



It's Your Life

with





MyBatis 로 데이터 가져오기!



mapper

인터페이스 설정

```
@Mapper 5 usages kdtTetz *  
public interface MemberMapper {  
    List<MemberDto> findAll();  
}
```

인터페이스 이므로 메서드를
구현할 필요가 없습니다!

사용하고자하는 메서드 명만 적으시면 됩니다!

실제 코드는 MyBatis 가 채웁니다!



mapper

xml 파일 생성



```
@Mapper 5 usages kdtTetz *  
public interface MemberMapper {  
    List<MemberDto> findAll();  
}
```



```
<mapper namespace="org.example.mapper.MemberMapper">  
    <select id="findAll" resultType="MemberDto">  
        select id, name  
        from members;  
    </select>  
</mapper>
```



DTO 객체 등록



```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE configuration
  PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
  "http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
  <typeAliases>
    <typeAlias alias="MemberDto" type="org.example.dto.member.MemberDto"/>
  </typeAliases>
</configuration>
```

타입에 대한 별명을 설정하고
해당 별명이 어느 객체를 참조하는지 지정!



MyBatis 를 사용한 데이터 계층

@Repository kdtTetz *

```
public class MemberRepository {  
    private final MemberMapper memberMapper; 2 usages
```

@Autowired kdtTetz *

```
public MemberRepository(MemberMapper memberMapper) {  
    this.memberMapper = memberMapper;  
}
```

```
public List<MemberDto> findAll() { return memberMapper.findAll(); }
```

```
}
```

@Autowired 어노테이션을 사용하여
@Mapper 로 등록 된
데이터 Mapper 를 자동으로 주입!

Mapper 인터페이스에 있는
findAll() 메서드를 사용하여
원하는 데이터를 DB 부터 받아서
가져옵니다!



MyBatis 를 사용한 컨트롤러!



```
@Controller  👤 kdtTetz *
@Slf4j
@RequestMapping(🌐"/member/v4")
public class MemberControllerV4 {
    private final MemberRepository memberRepository;

    @Autowired  👤 kdtTetz *
    public MemberControllerV4(MemberRepository memberRepository) {
        this.memberRepository = memberRepository;
    }

    @GetMapping(🌐"/show")  👤 kdtTetz *
    public String memberList(Model model) {
        log.info("=====> 회원 조회 페이지 호출, /member/list");

        model.addAttribute(attributeName: "memberList", memberRepository.findAll());
        return "member-show4";
    }
}
```

@Autowired 어노테이션을 사용하여
@Repository 에 등록된
MemberRepository 를 자동 주입

MemberRepository 의
findAll() 메서드로
DB 의 데이터를 받아서 전달!



View

페이지 추가



```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<!DOCTYPE html>
<html>
<head>
    <title>Member List</title>
</head>
<body>
<%@ include file="header.jsp"%>
<h1>회원 목록 V4</h1>
<ul>
    <li><b>ID - Name</b></li>
    <c:forEach var="member" items="${memberList}">
        <li>${member.id} - ${member.name}</li>
    </c:forEach>
</ul>
</body>
<html>
```

V1

[HOME](#) [회원 등록](#) [회원 목록](#) [TODO](#)

V2

[회원 등록 V2](#) [회원 목록 V2](#) [TODO](#)

V3

[회원 등록 V3](#) [회원 목록 V3](#) [TODO](#)

V4

[회원 등록 V4](#) [회원 목록 V4](#) [TODO](#)

회원 목록 V4

- ID - Name
- siwan - 김시완
- tetz - 이효석





시작해볼까





MyBatis 를

활용한 게시판 만들기



일단 DB 부터

만듭시다!



<https://github.com/xenosign/spring-code-repo/blob/main/posts.sql>

```
1 • USE mybatis;
2
3 • CREATE TABLE posts (
4     id INT PRIMARY KEY AUTO_INCREMENT,
5     title VARCHAR(255) NOT NULL,
6     content TEXT NOT NULL
7 );
8
9 • INSERT INTO posts (title, content) VALUES
10 ('첫 번째 게시물', '이것은 첫 번째 게시물의 내용입니다. a'),
11 ('두 번째 게시물', '이것은 두 번째 게시물의 내용입니다. b'),
12 ('세 번째 게시물', '이것은 세 번째 게시물의 내용입니다. c'),
13 ('네 번째 게시물', '이것은 네 번째 게시물의 내용입니다. d'),
14 ('다섯 번째 게시물', '이것은 다섯 번째 게시물의 내용입니다. e');
15
16 • SELECT * FROM posts;
```



그란데 말입니다

그란데 말입니다
그럼 이제 뭐부터 해야할까요!?

1. mapper 인터페이스
2. MyBatis Xml 파일
3. Dto 객체
4. 컨트롤러 작업
5. Repository 만들기

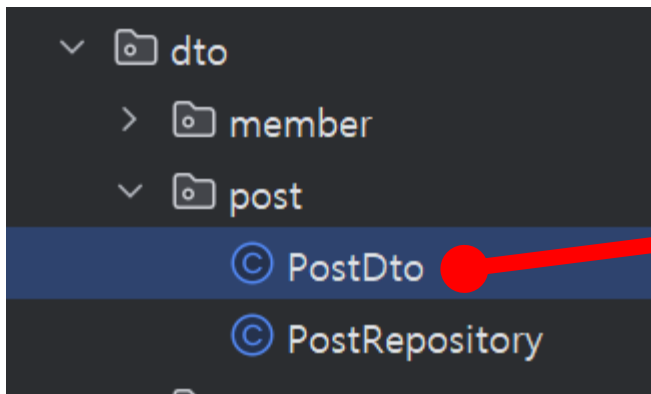






PostDto

만들기



가장 먼저 DB 에서 받아온 데이터를
자바에서 사용할 수 있도록
Dto 객체 부터 만들어 줍시다!

id	title	content
1	첫 번째 게시물	이것은 첫 번째 게시물의 내용입니다. a
2	두 번째 게시물	이것은 두 번째 게시물의 내용입니다. b
3	세 번째 게시물	이것은 세 번째 게시물의 내용입니다. c
4	네 번째 게시물	이것은 네 번째 게시물의 내용입니다. d
5	다섯 번째 게시물	이것은 다섯 번째 게시물의 내용입니다. e
	NULL	NULL

우리가 만들어야 할 테이블!



```
@Data 10 usages kdtTetz
@AllArgsConstructor
public class PostDto {
    private int id;
    private String title;
    private String content;
}
```

각각의 컬럼에 대응하는
id, title, content 멤버 선언!

롬복 어노테이션으로
생성자, Getter, Setter 생성



그란데 말입니다

그란데 말입니다
그럼 이제 뭐부터 해야할까요!?

