



It's Your Life

with





Spring 과

MyBatis



가수명 **한두승** 내가 잊었거든은 언제?

JTBC



왜에 왜에!!! 와이!



JDBC 에서 SQL 쿼리 쓰던거 생각 나시나요?



```
String sql = "SELECT p.post_id, p.title, p.content, p.created_at, u.username "  
    + "FROM posts p "  
    + "JOIN users u ON p.user_id = u.id "  
    + "WHERE p.post_id = ?";
```

```
try (Connection conn = DriverManager.getConnection(JDBC_URL, JDBC_USER, JDBC_PASSWORD);  
    PreparedStatement pstmt = conn.prepareStatement(sql)) {
```





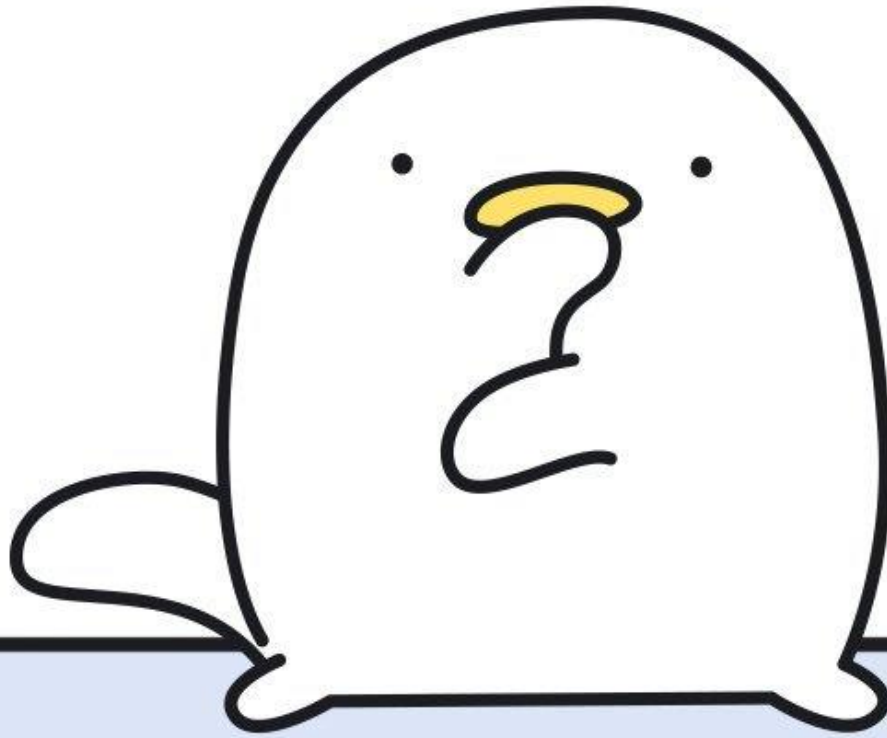
그런데 말입니다

그런데 말입니다
SQL 쿼리가 더 복잡해 진다면!?
쿼리가 동적으로 변해야 한다면!?

모든 쿼리 변경을 문자열로 처리 해줘야 합니다!

if 문을 자주 사용해야 하고
코드 가독성도 떨어집니다





왜 저러는 걸까요?

요기 공백이 보이시나요!?
요걸 빼먹으면 어떻게 될까요?



```
String sql = "SELECT p.post_id, p.title, p.content, p.created_at, u.username "  
+ "FROM posts p "  
+ "JOIN users u ON p.user_id = u.id "  
+ "WHERE p.post_id = ?";
```

재택근무



기대



현실





기대

```
SELECT p.post_id, p.title, p.content, p.created_at, u.username  
FROM posts p  
JOIN users u ON p.user_id = u.id  
WHERE p.post_id = ?
```





