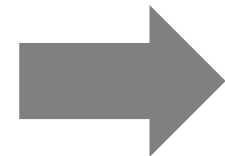
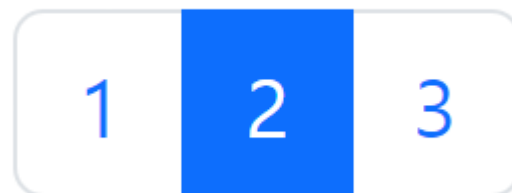
그렇지 않은 경우



페이지 번호 링크 처리

- 페이지 번호를 클릭하면, 해당 페이지로 이동한다.
- 페이지 번호를 파라미터로 전달한다.
- 리스트 링크를 생성하고, 페이지번호를 파라미터로 전달한다.

```
<a th:href="@{/board/list (page = ${page}) }"> [[${page}]] </a>
```



localhost:8080/board/list?page=2

Board List Page

[게시물 등록](#)

#	제목	작성자
20	20번글	둘리
19	19번글	둘리
18	18번글	둘리
17	17번글	둘리

페이징 처리

컨트롤러와 화면에서 페이징 처리

테스트

1. localhost:8080/board/list 주소로 접속한다.
2. 글번호를 기준으로 역정렬되었는지 확인한다.
3. 게시물이 10개씩만 출력되는지 확인한다.
4. 페이지번호를 클릭하면 해당페이지로 이동하는지 확인한다.
5. 현재 페이지 번호가 표시되는지 확인한다.

결과 화면

#	제목	작성자	등록일
30	30번글	둘리	2023/08/03
29	29번글	둘리	2023/08/03
28	28번글	둘리	2023/08/03
27	27번글	둘리	2023/08/03
26	26번글	둘리	2023/08/03
25	25번글	둘리	2023/08/03
24	24번글	둘리	2023/08/03
23	23번글	둘리	2023/08/03
22	22번글	둘리	2023/08/03
21	21번글	둘리	2023/08/03
<div>1 2 3</div>			

페이징 처리

상세화면에서 페이징 처리

페이지 번호 유지하기

- 목록페이지에서 상세페이지로 이동할 때 현재 페이지번호를 유지해야 한다.
- 이를 위해 상세화면으로 이동할 때, 페이지번호를 전달한다.

목록화면

Board List Page 게시물 등록		
#	제목	작성자
10	10번글	둘리
9	9번글	둘리
8	8번글	둘리
7	7번글	둘리
6	6번글	둘리

1

2

3

상세화면

Board Read Page	
번호	10
제목	10번글
내용	안녕하세요

3

목록으로

목록화면

Board List Page 게시물 등록		
#	제목	작성자
10	10번글	둘리
9	9번글	둘리
8	8번글	둘리
7	7번글	둘리
6	6번글	둘리

1

2

3

컨트롤러에서 상세 조회 페이지 처리

- 컨트롤러에서 게시물 상세 조회 메소드를 수정한다.
- 페이지번호를 수집하기 위한 파라미터를 추가한다.
- 상세화면에 그대로 페이지번호를 전달한다.

BoardController

```
@GetMapping("/read")
public void read(int no, @RequestParam(defaultValue = "0") int page, Model model) {
    DTO dto = service.read(no);
    model.addAttribute("dto", dto);
    model.addAttribute("page", page);
}
```

목록화면에서 페이지 처리

- 상세 화면으로 이동하는 링크를 수정한다.
- 현재 페이지번호를 파라미터로 추가한다.

list.html

```
<a th:href="@{/board/read(no = ${dto.no}, page= ${list.number + 1})}">
[[${dto.no}]]
</a>
```

↓

```
<a href="/board/read?no=10&page=3">
    10
</a>
```

상세화면에서 페이지 처리

- 목록으로 이동하는 링크를 변경한다
- 현재 페이지번호를 파라미터로 다시 전달한다

read.html

```
<a th:href="@{/board/list ( page = ${page} ) }">  
    <button type="button">목록으로</button>  
</a>
```

```
<a href="/board/list?page=3">  
    <button type="button" class="btn btn-info">목록으로</button>  
</a>
```

페이징 처리

상세화면에서 페이징 처리

테스트

1. localhost:8080/board/list 주소로 접속한다.
2. 3번 페이지로 이동한다.
3. 10번 게시물의 상세화면으로 이동한다.
4. 상세화면에서 "목록으로" 버튼을 클릭한다.
5. 목록화면에서 현재 페이지가 그대로 3번인지 확인한다.