1. mapper 인터페이스
2. MyBatis Xml 파일
3. 컨트롤러 작업
4. Repository 만들기

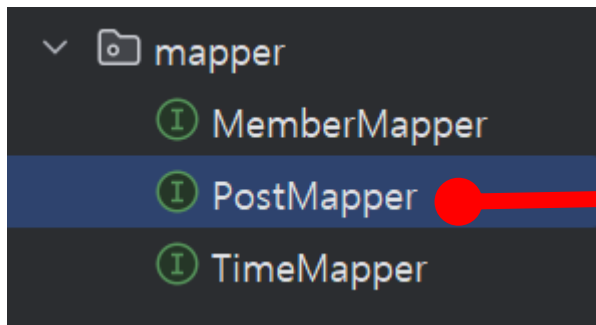






Mapper

인터페이스 만들기



실제 SQL 을 실행하는 메서드가 되는
Mapper 인터페이스를 만듭시다!

여기서 기본 틀을 짤 다음
나머지 작업을 이어갑시다!





```
@Mapper
public interface PostMapper {
    List<PostDto> findAll();
}
```

@Mapper 어노테이션으로
Mapper 를 스프링 빈으로 등록!

일단은 모든 목록을 받아오는
findAll() 추상 메서드만 선언



그란데 말입니다

그란데 말입니다
그럼 이제 뭐부터 해야할까요!?

1. MyBatis Xml 파일
2. 컨트롤러 작업
3. Repository 만들기





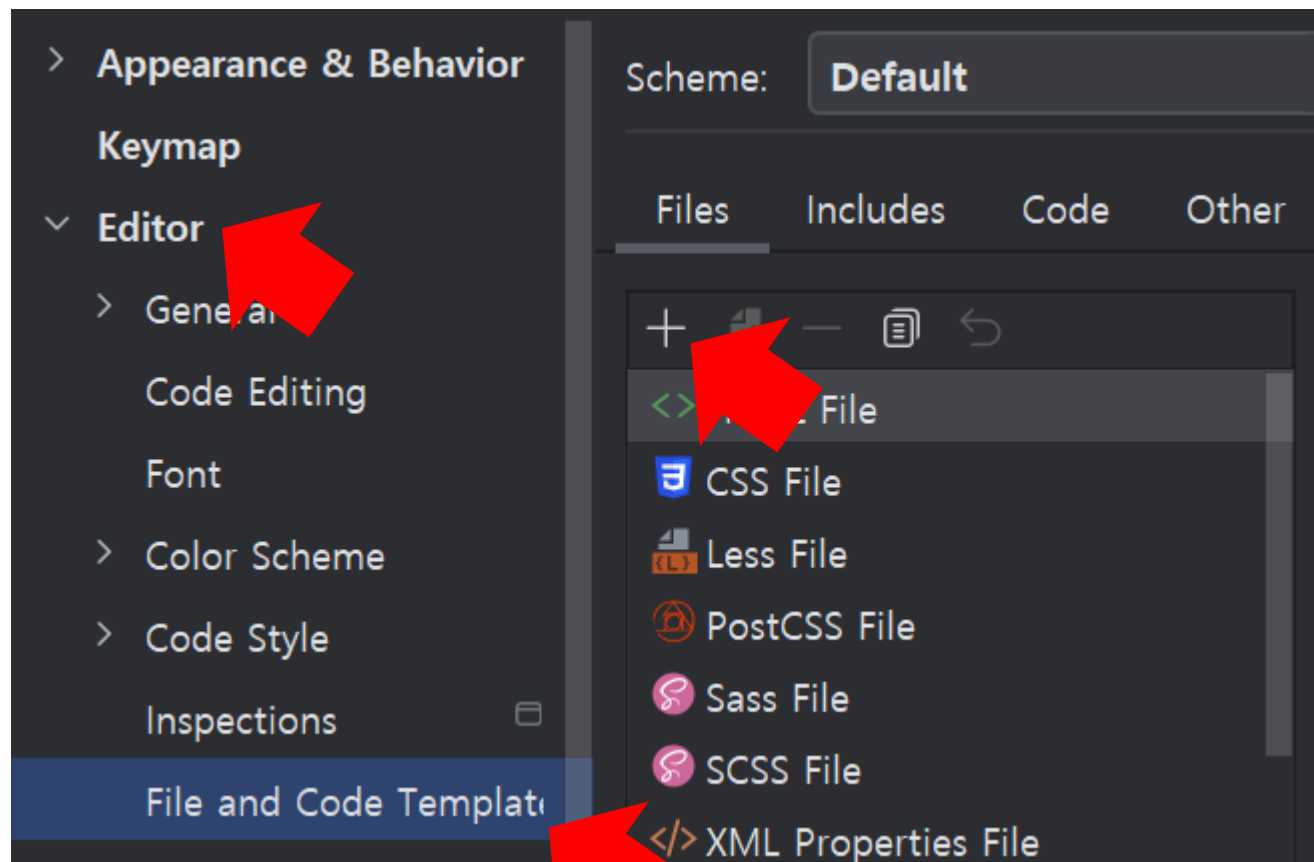
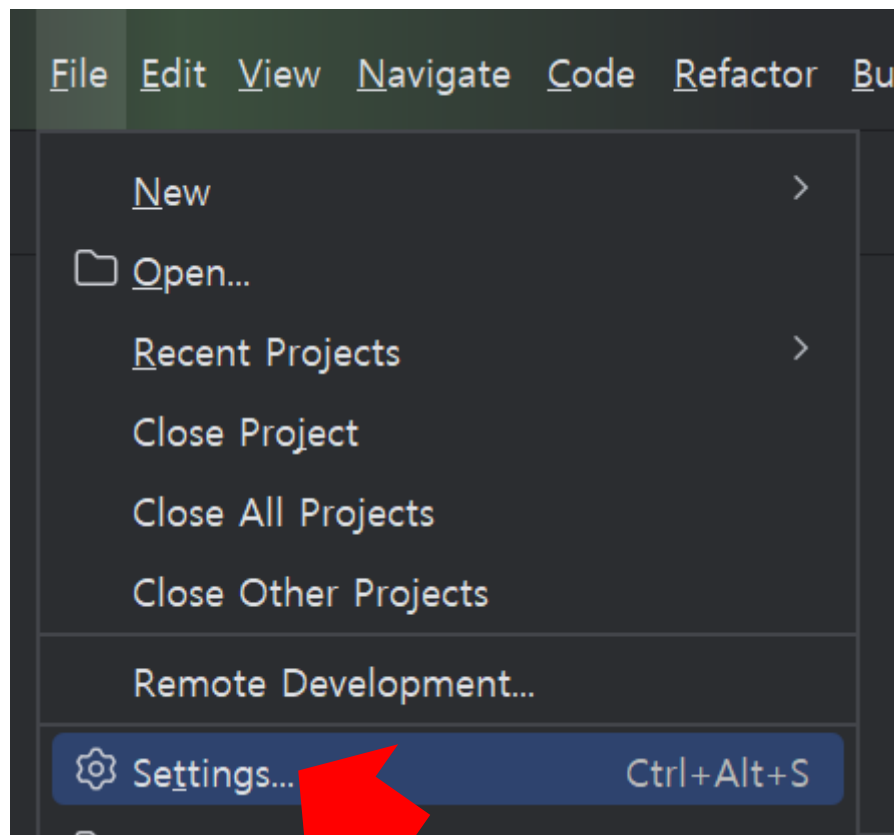


Mapper.xml

템플릿 등록


```
<?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="org.example.mapper.PostMapper">
    <select id="findAll" resultType="PostDto">
        select id, title, content
        from posts;
    </select>
</mapper>
```

Mapper 의 이런 코드를
외워서 쓸 순 없겠쥬?





