2018 January 12

SeokRae Kim

코드로 배우는 스프링 웹 프로젝트

Spring Study

내용

[**I.** **스프링 웹 프로젝트 스터디** 1](#_Toc503896315)

[**1.** **프로젝트의 기본 구조 구성** 1](#_Toc503896316)

[**1)** **개발 환경의 설정** 1](#_Toc503896317)

[**2)** **MySQL의 설정과 스프링 테스트** 8](#_Toc503896318)

[**3)** **스프링 + MyBatis + MySQL의 설정** 9](#_Toc503896319)

[**4)** **모델 2 방식과 스프링 MVC** 14](#_Toc503896320)

[**5)** **스프링 + MyBatis** 20](#_Toc503896321)

[**6)** **정리** 27](#_Toc503896322)

1. **스프링 웹 프로젝트 스터디**

본 문서는 “코드로 배우는 스프링 웹 프로젝트”책의 내용을 기반으로 하여 라이브러리의 필요성과 사용 목적, 코딩실습에 관련된 내용만이 작성되었습니다.

스프링의 개념과 이론적인 부분에 대해서는 Spring Study Theory를 참고해주세요.

* 1. **프로젝트의 기본 구조 구성**

|  |
| --- |
| “**프로젝트의 기본 구조 및 구성**”에서는 Spring Project를 위해서 “**프로젝트 생성**”, “**라이브러리의 초기화**”, “**프로젝트 실행**” 순으로 진행하여 **스프링이 실행되는 것을 확인**하고, **스프링 프로젝트의 폴더 구조를 분석** 하는 것을 목표로 한다. |

* + 1. **개발 환경의 설정**
       1. **Eclipse에 STS 플러그인 추가**

|  |
| --- |
| **Eclipse 실행** |
|  |

|  |
| --- |
| **Help -> Eclipse Marketplace** |
|  |

|  |
| --- |
| **Find에 “sts”를 검색하여 Install** |
|  |

* + - * 1. **Maven 빌드 시 오류 날 때**
* maven이라는 빌드 도구가 필요한 jar파일들과 프로젝트를 구성하게 된다.
* 이 과정에서 네트워크의 문제나 기존의 라이브러리와 충돌로 인하여 제대로 안받아질 수도 있다.
* .**m2를 삭제 후 재시작하여** Maven 라이브러리를 다시 받도록 하는 것이 좋다.
* 또한 사용자의 이름과 폴더가 영문이 아닐 경우 제대로 설치가 안되는 경우도 있다.

|  |
| --- |
| **.m2 폴더 위치** |
|  |

* + - 1. **Spring Project를 이용한 프로젝트 생성**

|  |
| --- |
| **Spring Legacy Project 생성** |
|  |

|  |
| --- |
| **Project name에 “ex10”작성** |
|  |

|  |
| --- |
| **Package명 작성** |
|  |

* + - 1. **라이브러리 초기화**
         1. **프로젝트 실행 테스트**

|  |
| --- |
| **프로젝트 실행 테스트** |
|  |

* + - * 1. **JDK 버전의 처리**

|  |
| --- |
| **프로젝트 preferences** |
|  |

|  |
| --- |
| **Project Facets -> Java 버전 1.8로 수정** |
|  |

|  |
| --- |
| **Pom.xml의 springframework 버전 4.3.11 설정** |
|  |
| <properties>  <java-version>1.6</java-version>  <org.springframework-version>**4.3.11**.RELEASE</org.springframework-version>  <org.aspectj-version>1.6.10</org.aspectj-version>  <org.slf4j-version>1.6.6</org.slf4j-version>  </properties> |
|  |

|  |
| --- |
| **프로젝트 실행 테스트** |
|  |

* + - * 1. **스프링 MVC 프로젝트 템플릿 구조**

|  |
| --- |
|  |

* + - * 1. **설치 작업 정리**

|  |
| --- |
| * Eclipse에 STS 플러그인을 다운로드 * pom.xml을 이용해서 스프링 4.3.11버전에 맞는 라이브러리를 추가 * Tomcat 서버를 설정 이후 테스트 |