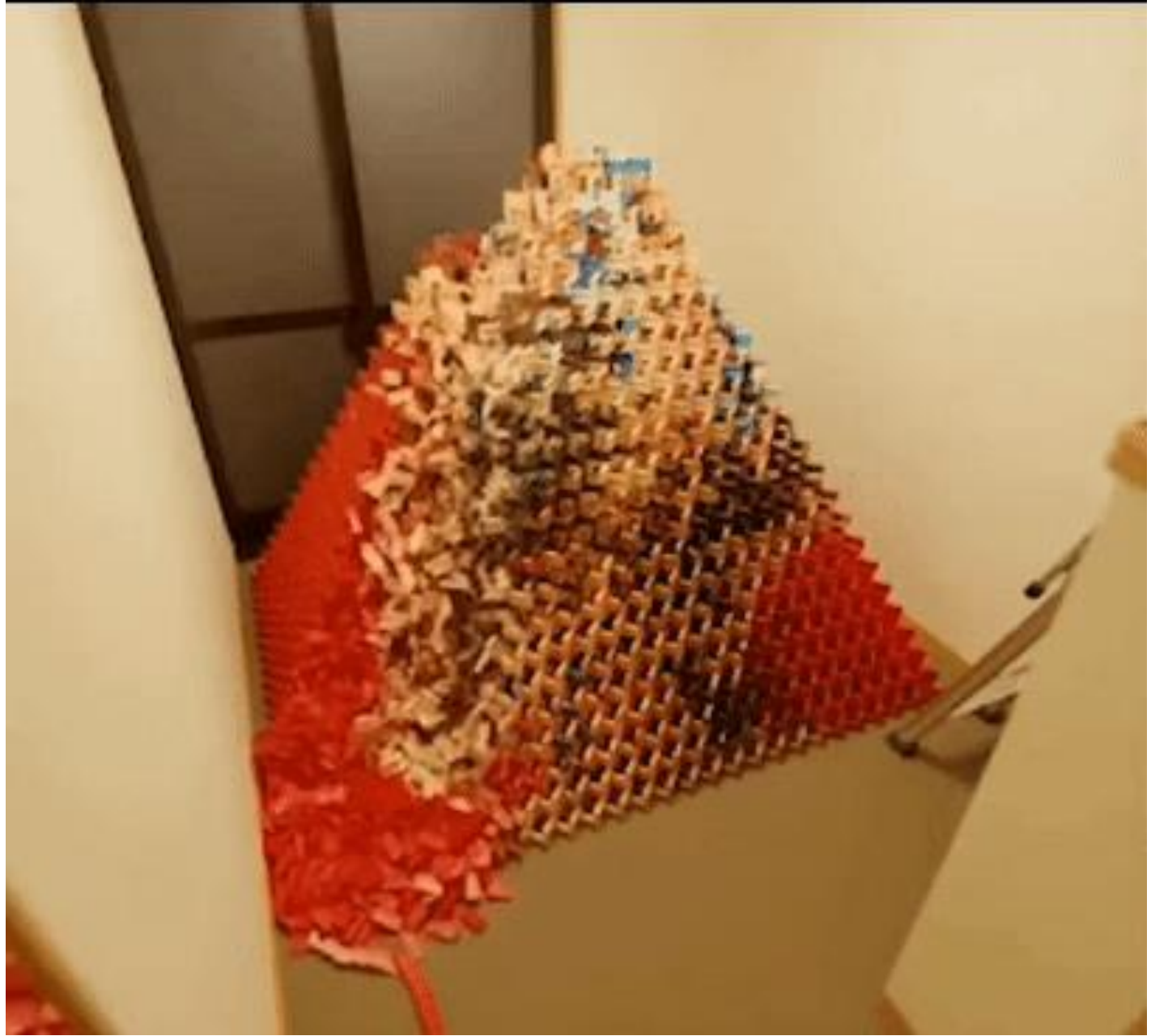
현실

```
SELECT p.post_id, p.title, p.content, p.created_at, u.username
FROM posts p
JOIN users u ON p.user_id = u.id
WHERE p.post_id = ?
```



후후 고작 칼 하나로 절 막을수 있다고
생각한건가요?







개발자



튀



MyBatis 의 등장으로!



- 동적 쿼리를 처리하기 편해졌습니다
- 전반적으로 자바 또는 스프링에서 데이터베이스 통신을 하기가 편해졌습니다
- 다만..... 세팅이 좀 귀찮기는 합니다! ☹



MyBatis

세팅하기



오케이 이제 부터 시작이다 렛츠고

build.grade 세팅



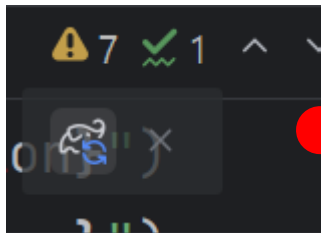


```
// 데이터베이스  
implementation 'com.mysql:mysql-connector-j:8.1.0'  
implementation 'com.zaxxer:HikariCP:2.7.4'  
implementation "org.springframework:spring-tx:${springVersion}"  
implementation "org.springframework:spring-jdbc:${springVersion}"  
implementation 'org.mybatis:mybatis:3.4.6'  
implementation 'org.mybatis:mybatis-spring:1.3.2'
```

<https://github.com/xenosign/spring-code-repo/blob/main/mybatis1.gradle>



해당 레포의 코드를
build.gradle 에 추가해 주세요!



코끼리 눌러서 싱크 작업!