Name: Extension:

File name:

```
<?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="${PACKAGE_NAME}.${NAME}">
</mapper>
```

파일명이 바로 Mapper 인터페이스를
참조할 수 있도록 파일명을 여기에 반영!

자신이 속한 폴더 명을 그대로
패키지 명에 적용시켜서 Mapper 인터페이스의
패키지를 찾아 갈 수 있도록 세팅!



PostMapper.xml

작성하기!

> kbsp
map
M
P
Ti
resources
org
example
map
M
P
Ti
application.
log4j.xml
log4jdbc.log

New

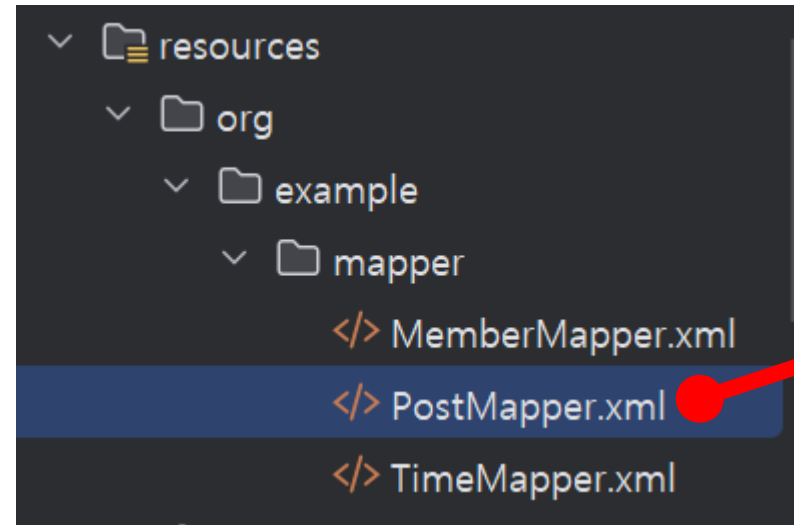
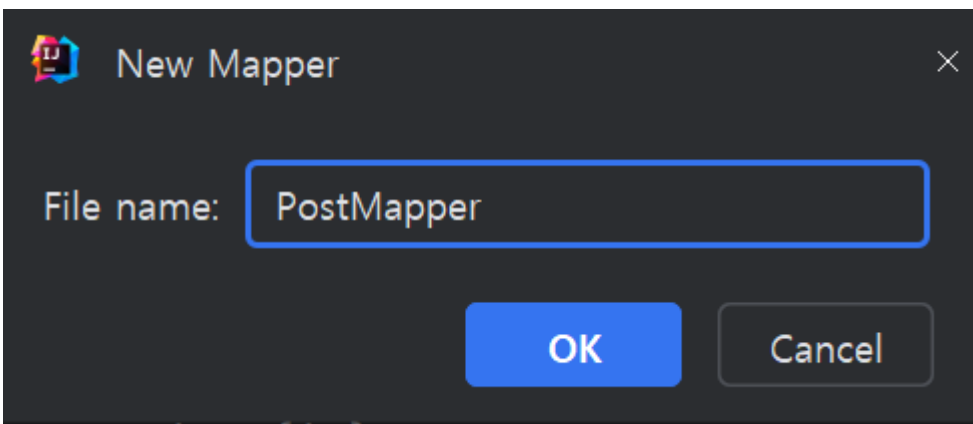
Cut Ctrl+X
Copy Ctrl+C
Copy Path/Reference...
Paste Ctrl+V
Find Usages Alt+F7
Find in Files... Ctrl+Shift+F
Replace in Files... Ctrl+Shift+R
Analyze >
Refactor >
Bookmarks >
Reformat Code Ctrl+Alt+L
Optimize Imports Ctrl+Alt+O
Delete... Delete

File

Scratch File Ctrl+Alt+Shift+Insert
Directory
JavaScript File
TypeScript File
HTML File
Stylesheet
package.json
Dockerfile
API HTTP Request
OpenAPI Specification
Kubernetes Resource
Helm Chart
Servlet
Mapper

Mapper 를 선택!





Mapper 인터페이스와 동일한
패키지 폴더 구조를 가지는 곳에

Mapper 인터페이스와 동일한
이름의 xml 파일을 만들어 줍시다!



```
<?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="org.example.mapper.PostMapper">
</mapper>
```



템플릿 기능으로 인해서
자동으로 패키지와 Mapper 클래스
위치가 생성!



```
<mapper namespace="org.example.mapper.PostMapper">
  <select id="findAll" resultType="PostDto">
    select id, title, content
    from posts;
  </select>
</mapper>
```

인터페이스에 유일하게 등록 된
메서드인 findAll 의 SQL 쿼리를 작성!

PostDto 등록!



</> mybatis-config.xml

webapp

WEB-INF

views

member

post

JSP post-new.jsp

JSP post-show.jsp

JSP post-update.jsp

todo

```
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     <typeAliases>
7         <typeAlias alias="MemberDto" type="org.example.dto.member.MemberDto"/>
8         <typeAlias alias="PostDto" type="org.example.dto.post.PostDto"/>
9     </typeAliases>
10 </configuration>
```

MyBatis 는 현재 PostDto 클래스의
위치를 모르는 상태이므로
PostDto 의 위치를 등록 시켜 줍니다!



그란데 말입니다

그란데 말입니다
그럼 이제 뭐부터 해야할까요!?