JPA의 쿼리 메소드와 @Query 는 많은 기능을 구현할 수 있지만, 고정된 형태의 조건만 사용할 수 있다.
 다양한 조합으로 검색조건을 만들어야 할 때는 querydsl을 사용하면 된다.
 querydsl는 여러 조건을 조합하여 동적으로 쿼리를 생성하는 기능이다.

검색범위	<input checked="" type="radio"/> 전체 <input type="radio"/> 훈련 기관/과정 검색
검색어	<input type="text" value="나에게 맞는 훈련과정(기관)을 찾아보세요."/>
지역	<input type="button" value="지역선택"/> <input type="button" value="전국 > 전체 x"/> <input type="checkbox"/> 원격훈련포함
NCS 직종	<input type="button" value="직종선택"/>
훈련유형	<input type="button" value="훈련유형 선택"/> <input type="button" value="전체 x"/>
개강일자	<input type="text" value="2024-09-26"/> ~ <input type="text" value="2025-09-26"/> <input type="button" value="초기화"/> <input type="button" value="1개월"/> <input type="button" value="3개월"/> <input type="button" value="1년"/>
주말여부	<input checked="" type="radio"/> 전체 <input type="radio"/> 주중반 <input type="radio"/> 주말반 <input type="radio"/> 주중+주말반
주야구분	<input checked="" type="radio"/> 전체 <input type="radio"/> 주간 <input type="radio"/> 야간 <input type="radio"/> 혼합
훈련구분	<input checked="" type="radio"/> 전체 <input type="radio"/> 일반과정 <input type="radio"/> 인터넷과정 <input type="radio"/> 혼합과정 <input type="radio"/> 스마트혼합훈련
자비부담금	<input checked="" type="radio"/> 전체 <input type="radio"/> 자비부담금있음 <input type="radio"/> 자비부담금없음
140시간 이상 여부	<input checked="" type="radio"/> 전체 <input type="radio"/> 140시간 이상 <input type="radio"/> 140시간 미만
직업능력개발 훈련시설여부	<input type="checkbox"/> 직업능력개발훈련시설여부

예시: HRD 훈련 기관 검색 기능

다음과 같은 상황을 처리할 수 있다.

- '검색어'와 같이 하나의 항목으로 검색하는 경우
- '검색어 + 지역'과 같이 두 가지 항목으로 검색하는 경우
- '지역 + NCS 직종 + 개강 일자'와 같이 세 가지 항목으로 검색하는 경우

동적 쿼리를 작성하기 위해 엔티티 클래스에 대응하여 **Q도메인**이 생성된다.
Q도메인은 엔티티 클래스를 기반으로 생성된다.

```
/**
 * QBoard is a Querydsl query type for Board
 */
@Generated("com.querydsl.codegen.DefaultEntitySerializer")
public class QBoard extends EntityPathBase<Board> {

    private static final long serialVersionUID = -164971147L;

    public static final QBoard board = new QBoard("board");

    public final QBaseEntity _super = new QBaseEntity(this);
```