* + 1. **MySQL의 설정과 스프링 테스트**
       1. **MySQL 연결 테스트와 jUnit**
          1. **MySQL 테스트 관련 라이브러리**

|  |
| --- |
| **MySQL Connector/J JDBC 라이브러리 추가** |
| <!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <version>5.1.41</version>  </dependency> |

|  |
| --- |
| **JDK의 버전 변경** |
| <properties>  <java-version>**1.8**</java-version>  <org.springframework-version>4.3.11.RELEASE</org.springframework-version>  <org.aspectj-version>1.6.10</org.aspectj-version>  <org.slf4j-version>1.6.6</org.slf4j-version>  </properties> |

|  |
| --- |
| **jUnit의 버전 변경** |
| <!-- Test -->  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>**4.12**</version>  <scope>test</scope>  </dependency> |

* + - * 1. **JDBC 연결 테스트 try – with 코딩**

|  |
| --- |
| **test/java/패키지/MySQLConnectionTest.java** |
| public class MySQLConnection {  private static final String DRIVER = "com.mysql.jdbc.Driver";  private static final String URL = "jdbc:mysql://localhost:3306/{데이터베이스명}?useSSL=false";  private static final String USER = "{DB계정명}";  private static final String PW = "{DB비밀번호}";  @Test  public void testConnection() throws Exception {  Class.forName(DRIVER);  try (Connection con = DriverManager.getConnection(URL, USER, PW)) {  System.out.println(con);  } catch (Exception e) {  System.out.println(e);  }  }  } |
|  |

* + - 1. **작업 내역 정리**

|  |
| --- |
| * MySQL 설치 * MySQL JDBC 라이브러리 5.1.41 버전 추가 * Java 1.8 버전 수정 * jUnit 4.12 버전 수정 * JDBC 연결 테스크 코드 |

* + 1. **스프링 + MyBatis + MySQL의 설정**

|  |
| --- |
| **작업 목표** |
| * 스프링과 MyBatis를 연동하기 위한 라이브러리 설정 * 데이터베이스와의 연결을 담당하는 DataSource 객체 설정 * MyBatis의 핵심인 SqlSessionFactory 객체 설정 및 테스트 |

* + - 1. **일반적인 스프링 웹 프로젝트의 구성**
         1. **이 책의 프로젝트 구성**

|  |  |  |
| --- | --- | --- |
| **Presentation Layer** | | UI를 담당하는 구성 요소 |
|  | **Model** | 데이터를 다루는 영역 |
|  | **controller** | 모델과 뷰 영역간의 조정 역할 |
| **Business Logic Layer** | | 서비스 계층으로 고객의 요구사항을 반영하는 계층 |
|  | **Service** | 사용자 레벨 관리 기능 추가 |
| **Data Access Layer** | | Persistence Layer로 데이터 처리를 전문으로 담당 |
|  | **DAO** | DB를 사용해 데이터를 조회하거나 조작하는 기능을 전담하도록 만든 오브젝트 |
|  | **MyBatis** | 개발자가 지정한 SQL, 저장프로시저를 제공하는 퍼시스턴스 프레임워크 |

|  |
| --- |
|  |

* + - 1. **MyBatis와의 구성**

|  |
| --- |
|  |

* + - 1. **MyBatis 연동을 위한 준비**

|  |
| --- |
| **모듈 추가** |
| * **spring-jdbc**: JDBC의 장점과 단순성을 유지하는 간결한 api를 제공 * **spring-test**: 스프링 컴포넌트의 jUnit 혹은 TestNG와 관련된 유닛 테스트와 통합테스트 지원 |