1. 컨트롤러 작업

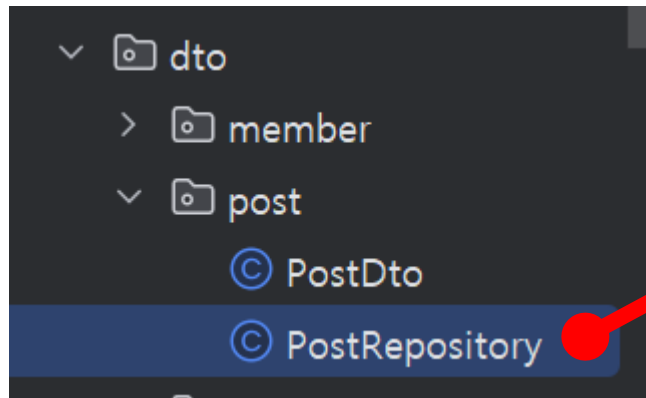






PostRepository

작성하기!



실제적으로 PostDto 객체를
관리하고 저장하는 PostRepository
Repo 객체를 생성

스프링 빈에 저장소의 기능으로 등록!



```
@Repository kdtTetz *  
@RequiredArgsConstructor  
public class PostRepository {  
    private final PostMapper postMapper;  
  
    public List<PostDto> findAll() { 1 usage  
        return postMapper.findAll();  
    }  
}
```

final 이 붙은 멤버 변수에 대한
생성자를 만들어 주는
@RequiredArgsConstructor 를
사용하여 자동으로 생성자 주입 및

@Mapper 로 스프링 빈에 등록된
PostMapper 가져와서 의존성 주입!



그란데 말입니다

그란데 말입니다
멤버 변수 + 의존성 주입 객체에
final 이 붙으면 어떤 점이 좋을까요!?



1. 불변성 보장

→ 객체 생성 이후 값 변경이 불가능, 코드 안전성이 증대

2. 의존성 주입 오류 방지

→ final 필드가 붙으면 초기화가 안되었을 경우 컴파일 에러가 발생하므로 초기화 없이 사용되는 문제 해결!

3. 생성자 주입이 강제

→ final 필드는 초기화가 필요하므로 반드시 생성자를 만들어줘야 하므로 스프링에 의해 자동으로 의존성 주입이 발생하여 안정성이 증대


```
@Repository  👤 kdtTetz *  
@RequiredArgsConstructor  
public class PostRepository {  
    private final PostMapper postMapper;  
  
    public List<PostDto> findAll() {  
        return postMapper.findAll();  
    }  
}
```

전체 목록을 가져오는
findAll() 메서드를 만들고
실제 기능은 MyBatis 가 등록해 놓은
postMapper 의 findAll() 을
가져와서 사용!





그란데 말입니다

그란데 말입니다
그럼 이제 뭐부터 해야할까요!?

1. 컨트롤러 작업



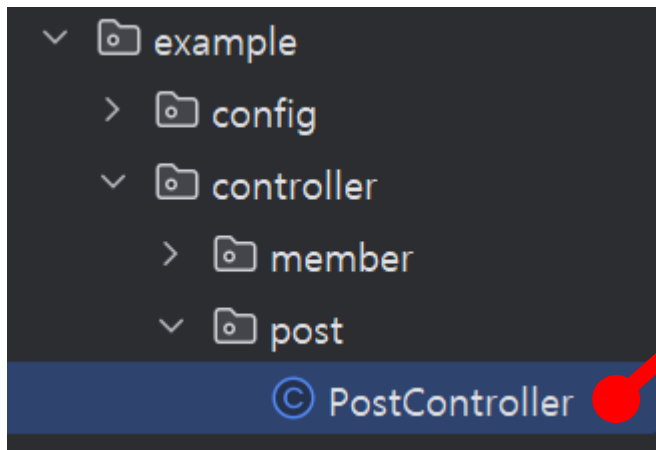


PostController

작성하기!



게시판에 관련된 모든 기능을
담당할 PostController 생성





컨트롤러에 필요한 어노테이션 추가!

RequestMapping 을 사용하여
특정 컨텍스트로 오는 모든 요청 받기

```
@Controller  👤 kdtTetz *
@Slf4j
@RequiredArgsConstructor
@RequestMapping(🌐 "/post/v1")
public class PostController {
    private final PostRepository postRepository;
    private String context = "/post"; 1 usage

    // 게시글 목록
    @GetMapping(🌐 "/show")  👤 kdtTetz
    public String postList(HttpServletRequest request, Model model) {
        log.info("=====> 게시글 목록 페이지 호출, " + request.getRequestURI());

        model.addAttribute(attributeName: "postList", postRepository.findAll());
        return context + "/post-show";
    }
}
```

```
@Controller  👤 kdtTetz *
@Slf4j
@RequiredArgsConstructor
@RequestMapping(🌐 "/post/v1")
public class PostController {
    private final PostRepository postRepository;
    private String context = "/post"; // usage

    // 게시글 목록
    @GetMapping(🌐 "/show")  👤 kdtTetz
    public String postList(HttpServletRequest request, Model model) {
        log.info("=====> 게시글 목록 페이지 호출, " + request.getRequestURI());

        model.addAttribute(attributeName: "postList", postRepository.findAll());
        return context + "/post-show";
    }
}
```

final 멤버로 안정성 증대!

View 페이지 폴더도 정리할 것이므로
폴더 context 설정



```
@Controller  👤 kdtTetz *
@Slf4j
@RequiredArgsConstructor
@RequestMapping(🌐 "/post/v1")
public class PostController {
    private final PostRepository postRepository;
    private String context = "/post"; 1 usage

    // 게시글 목록
    @GetMapping(🌐 "/show") • kdtTetz
    public String postList(HttpServletRequest request, Model model) {
        log.info("=====> 게시글 목록 페이지 호출, " + request.getRequestURI());

        model.addAttribute(attributeName: "postList", postRepository.findAll());
        return context + "/post-show";
    }
}
```

/show GET 요청에 매핑



```
@Controller  👤 kdtTetz *
@Slf4j
@RequiredArgsConstructor
@RequestMapping(🌐 "/post/v1")
public class PostController {
    private final PostRepository postRepository;
    private String context = "/post"; 1 usage

    // 게시물 목록
    @GetMapping(🌐 "/show")  👤 kdtTetz
    public String postList(HttpServletRequest request, Model model) {
        log.info("=====> 게시물 목록 페이지 호출, " + request.getRequestURI());

        model.addAttribute(attributeName: "postList", postRepository.findAll());
        return context + "/post-show";
    }
}
```