build.gradle 파일에 아래 내용을 추가한다.

```
// querydsl 라이브러리 추가
implementation 'com.querydsl:querydsl-jpa:5.0.0:jakarta'
annotationProcessor "com.querydsl:querydsl-
apt:${dependencyManagement.importedProperties['querydsl.version']}:jakarta"
annotationProcessor "jakarta.annotation:jakarta.annotation-api"
annotationProcessor "jakarta.persistence:jakarta.persistence-api"

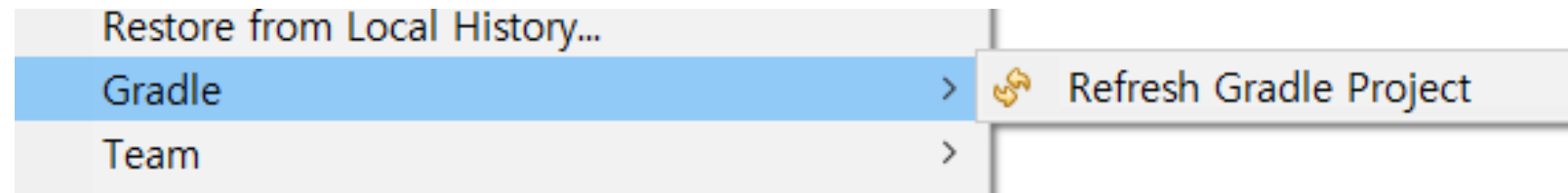
// querydsl 관련 설정 추가
def querydslDir = "src/main/generated"

sourceSets {
    main.java.srcDirs += [ querydslDir ]
}

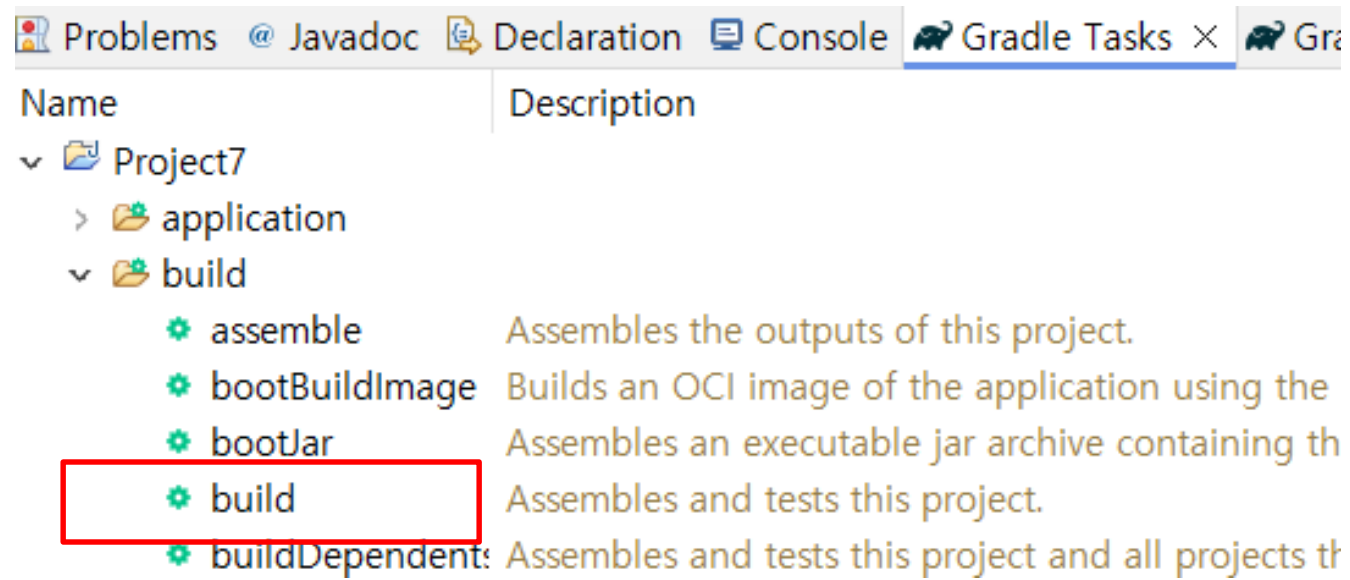
tasks.withType(JavaCompile) {
    options.getGeneratedSourceOutputDirectory().set(file(querydslDir))
}

clean.doLast {
    file(querydslDir).deleteDir()
}
```

Gradle의 Refresh 메뉴를 실행하여 변경한 내용을 업데이트 한다.



Task 메뉴에서 build를 실행한다.



task를 실행하면 generated 폴더 아래 Q도메인 클래스가 생성된다.

```
▼ src/main/generated
  ▼ com.example.demo.entity
    > QBaseEntity.java
    > QBoard.java
```

```
/**
 * QBoard is a Querydsl query type for Board
 */
@Generated("com.querydsl.codegen.DefaultEntitySerializer")
public class QBoard extends EntityPathBase<Board> {

    private static final long serialVersionUID = -164971147L;

    public static final QBoard board = new QBoard("board");

    public final QBaseEntity _super = new QBaseEntity(this);
```

BoardRepository에서 querydsl 메소드를 사용하기 위해 QuerydslPredicateExecutor를 추가로 상속받는다.

```
interface BoardRepository extends JpaRepository<Board, Integer>,
    QuerydslPredicateExecutor<Board>
```

새로운 리파지토리 단위 테스트를 생성하고, Q도메인을 사용하여 검색 기능을 테스트한다.

```
@Test
void 다중항목검색테스트1 () {

    Pageable pageable = PageRequest.of(0, 10);
    QBoard qBoard = QBoard.board;
    BooleanBuilder builder = new BooleanBuilder();

    BooleanExpression expression = qBoard.content.contains("안녕");
    builder.and(expression);
    BooleanExpression expression2 = qBoard.writer.contains("둘리");

    builder.and(expression2);
    Page<Board> result = repository.findAll(builder, pageable);
    List<Board> list = result.getContent();
}
```

이 예제에서는 검색 화면까지는 다루지 않습니다.

검색 기능은 프로젝트 시간에 Querydsl을 활용하여 구현해 보세요.