* + - * 1. **Spring-jdbc, spring-test, MyBatis, mybatis-spring 추가**

|  |  |
| --- | --- |
| **이름** | **pom.xml** |
| **MyBatis** | <!-- https://mvnrepository.com/artifact/org.mybatis/mybatis -->  <dependency>  <groupId>org.mybatis</groupId>  <artifactId>mybatis</artifactId>  <version>3.4.1</version>  </dependency> |
| **MyBatis-Spring** | <!-- https://mvnrepository.com/artifact/org.mybatis/mybatis-spring -->  <dependency>  <groupId>org.mybatis</groupId>  <artifactId>mybatis-spring</artifactId>  <version>1.3.0</version>  </dependency> |
| **spring-jdbc** | <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-jdbc</artifactId>  <version>${org.springframework-version}</version>  </dependency> |
| **spring-test** | <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-test</artifactId>  <version>${org.springframework-version}</version>  </dependency> |

* + - * 1. **Spring Project에서 root-context.xml 파일의 수정**
* 웹 자원과 관련되지 않은 모든 자원의 설정은 root-context
* Namespaces 탭을 이용해 사용 가능한 XML 태그의 폭을 넓힘

|  |
| --- |
| **pom.xml에 라이브러리 설정 시 root-context.xml에 Namespace 태그 추가 가능** |
|  |
|  |

* + - 1. **MySQL과의 연결을 담당하는 DataSource 설정하기**
* DataSource는 JDBC의 커넥션을 처리하는 기능을 갖고 있다.

|  |
| --- |
| **spring-jdbc 모듈 클래스를 이용해서 DataSource 추가** |
| <!-- spring-jdbc 모듈의 클래스를 이용해서 DataSource 추가 -->  <bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">  <property name="driverClassName" value ="com.mysql.jdbc.Driver"></property>  <property name="url" value="jdbc:mysql://{아이피}:3306/{데이터베이스명}?useSSL=false"></property>  <property name="username" value="{DB아이디}"></property>  <property name="password" value="{DB비밀번호}"></property>  </bean> |

* + - * 1. **DataSource의 테스트 진행**
* 인스턴스 변수의 @Inject 어노테이션 처리된 DataSource는 스프링이 생성해서 주입
* jUnit의 버전 4.11이상인지 확인

|  |
| --- |
| **/test/패키지/DataSourceTest.java** |
| @RunWith(SpringJUnit4ClassRunner.class)  @ContextConfiguration(locations = { "file:src/main/webapp/WEB-INF/spring/\*\*/root-context.xml" })  public class DataSourceTest {  @Inject  private DataSource ds;  @Test  public void testConnection() {  try(Connection con = ds.getConnection()){  System.out.println(con);  }catch(Exception e) {  System.out.println(e);  }  }  } |
|  |

* + - 1. **MyBatis 연결**
* DataSource가 정상적으로 설정 이후 MyBatis와 MySQL을 연동시키는 작업을 한다.
  + - * 1. **SqlSessionFactory 객체 설정**
* SqlSessionFactory는 데이터베이스와의 연결과 SQL의 실행에 대한 모든 것을 가진 중요한 객체
* SqlSessionFactoryBean을 이용하여 SqlSessionFactory를 생성

|  |
| --- |
| **SqlSessionFactoryBean을 root-context.xml에 설정** |
| <!-- MyBatis-Spring 모듈을 받아 SqlSessionFactoryBean을 추가하여 SqlSessionFactory 사용 -->  <bean id="sqlSessionFactory" class="**org.mybatis.spring.SqlSessionFactoryBean**">  <property name="dataSource" ref="dataSource"></property>  </bean> |

* + - * 1. **mybatis-config.xml 파일의 추가**
* XML 파일의 DTD나 XML 스키마 필요
* ‘mybatis config dtd’ 추가

|  |
| --- |
| **src/main/resources/mybatis-config.xml 생성** |
| <?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> |

|  |
| --- |
| **root-context.xml에 mybatis-config.xml 파일 적용 설정** |
| <!-- MyBatis-Spring 모듈을 받아 SqlSessionFactoryBean을 추가하여 SqlSessionFactory 사용 -->  <bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">  <property name="dataSource" ref="dataSource"></property>  **<property name="configLocation" value="classpath:/mybatis-config.xml"></property>**  </bean> |