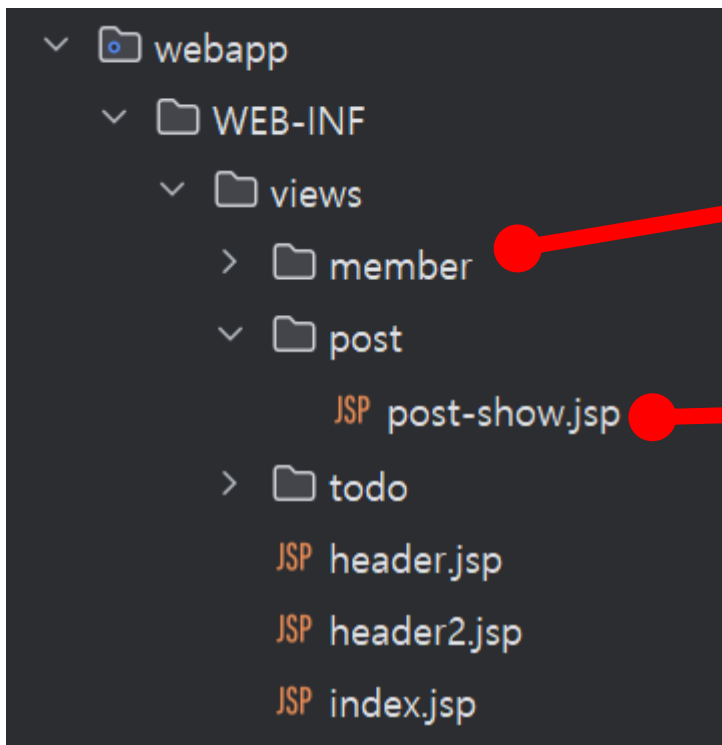
MyBatis 의 findAll() 로
모든 게시글을 받아오기

DB 로 부터 받은 데이터를
/post-show.jsp 로 전달!



post-show.jsp

View 페이지 작성



먼저 지저분한 폴더를 정리해 줍시다!



post-show.jsp 뷰 페이지를 추가!



<https://github.com/xenosign/spring-code-repo/blob/main/post-show.jsp>

View 파트는
코드를 복붙해서 사용 합시다!

어차피 프로젝트에서는
Vue 또는 다른 프론트 프레임워크
사용을 하게 될 예정!

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Post List</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      margin: 20px;
      padding: 0;
    }

    h1 {
      color: #333;
      text-align: center;
      margin-bottom: 20px;
    }
  </style>
</head>
<body>
  <h1>Post List</h1>
</body>
</html>
```



<https://github.com/xenosign/spring-code-repo/blob/main/header2.jsp>

기존 메뉴가 지저분 하므로
새로운 header2.jsp 생성하여 적용

webapp
WEB-INF
views
member
post
JSP post-show.jsp
todo
JSP header.jsp
JSP header2.jsp
JSP index.jsp

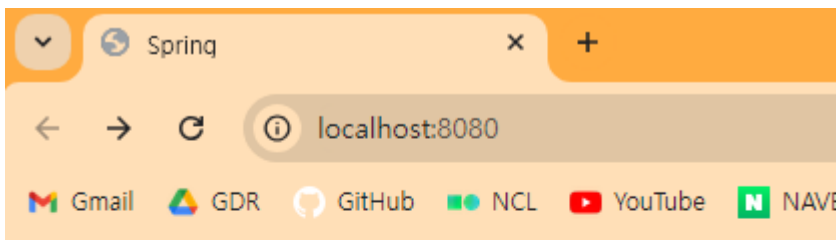
```
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<header>
  <h3>V1</h3>
  <a href="/">HOME</a>
  <a href="/post/v1/show">게시글 목록</a>
</header>
```



확인할 시간!!

(잔뜩 기대중)





V1

[HOME](#) [게시글 목록](#)

Hello, Spring World!

V1

[HOME](#) [게시글 목록](#)

글 목록

제목 검색

제목에서 찾을 단어 입력

/

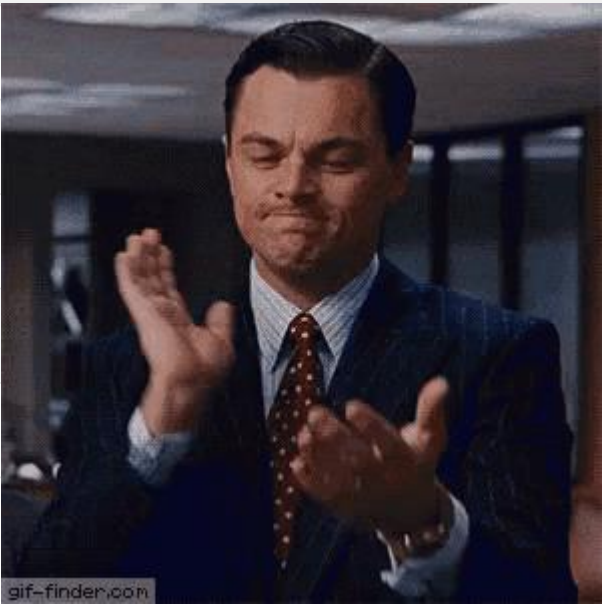
내용 검색

내용에서 찾을 단어 입력

검색

ID	Title	Content	Actions
1	첫 번째 게시물	이것은 첫 번째 게시물의 내용입니다. a	<div>수정삭제</div>
2	두 번째 게시물	이것은 두 번째 게시물의 내용입니다. b	<div>수정삭제</div>
3	세 번째 게시물	이것은 세 번째 게시물의 내용입니다. c	<div>수정삭제</div>
4	네 번째 게시물	이것은 네 번째 게시물의 내용입니다. d	<div>수정삭제</div>
5	다섯 번째 게시물	이것은 다섯 번째 게시물의 내용입니다. e	<div>수정삭제</div>

새글 작성하기





게시글

삭제 기능 추가



그란데 말입니다

그란데 말입니다
그럼 이제 뭐부터 해야할까요!?

1. mapper 인터페이스
2. MyBatis Xml 파일
3. 컨트롤러 작업
4. Repository 만들기





인터페이스에 삭제 메서드 추가

```
@Mapper 4 usages kdtTetz *  
public interface PostMapper {  
    List<PostDto> findAll(); 1 usage  
    int delete(@Param("id") Long id);  
}
```

삭제 기능을 할 delete 추상 메서드 추가!

어떤 게시글을 지울지 선택을 해야하므로
해당 게시글의 id 를 전달해야 합니다!

이때 MyBatis 의 xml 에 해당 값을 전달해야 하므로
@Param 어노테이션으로 매개변수 이름을 지정해 줍니다!



그란데 말입니다

그란데 말입니다

delete 메서드의 타입은 왜 int 일까요!?





PostMapper.xml

삭제 쿼리 추가 &

매개변수 사용하기!