오케이 이제 부터 시작이다 렛츠고

▼ resources

> org

⚙ application.properties

</> log4j.xml

</> mybatis-config.xml





application.

properties 세팅



```
jdbc.driver=com.mysql.cj.jdbc.Driver  
jdbc.url=jdbc:mysql://localhost:3306/mybatis  
jdbc.username=root  
jdbc.password=1234
```

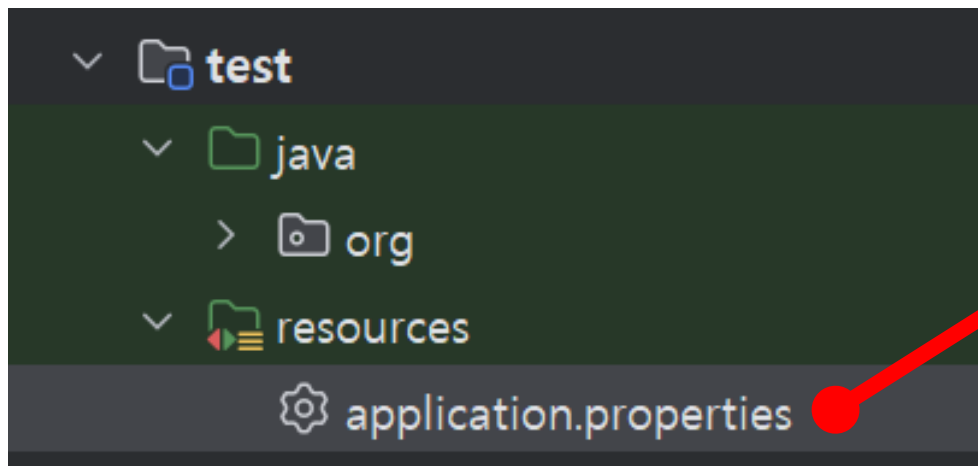
데이터 베이스 이름,
본인 root 아이디, 비밀번호
확인 필수!!!!!!!!!!!!!!

<https://github.com/xenosign/spring-code-repo/blob/main/application.properties>

정말 수많은 개발자들에게
시련을 준 것이 있습니다!!

잠

판



테스트 유닛은 속성 정보를
자신이 가지고 있는 resources 폴더에서
가지고 오기 때문에 여기다가
또 파일을 등록 해줘야 합니다!!



```
jdbc.driver=com.mysql.cj.jdbc.Driver  
jdbc.url=jdbc:mysql://localhost:3306/mybatis  
jdbc.username=root  
jdbc.password=1234
```



<https://github.com/xenosign/spring-code-repo/blob/main/application.properties>



mybatis-config.xml

세팅



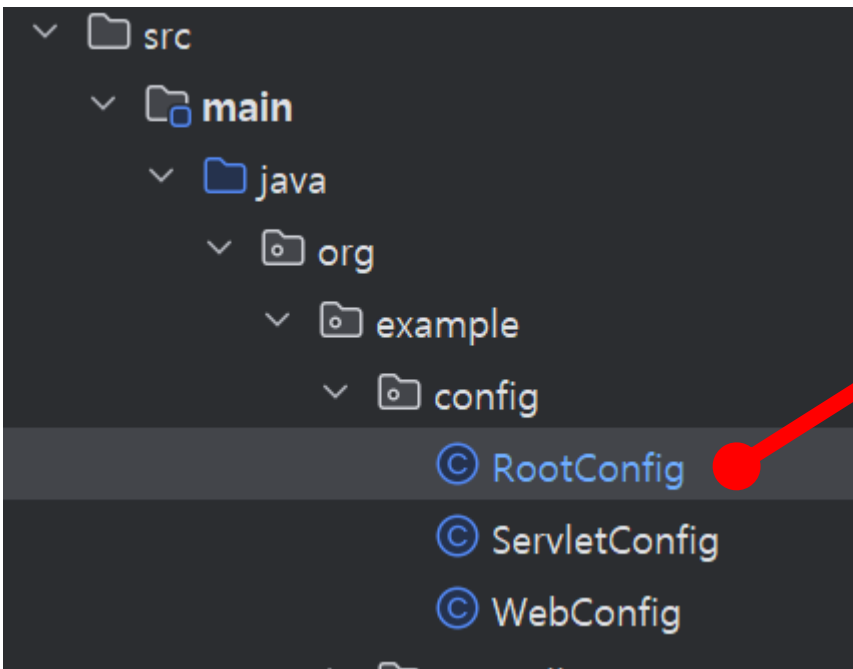
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE configuration
    PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
</configuration>
```

<https://github.com/xenosign/spring-code-repo/blob/main/mybatis-config.xml>



Spring

RootConfig 설정



이제 MyBatis 설정을
스프링에 적용해 봅시다!

RootConfig 에 코드 추가!

<https://github.com/xenosign/spring-code-repo/blob/main/RootConfig.java>



```
@Configuration  @Tetz +1 *
@ComponentScan(basePackages = "org.example")
@MapperScan(basePackages = {"org.example.mapper"})
public class RootConfig {

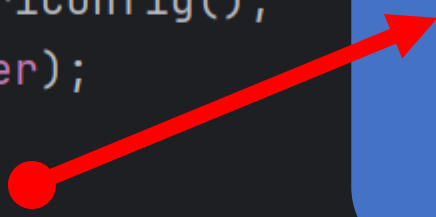
    @Value("${jdbc.driver}")
    private String driver;
    @Value("${jdbc.url}")
    private String jdbcUrl;
    @Value("${jdbc.username}")
    private String username;
    @Value("${jdbc.password}")
    private String password;
```

application.properties 로 부터
지정한 값을 꺼내오는 코드

@Value 어노테이션 사용

@Bean Tetz *

```
public DataSource dataSource() {  
    HikariConfig config = new HikariConfig();  
    config.setDriverClassName(driver);  
    config.setJdbcUrl(jdbcUrl);  
    config.setUsername(username);  
    config.setPassword(password);  
    HikariDataSource dataSource = new HikariDataSource(config);  
    return dataSource;  
}
```



Hikari 라이브러리를 이용하여
데이터 베이스를 연결하고
연결한 정보를 스프링 Bean 으로 등록!

스프링 빈을 전체 관리하는
ApplicationContext 를 @Autowired 로
주입 받아서 사용하기!