* + - * 1. **MyBatis의 연결 테스트**
* root-context.xml파일에서 설정한 SqlSessionFactoryBean 설정 시 SqlSessionFactory을 주입

|  |
| --- |
| **test/패키지/MyBatisTest.java** |
| @RunWith(SpringJUnit4ClassRunner.class)  @ContextConfiguration(locations = { "file:src/main/webapp/WEB-INF/spring/\*\*/root-context.xml" })  public class MyBatisTest {    @Inject  private SqlSessionFactory sqlFactory;    @Test  public void testFactory() {  System.out.println(sqlFactory);  }    @Test  public void testSession() {  try(SqlSession session = sqlFactory.openSession()){  System.out.println(session);  }catch(Exception e) {  System.out.println(e);  }  }  } |
|  |

* + - 1. **작업 내역의 정리**

|  |
| --- |
| * MyBatis, MyBatis-Spring, spring-jdbc, spring-test 모듈 추가 * root-context.xml에 DataSource설정 추가 * DataSource 테스트 * MyBatis 연결을 위한 SqlSessionFactoryBean 설정, mybatis-config.xml 파일 생성 * MyBatis 연결 테스트 |

* + 1. **모델 2 방식과 스프링 MVC**
       1. **모델2 패턴의 이해**
          1. **스프링 MVC의 구조에서 컨트롤러 생성**
* **MVC 패턴의 이해(이론)**
* **servlet-context.xml 구조 분석**
* **컨트롤러를 중심으로 테스트(void, String, Object, redirect, JSON)**

**void 리턴 타입**

|  |
| --- |
| **SampleController – return void** |
| @Controller  public class SampleController {  private static final Logger logger = LoggerFactory.getLogger(SampleController.class);    @RequestMapping("doA")  public void doA() {  logger.info("doA called.....");  }  @RequestMapping("doB")  public void doB() {  logger.info("doB called.....");  }    } |
|  |

**String 리턴 타입**

|  |
| --- |
| **SampleController2 – return String** |
| /\*\*  \* @String 리턴타입은 '문자열 + .jsp'파일을 찾아서 실행한다.  \* @author SeokRae  \*  \*/  @Controller  public class SampleController2 {  private static final Logger logger = LoggerFactory.getLogger(SampleController2.class);  @RequestMapping("doC")  public String doC(@ModelAttribute("msg") String msg) {  logger.info("doC called .....");  return "result";  }  } |
|  |
| **result.jsp** |
|  |

**Object 리턴 타입**

|  |
| --- |
| **SampleController3 – return Object** |
| @Controller  public class SampleController3 {  private static final Logger logger = LoggerFactory.getLogger(SampleController3.class);    @RequestMapping("/doD")  public String doD(Model model) {    // make sample data  ProductVO product = new ProductVO("Sample Product", 10000);  logger.info("doD");    // 뷰로 데이터 넘기기  model.addAttribute(product);    return "productDetail";  }  } |
|  |
| **ProductDetail.jsp** |
| <%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>  <!DOCTYPE html>  <html>  <head>  <meta charset="UTF-8">  <title>Insert title here</title>  </head>  <body>  <span>${productVO.name }</span>  <span>${productVO.price }</span>  </body>  </html> |
|  |

**리다이렉트의 경우**

|  |
| --- |
| **SampleController4 – redirect** |
| @Controller  public class SampleController4 {  private static final Logger logger = LoggerFactory.getLogger(SampleController4.class);    @RequestMapping("/doE")  public String doE(RedirectAttributes rttr) {  logger.info("doE called but redirect to /doF.....");  rttr.addFlashAttribute("msg", "This is the Message!! with redirected");  return "redirect:/doF";  }    @RequestMapping("/doF")  public void doF(@ModelAttribute String msg) {  logger.info("doF called ..... " + msg);  }  } |
|  |
|  |