```
<mapper namespace="org.example.mapper.PostMapper">
  <select id="findAll" resultType="PostDto">
    select id, title, content
    from posts;
  </select>

  <delete id="delete">
    delete from posts
    where id=#{id}
  </delete>
</mapper>
```

delete 쿼리를 쓸 것이므로 <delete> 로 시작!
인터페이스 Mapper 에 등록된 메서드 명을 id 에 부여!

인터페이스에서
@Param("id") 로 전달한 매개변수는
#{ } 를 이용하여 값을 꺼내서 사용!



리포지토리에 삭제 메서드 추가



```
@Repository  kdtTetz *
@RequiredArgsConstructor
public class PostRepository {
    private final PostMapper postMapper;

    public List<PostDto> findAll() { 1 usage
        return postMapper.findAll();
    }

    public int delete(Long id) { no usages
        return postMapper.delete(id);
    }
}
```

인터페이스에 선언된 delete 메서드를
가져와서 그대로 사용!



컨트롤러에 삭제 기능 매핑

```
// 게시물 삭제
```

```
@PostMapping("/delete")
```

```
public String postDelete(@RequestParam("id") String id, HttpServletRequest request)
```

```
    log.info("=====> 게시물 삭제 기능 호출, " + request.getRequestURI());
```

```
    long postId = Long.parseLong(id);
```

```
    int affectedRows = postRepository.delete(postId);
```

```
    if (affectedRows > 0) log.info("삭제 성공");
```

```
    return "redirect:/post/v1/show";
```

```
}
```

삭제 기능인 만큼 post 방식으로 요청

사실 delete 쓰는게 맞지만...
다들 강 post 씁니다 😊

요청에서 받은 id 파라미터를
리포지토리의 delete 에 전달하여
결국 Mapper 의 delete 가 MaBatis 에
전달 합니다!!



```
// 게시물 삭제
@PostMapping("/delete")
public String postDelete(@RequestParam("id") String id, HttpServletRequest request) {
    log.info("===== > 게시물 삭제 기능 호출, " + request.getRequestURI());

    long postId = Long.parseLong(id);
    int affectedRows = postRepository.delete(postId);

    if (affectedRows > 0) log.info("삭제 성공");

    return "redirect:/post/v1/show";
}
```

쿼리 성공 여부는
해당 쿼리로 변경된 Row의 수를 받으므로
affectedRows가 1 이상이면 정상 작동!

그리고 redirect로 게시물 목록 컨트롤러를
다시 요청하여 게시물 목록 페이지로 이동!



그란데 말입니다

왜 여기서는 post-show.jsp 파일을
다시 요청 안하고

기존 게시판 목록을 요청하는 주소로
리다이렉트를 할까요!?



그란데 말입니다

삭제 기능 테스트





```
if (affectedRows > 0) log.info("삭제 성공");
```

```
INFO : org.example.controller.post.PostController - =====> 게시물 삭제 기능 호출,  
      /post/v1/delete  
INFO : org.example.controller.post.PostController - 삭제 성공  
INFO : org.example.controller.post.PostController - =====> 게시물 목록 페이지 호출,  
      /post/v1/show
```

글 목록



제목 검색

제목에서 찾을 단어 입력

/

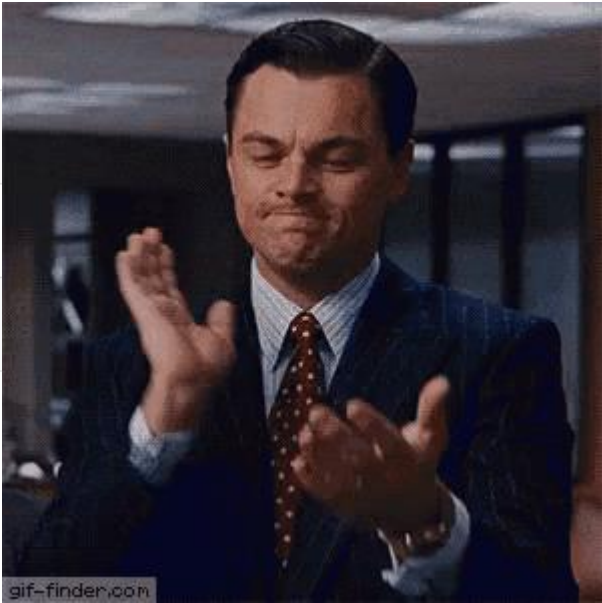
내용 검색

내용에서 찾을 단어 입력

검색

ID	Title	Content	Actions	
1	첫 번째 게시물	이것은 첫 번째 게시물의 내용입니다. a	수정	삭제
2	두 번째 게시물	이것은 두 번째 게시물의 내용입니다. b	수정	삭제
3	세 번째 게시물	이것은 세 번째 게시물의 내용입니다. c	수정	삭제
4	네 번째 게시물	이것은 네 번째 게시물의 내용입니다. d	수정	삭제

새글 작성하기





실습!

글 추가 기능 만들기!

실습, 글 추가 기능 만들기!



- 새로운 게시글을 추가하는 기능을 만들어 봅시다
- 게시글 추가 기능을 위해서는 크게 2가지 작업을 해야 합니다
- 첫째, 새로운 게시글을 쓰는 페이지를 불러오는 컨트롤러
- 둘째, 새로운 게시글 추가 요청이 들어가면 해당 요청을 받아서 DB 에 새로운 데이터를 추가하고 다시 게시글 목록 페이지로 리다이렉트하는 기능
- 아래 주어진 코드를 활용해서 게시글 추가 기능을 만들어 주세요!

실습, 글 추가 기능 만들기!



```
<a class="new-button" href="/post/v1/new">새글 작성하기</a>  
</body>  
</html>
```

- 새 글 추가하기 페이지는 GET 방식 /post/v1/new 의 요청을 받아서 페이지가 보여 집니다
- 새 글 추가 페이지는 post-new.jsp 파일로 작성 하시면 되며, 아래의 github 코드를 그대로 사용하면 됩니다!
- <https://github.com/xenosign/spring-code-repo/blob/main/post-new.jsp>

실습, 글 추가 기능 만들기!



```
<form action="/post/v1/new" method="post">
  <label for="title">제목</label>
  <input type="text" id="title" name="title" required>

  <label for="content">내용</label>
  <textarea id="content" name="content" required></textarea>

  <input type="submit" value="글 작성">
  <a class="cancel-button" href="/post/v1/show">취소</a>
</form>
```

- 새 글 추가 페이지 요청과 실제 글 추가 기능은 Method 로 구분 됩니다
- GET 방식 요청은 새 글 추가 페이지를 보여주고, POST 방식을 실제로 새로운 글을 추가하는 기능을 처리합니다!

실습, 글 추가 기능 만들기!



```
@Mapper 4 usages kdtTetz *
public interface PostMapper {
    List<PostDto> findAll(); 1 usage kdtTetz
    int delete(@Param("id") Long id); 1 usage kdtTetz
    int save(@Param("title") String title, @Param("content") String content);
}
```

- PostMapper 인터페이스의 코드입니다. 게시글 추가 요청은 save 메서드를 사용하며 title, content 를 매개변수로 받아 @Param 어노테이션을 사용하여 MyBatis 에 title, content 라는 이름으로 전달 합니다!
- 여러분은 PostMapper.xml 파일의 쿼리를 완성해 주시면 됩니다
- 또한, PostRepository 코드와 PostController 코드를 완성하시면 됩니다

실습, 글 추가 기능 만들기!