```
@Autowired  
ApplicationContext applicationContext;
```

```
@Bean  👤 Tetz
```

```
public SqlSessionFactory sqlSessionFactory() throws Exception {  
    SqlSessionFactoryBean sqlSessionFactory = new SqlSessionFactoryBean();  
    sqlSessionFactory.setConfigLocation(  
        applicationContext.getResource(location: "classpath:/mybatis-config.xml"));  
    sqlSessionFactory.setDataSource(dataSource());  
    return (SqlSessionFactory) sqlSessionFactory.getObject();  
}
```


```
@Bean  👤 Tetz
```

```
public DataSourceTransactionManager transactionManager(){  
    DataSourceTransactionManager manager = new DataSourceTransactionManager(dataSource());  
    return manager;  
}
```


마이 바티스 관련 설정 코드들입니다!

@Autowired

```
ApplicationContext applicationContext;
```

@Bean  Tetz

```
public SqlSessionFactory sqlSessionFactory() throws Exception {  
    SqlSessionFactoryBean sqlSessionFactory = new SqlSessionFactoryBean();  
    sqlSessionFactory.setConfigLocation(  
        applicationContext.getResource(location: "classpath:/mybatis-config.xml"));  
    sqlSessionFactory.setDataSource(dataSource());  
    return (SqlSessionFactory) sqlSessionFactory.getObject();  
}
```

@Bean  Tetz

```
public DataSourceTransactionManager transactionManager(){  
    DataSourceTransactionManager manager = new DataSourceTransactionManager(dataSource()  
    return manager;  
}
```



접속 테스트!

```
@Value("${jdbc.username}")
private String username;
@Value("${jdbc.password}")
private String password;

@Bean
public DataSource dataSource() {
    HikariConfig config = new HikariConfig();
    config.setDriverClassName("com.mysql.cj.jdbc.Driver");
    config.setJdbcUrl("jdbc:mysql://localhost:3306/testdb");
    config.setUsername(username);
    config.setPassword(password);
    HikariDataSource dataSource = new HikariDataSource(config);
    return dataSource;
}
```

Show Context Actions

Paste

Copy / Paste Special

Column Selection Mode Alt+Shift+Insert

Find Usages Alt+F7

Go To

Folding

Analyze

Refactor

Generate... Alt+Insert

Open In

접속 테스트 를 위해서
dataSource 우클릭 → Go To → Test 선택

Navigation Bar Alt+Home

Declaration or Usages Ctrl+B

Implementation(s) Ctrl+Alt+B

Type Declaration Ctrl+Shift+B

Super Method Ctrl+U

Related Symbol... Ctrl+Alt+Home

Test Ctrl+Shift+T



```
@Bean  Tetz *
public DataSource dataSource() {
    HikariConfig config
    config.setDriverClass
    config.setJdbcUrl(jd
    config.setHikariConfigName("user")
}
```

Choose Test for RootConfig (0 found) 📌

💡 Create New Test...

Create New Test 선택

Create Test

Testing library: JUnit5

Class name: RootConfigTest

Superclass:

Destination package: org.example.config

Generate:

☐ setUp/@Before

☐ tearDown/@After

Generate test methods for: ☐ Show inherited methods

	Member
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/> dataSource():DataSource
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/> sqlSessionFactory():SqlSessionFactory
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/> transactionManager():DataSourceTransactionManager

?

OK Cancel



일단 모든 테스트를 선택 → OK

```
7  ✓ class RootConfigTest {  👤 Tetz *  
8  
9      @Test new *  
10 void dataSource() {  
11     }  
12  
13     @Test new *  
14 void sqlSessionFactory() {  
15     }  
16  
17     @Test new *  
18 void transactionManager() {  
19     }  
20 }
```





테스트 코드 작성!



```
✓ @Configuration  👤 Tetz *  
@SpringJUnitConfig  
@ContextConfiguration(classes = RootConfig.class)  
@Slf4j  
@PropertySource("classpath:application.properties")  
class RootConfigTest {
```

```
✓ @Test  new *  
void sqlSessionFactory() {  
}
```

```
}
```

테스트 실행을 위한 어노테이션 추가!

```
class RootConfigTest {  
    @Autowired  
    private SqlSessionFactory sqlSessionFactory;
```

RootConfig 에서 등록한
SqlSessionFactory 주입 받기

```
@Test @Tetz *  
void sqlSessionFactory() {  
    try (SqlSession session = sqlSessionFactory.openSession();  
        Connection con = session.getConnection()) {  
        log.info("SqlSession: {}", session);  
        log.info("Connection: {}", con);  
    } catch (Exception e) {  
        fail(e.getMessage());  
    }  
}
```

설정한 Sql 서버에 접속이 가능한지
확인하는 테스트

Connection 이 안되면
예외가 발생하므로 테스트가 실패함

우리 DB 만들었나요!?

잠

관

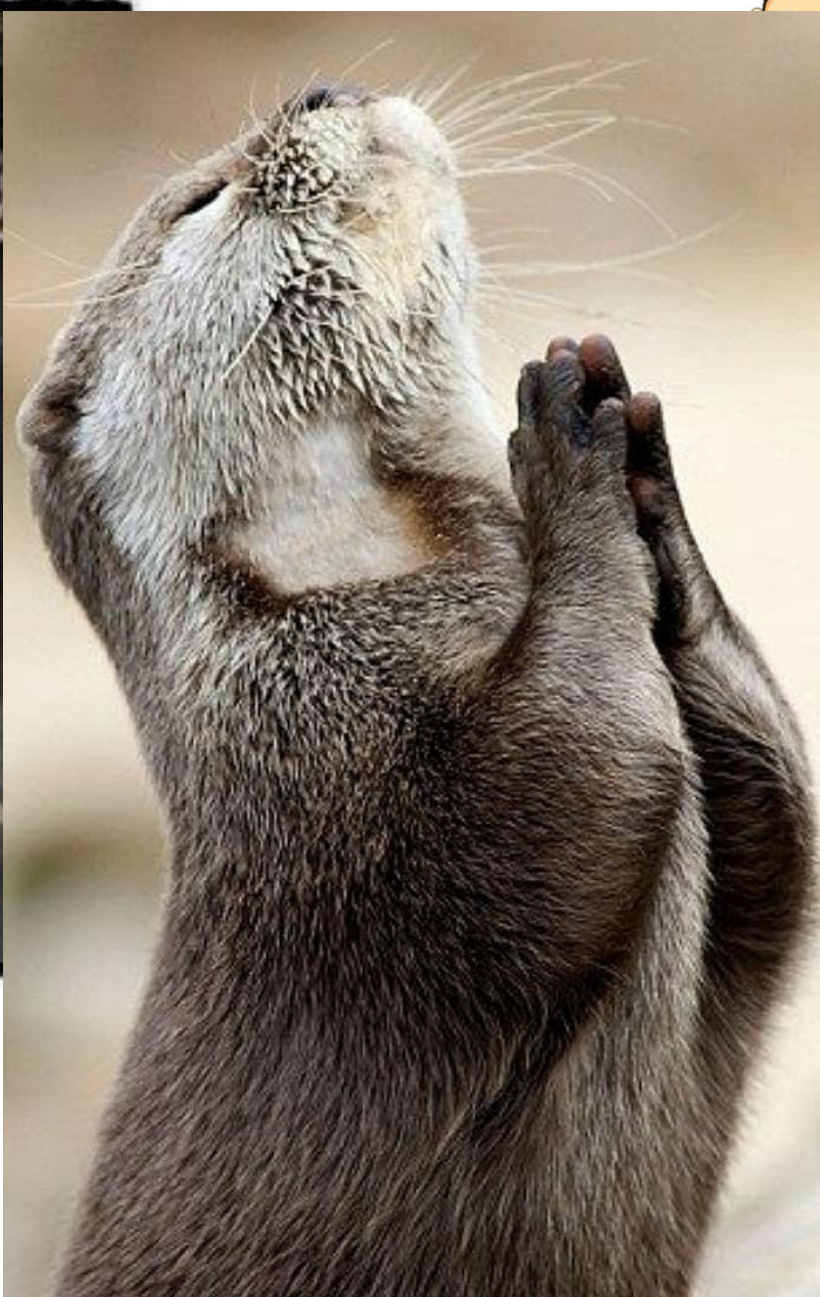


데이터 베이스 만들기