```
// 게시물 추가
@PostMapping("/new") new *
public String postSave(
    @RequestParam("title") String title,
    @RequestParam("content") String content,
    HttpServletRequest request
) {
    log.info("=====> 게시물 추가 기능 호출, " + request.getRequestURI());

    // 게시물 추가 기능 수행

    return "redirect:/post/v1/show";
}
```

- 해당 코드는 PostController 의 게시물 추가 기능 컨트롤러입니다!
- 게시물 추가가 성공하면 log 를 활용하여 게시물 추가 성공을 출력 한 다음, 게시물 목록 페이지로 리다이렉트 시켜서 추가된 게시글이 바로 보여지면 됩니다!



글 작성

제목

글 작성 테스트 입니다

내용

글 작성 테스트 입니다!!

```
INFO : org.example.controller.post.PostController - =====> 게시글 추가 기능 호출,  
/post/v1/new  
INFO : org.example.controller.post.PostController - 게시글 추가 성공!
```

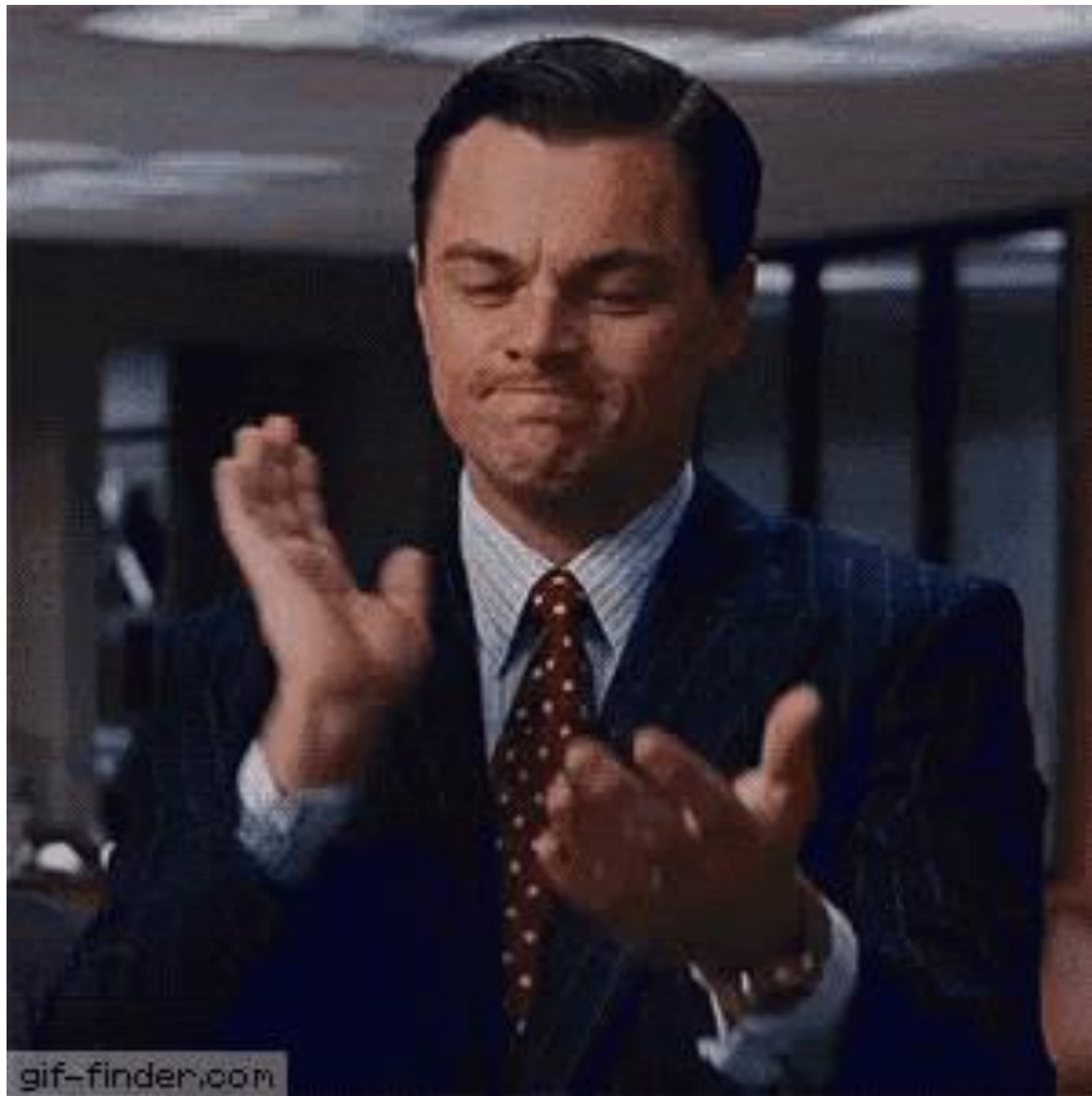
글 작성

취소

제목 검색 / 내용 검색

ID	Title	Content
1	첫 번째 게시물	이것은 첫 번째 게시물의 내용입니다. a
2	두 번째 게시물	이것은 두 번째 게시물의 내용입니다. b
3	세 번째 게시물	이것은 세 번째 게시물의 내용입니다. c
4	네 번째 게시물	이것은 네 번째 게시물의 내용입니다. d
6	글 작성 테스트 입니다	글 작성 테스트 입니다!!

새글 작성하기





MyBatis 를 이용한 동적 쿼리!



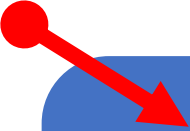
조건이 2개인

검색 기능 만들기!

글 목록



제목 검색 / 내용 검색



동적 쿼리 테스트를 위해서
제목과 내용에서 각각 내용을 찾거나
아니면 두 조건을 만족하는 글을 찾는
검색 기능을 만들어 보시다!