```
1 • CREATE DATABASE mybatis;
2 • USE mybatis;
3
4 • CREATE TABLE members (
5     id VARCHAR(50) NOT NULL,
6     name VARCHAR(50) NOT NULL
7 );
8
9 • INSERT INTO members (id, name) VALUES ('tetz', '이효석');
10 • INSERT INTO members (id, name) VALUES ('siwan', '김시완');
11 • SELECT * FROM members;
```

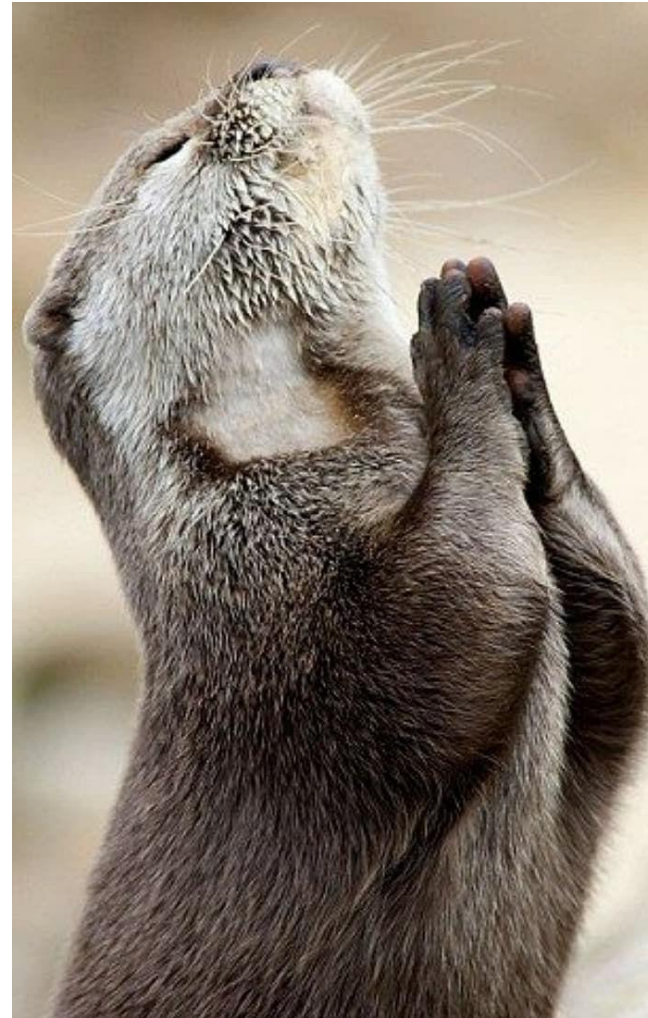
<https://github.com/xenosign/spring-code-repo/blob/main/mybatis.sql>





실행을 눌러서 테스트를 수행!

```
22 @PropertySource("classpath:application-test.properties")
23 class RootConfigTest {
24     @Autowired
25     private SqlSessionFactory sqlSessionFactory;
26
27     @Test
28     void sqlSessionFactory() {
29         try (SqlSession session = sqlSessionFactory.openSession()) {
30             Connection connection = session.getConnection();
31             log.info("SqlSession created: " + session);
32             log.info("Connection created: " + connection);
33         } catch (Exception e) {
34             fail(e.getMessage());
35         }
36     }
37 }
```



Test Results

41 ms

Tests passed: 1 of 1 test – 41 ms

> Task :processTestResources UP-TO-DATE

> Task :testClasses UP-TO-DATE

INFO : org.springframework.test.context.support.DefaultTestContextBootstrapper - Loaded default

TestExecutionListener class

.context.web.ServletTest

.DirtyContextBeforeMod

.ApplicationEventsTestEx

.DependencyInjectionTest

.DirtyContextTestExecu

.TransactionalTestExecut

.SqlScriptsTestExecution

.EventPublishingTestExec

INFO : org.springframework

TestExecutionListeners:

org.springframework.test

A man in a white shirt is running through a crowd of people, looking surprised or excited. He is surrounded by several "oh" speech bubbles in different colors (purple, blue, green, yellow). The background is blurred, suggesting a fast-paced environment.



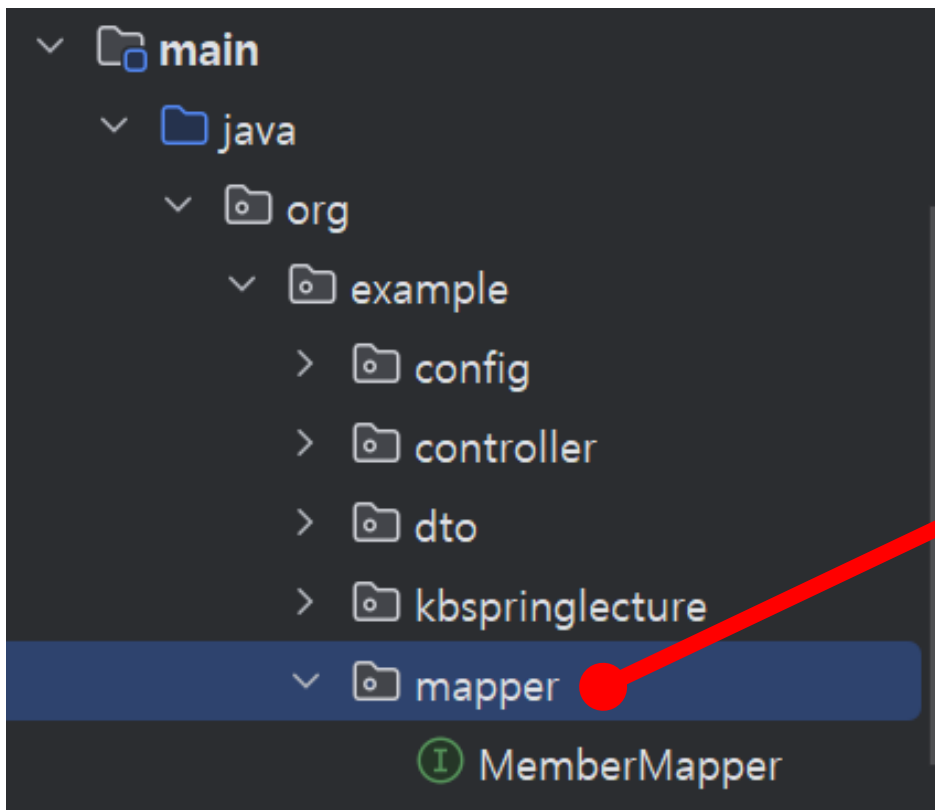


MyBatis 로 데이터 가져오기!



mapper

인터페이스 설정



MyBatis 는 mapper 라는
인터페이스를 이용해서 데이터를 컨트롤 합니다!

즉, 인터페이스로 틀만 만들고
내부 구현은 MyBatis 가 채우는 형태!

```
@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!



MyBatis 는 그럼 어떻게
코드를 채울까요!?



그런데 말입니다



mapper

xml 파일 생성

resources

org

example

mapper

MemberMapper.xml

바로 resources 폴더의 ~~.xml 파일이
해당 부분을 채워 줍니다!

여기서 주의 하셔야 할 점은
resources 폴더는 Java 폴더가 아니기 때문에
폴더를 . 으로 구분하면 안됩니다
반드시 / 를 사용해서 만들어 줘야합니다!!

→ . 을 사용하면 org.example.mapper 라는
폴더만 생성이 됩니다!



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

xml 파일을 통해서 실제 데이터 접근을
컨트롤 합니다!

SQL 구문을 명시

해당 xml 파일에 매칭되는
인터페이스 위치를 정확하게 입력하기!


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

인터페이스에서 매칭되는 추상 메서드 이름과
리턴 타입을 명시하기!

실제로 수행할 쿼리문을 작성



```
@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



그란데 말입니다

MyBatis 는 MemberDto 가 어떻게 생겼는지
알 수 있을까요!?





resources
org
application.properties
log4j.xml
log4jdbc.log4j2.properties
mybatis-config.xml

설정 파일에 Dto 객체 위치를 설정

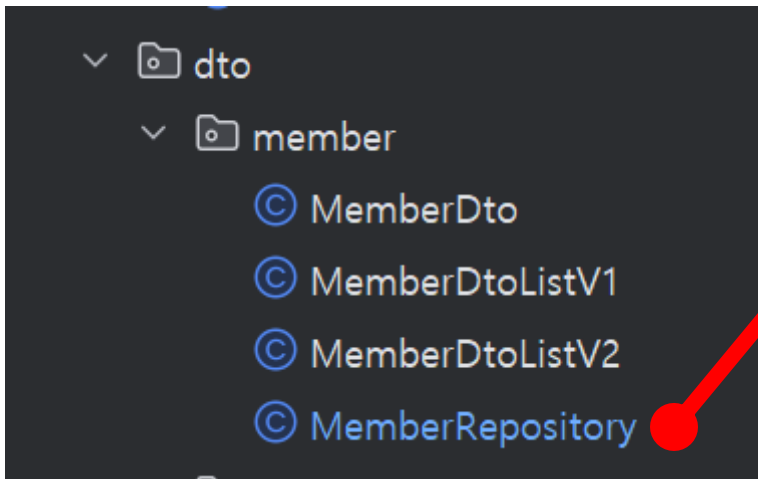


```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE configuration
  PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
  "http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
  <typeAliases>
    <typeAlias alias="MemberDto" type="org.example.dto.member.MemberDto"/>
  </typeAliases>
</configuration>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

타입에 대한 별명을 설정하고
해당 별명이 어느 객체를 참조하는지 지정!



MyBatis 를 사용한 데이터 계층



마이바티스를 이용하여 DB로부터
데이터를 불러오는 클래스를 만들어 봅시다!

이 친구는 정확히는 저장소 의미로 사용되므로
Repository로 명명합니다!



```
@Repository kdtTetz *
public class MemberRepository {
    private final MemberMapper memberMapper; 2 usages

    @Autowired kdtTetz *
    public MemberRepository(MemberMapper memberMapper) {
        this.memberMapper = memberMapper;
    }

    public MemberDto save(final MemberDto memberDto) { return null; }

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

@Repository 어노테이션을 쓰면
자동으로 해당 클래스를
스프링 빈으로 등록합니다!

@Component 와 동일한 기능 이지만
데이터 저장 및 전송에 특화된
어노테이션 입니다!


```
@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!