```
<form action="/post/v1/search" method="get">
  <label for="title">제목 검색</label>
  <input type="text" name="title" id="title" placeholder="제목에서 찾을 단어 입력">
  <label for="content">/ 내용 검색</label>
  <input type="text" name="content" id="content" placeholder="내용에서 찾을 단어 입력">
  <input type="submit" value="검색">
</form>
```

post-show.jsp 의 검색 파트를 보면
제목과 내용을 각각 title, content 으로
받아서 /post/v1/search 주소로
요청을 보냅니다!



제목 검색 / 내용 검색

제목 검색 / 내용 검색

제목 검색 / 내용 검색

제목 검색 / 내용 검색

즉, 검색 쿼리는 위의 4가지 상황에 맞게 쿼리문이 적절하게 변경 되어야 합니다!

제목과 내용이 없는 경우 → 전체 검색
제목만 있는 경우 → 제목에서 해당 단어가 들어간 글 검색
내용만 있는 경우 → 내용에서 해당 단어가 들어간 글 검색
제목과 내용이 다 있는 경우 → 두 조건을 만족하는 글 검색



그런데 말입니다

여기서 MyBatis 를 쓰지 않고
JDBC 로 동적 쿼리를 처리하려면
어떻게 해야 할까요!?



그런데 말입니다



```
public List<PostDto> findByCond(String title, String content) throws SQLException {  
    List<PostDto> results = new ArrayList<>();  
  
    String baseQuery = "SELECT id, title, content FROM posts WHERE 1=1";  
    String titleCondition = "";  
    String contentCondition = "";  
  
    if (title != null && !title.isEmpty()) {  
        titleCondition = " AND title LIKE ?";  
    }  
    if (content != null && !content.isEmpty()) {  
        contentCondition = " AND content LIKE ?";  
    }  
}
```

모든 부분을 문자열로 처리를 해야 합니다!

조건에 따라서 문자열을 생성해서
넣어주는 로직이 들어가야 하므로
if 문이 많이 생성 됩니다!



```
String finalQuery = baseQuery + titleCondition + contentCondition;  
  
try (Connection connection = dataSource.getConnection();  
    PreparedStatement statement = connection.prepareStatement(finalQuery)) {  
  
    int parameterIndex = 1;  
  
    if (!titleCondition.isEmpty()) {  
        statement.setString(parameterIndex++, "%" + title + "%");  
    }  
    if (!contentCondition.isEmpty()) {  
        statement.setString(parameterIndex++, "%" + content + "%");  
    }  
}
```

거기에다가 PreparedStatement 까지
사용해서 매개 변수로 받은 값의 유무에
따라 추가를 하고 빼줘야 합니다!

+ 우리가 잊고 있었던 Try / Catch 는
덤입니다 😊



그란데 말입니다

단순 검색 조건이 2개인데 코드는
23 줄이 추가가 되었습니다 😊
여러분들은 요런걸 쓰고 싶으신가요!?





해당 기능을 MyBatis 로!



PostMapper

인터페이스 작업

```
@Mapper 4 usages kdtTetz *  
public interface PostMapper {  
    List<PostDto> findAll(); 1 usage kdtTetz  
    int delete(@Param("id") Long id); 1 usage kdtTetz  
    int save(@Param("title") String title, @Param("content") String content); new *  
    List<PostDto> findByCond(@Param("title") String title, @Param("content") String content);  
}
```

Mapper 인터페이스에 조건에 따른 조건 메서드인
findByCond 를 추가해 줍니다!

해당 메서드는 title, content 조건에 따라 변화하므로
매개변수도 2개를 받아서 처리해 줍니다!



PostMapper.xml

작업



PostMapper 인터페이스의 메서드명과 리턴 타입 설정

```
<select id="findByCond" resultType="PostDto">
    SELECT id, title, content
    FROM posts
    <where>
        <if test="title != null and title != ''">
            AND title LIKE CONCAT('%', #{title}, '%')
        </if>
        <if test="content != null and content != ''">
            AND content LIKE CONCAT('%', #{content}, '%')
        </if>
    </where>
</select>
```

기본이 되는 SQL 쿼리 작성하기

```
<select id="findByCond" resultType="PostDto">
```

```
    SELECT id, title, content
```

```
    FROM posts
```

```
    <where>
```

```
        <if test="title != null and title != ''">
```

```
            AND title LIKE CONCAT('%', #{title}, '%')
```

```
        </if>
```

```
        <if test="content != null and content != ''">
```

```
            AND content LIKE CONCAT('%', #{content}, '%')
```

```
        </if>
```

```
    </where>
```

```
</select>
```

where 절 부터는 MyBatis 의 기능을
활용하여 조건부 쿼리를 완성 합니다!

```
<select id="findByCond" resultType="PostDto">
    SELECT id, title, content
    FROM posts
    <where>
        <if test="title != null and title != ''">
            AND title LIKE CONCAT('%', #{title}, '%')
        </if>
        <if test="content != null and content != ''">
            AND content LIKE CONCAT('%', #{content}, '%')
        </if>
    </where>
</select>
```

특정 쿼리를 덧붙일 것인지 말것인지를
결정하는 조건을 <if> 로 부여!

AND 구문은 MyBatis 가 상황에 따라
추가 여부를 결정하므로
걱정 없이 사용하면 됩니다!

매개 변수로 전달한
title 과 content 의 유무에 따라
해당 쿼리문이 자동으로 처리!



PostRepository

작업



```
public List<PostDto> findByCond(String title, String content) {  
    return postMapper.findByCond(title, content);  
}
```

컨트롤러 부터 전달 받은 title, content 매개 변수를
그대로 postMapper 인터페이스의 메서드에
전달하여 MyBatis 가 쿼리를 수행 하도록 설정



PostController

작업

```
// 게시물 검색
```

```
@GetMapping("/search") new *
```

```
public String postSearch(
```

```
    @RequestParam("title") String title,
```

```
    @RequestParam("content") String content,
```

```
    HttpServletRequest request,
```

```
    Model model
```

```
) {
```

```
    log.info("=====> 게시물 검색 기능 호출, " + request.getRequestURI());
```

```
    model.addAttribute(attributeName: "postList", postRepository.findByCond(title, content));
```

```
    return context + "/post-show";
```

```
}
```

조건에 따라 검색을 해야하므로
jsp 페이지로 부터 전달 된 파라미터를
각각 변수에 담기

MyBatis 에 의해 수행 된
조건 검색의 결과 데이터를 받아서
해당 데이터를 post-show.jsp 로 전달



확인할 시간!!

(잔뜩 기대중)



제목 검색 / 내용 검색

ID	Title	Content
1	첫 번째 게시물	이것은 첫 번째 게시물의 내용입니다. a
2	두 번째 게시물	이것은 두 번째 게시물의 내용입니다. b
3	세 번째 게시물	이것은 세 번째 게시물의 내용입니다. c
4	네 번째 게시물	이것은 네 번째 게시물의 내용입니다. d
6	글 작성 테스트 입니다	글 작성 테스트 입니다!!



제목 검색 / 내용 검색

ID	Title	Content
6	글 작성 테스트 입니다	글 작성 테스트 입니다!!



제목 검색 / 내용 검색

제목 검색 / 내용 검색

ID	Title	Content
4	네 번째 게시물	이것은 네 번째 게시물의 내용입니다. d



제목 검색 게시물 / 내용 검색 a 검색

제목 검색 제목에서 찾을 단어 입력 / 내용 검색 내용에서 찾을 단어 입력 검색

ID	Title	Content
1	첫 번째 게시물	이것은 첫 번째 게시물의 내용입니다. a