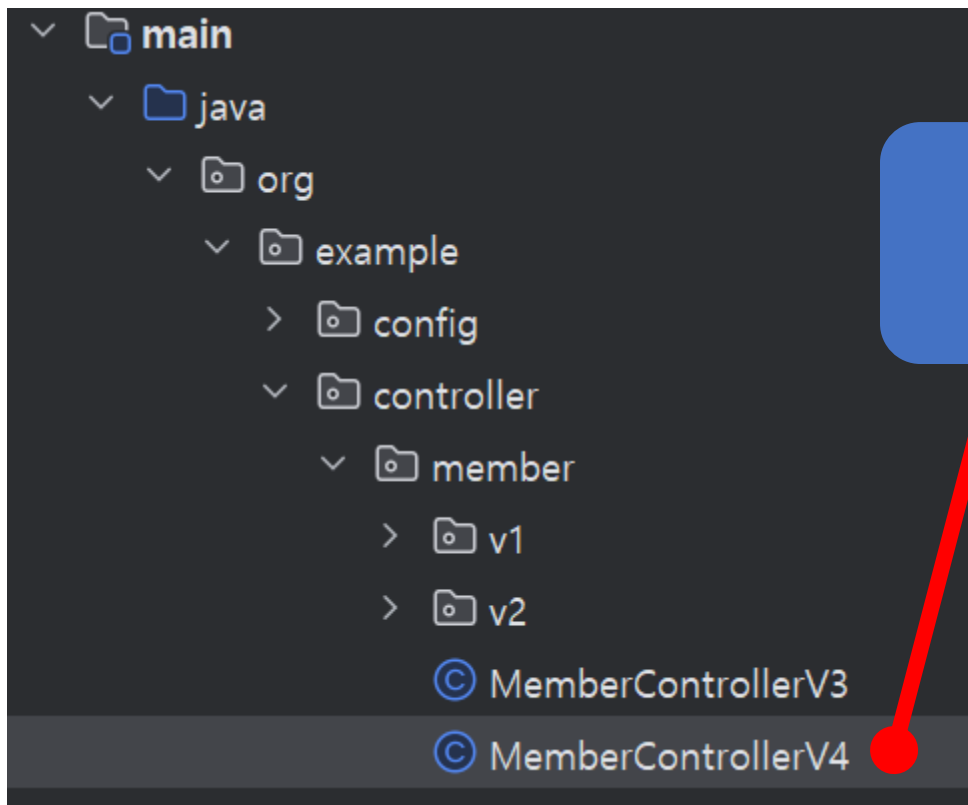
Mapper 인터페이스에 있는
findAll() 메서드를 사용하여
원하는 데이터를 DB 부터 받아서
가져옵니다!



MyBatis 를 사용한 컨트롤러!



마이바티스를 적용한 V4 컨트롤러를 만들어
봅시다!





```
@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";
    }
}
```

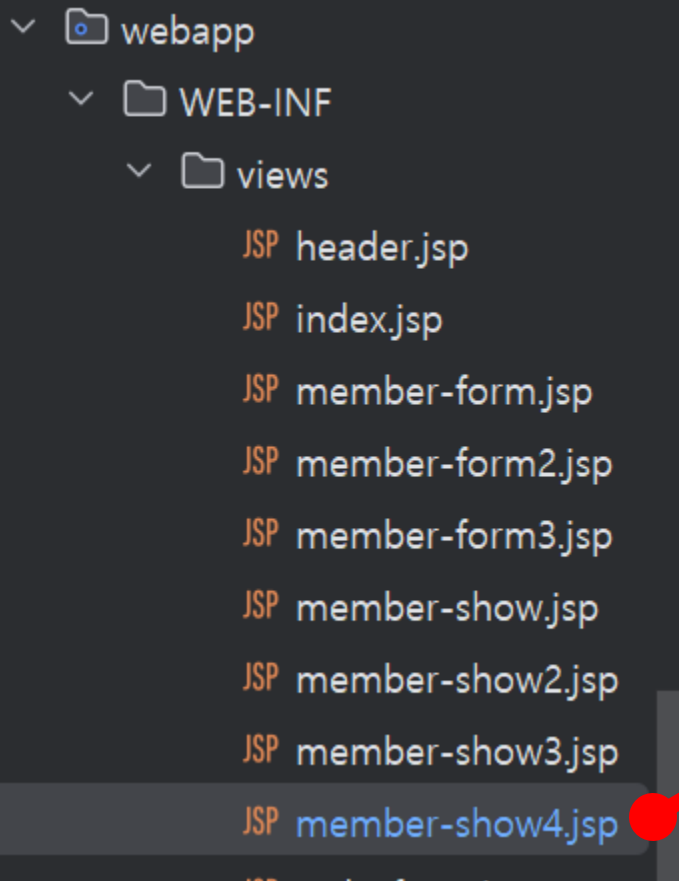
@Autowired 어노테이션을 사용하여
@Repository 에 등록된
MemberRepository 를 자동 주입

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



View

페이지 추가



데이터 베이스로부터 받은
데이터를 출력하는 show4
뷰 페이지 작성



```
<%@ 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 - 이호석