**JSON 리턴 타입**

* JSON데이터에 대한 처리를 간단하게 할 수 있는 Jackson-databind 라이브러리를 추가

|  |
| --- |
| **pom.xml** |
| <!-- https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-databind -->  <dependency>  <groupId>com.fasterxml.jackson.core</groupId>  <artifactId>jackson-databind</artifactId>  <version>2.8.4</version>  </dependency> |

|  |
| --- |
| **SampleController5 – JSON** |
| @Controller  public class SampleController5 {  @RequestMapping("/doJSON")  public @**ResponseBody** ProductVO doJSON() {    ProductVO vo = new ProductVO("샘플상품", 30000);    return vo;  }  } |
|  |

* + - * 1. **WAS없이 컨트롤러 테스트**

|  |
| --- |
| **pom,xml의 servlet 버전 수정** |
| <dependency>  <groupId>javax.servlet</groupId>  <artifactId>javax.servlet-api</artifactId>  <version>3.1.0</version>  </dependency> |
| **test/패키지/SampleControllerTest** |
| @RunWith(SpringJUnit4ClassRunner.class)  @WebAppConfiguration  @ContextConfiguration(locations = { "file:src/main/webapp/WEB-INF/spring/\*\*/\*.xml" })  public class SampleControllerTest {  private static final Logger logger = LoggerFactory.getLogger(SampleControllerTest.class);    @Inject  private WebApplicationContext wac;  private MockMvc mockMvc;  @Before  public void setup() {  this.mockMvc = MockMvcBuilders.webAppContextSetup(this.wac).build();  logger.info("setup .....");  }  @Test  public void testDoA() throws Exception {  mockMvc.perform(MockMvcRequestBuilders.get("/doA"));  }  } |
|  |

* + - 1. **정리**

|  |
| --- |
| * 타입별 컨트롤러 생성 * WAS없이 컨트롤러 테스트 |

* + 1. **스프링 + MyBatis**

|  |
| --- |
| * 테이블 생성, VO 클래스 작성 * DAO 인터페이스 작성 및 구현(@Repository, @Inject SqlSession) * XML Mapper 작성 및 인식을 위한 설정(root-context.xml) |

* + - 1. **테이블 생성 및 개발 준비**
         1. **테이블 생성**

|  |
| --- |
| **Member Table 생성** |
| CREATE TABLE tbl\_member (  userid varchar(50) not null  , userpw varchar(50) not null  , username varchar(50) not null  , email varchar(100)  , regdate timestamp default now()  , updatedate timestamp default now()  , primary key(userid)  ); |

* + - * 1. **도메인 객체 생성**

|  |
| --- |
| **테이블의 속성과 같은 멤버 변수를 가진 도메인(VO) 클래스 생성** |
| public class MemberVO {  private String userid;  private String userpw;  private String username;  private String email;  private Date regdate;  private Date updatedate; |

* + - 1. **DAO 인터페이스 작성**
* 현재 시간을 체크하는 기능
* 데이터 추가 기능

|  |
| --- |
| **MemberDAO 작성** |
| public interface MemberDAO {    public String getTime();    public void insertMember(MemberVO vo);  } |

* + - 1. **XML Mapper 작성**
* **XML로 작성된 Mapper의 위치 src/main/resources/mappers에 위치**
* **XML Mapper 파일 작성, DTD 추가**
* **SQL 작성**

|  |
| --- |
| **mappers/memberMapper.xml 작성** |
| <?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.zerock.mapper.MemberMapper">  <select id="getTime" resultType="string">  select now()  </select>    <insert id="insertMember">  INSERT INTO tbl\_member (  userid  , userpw  , username  , email  )  VALUES (  #{userid}  , #{userpw}  , #{username}  , #{email}  )  </insert>  </mapper> |

* + - * 1. **mybatis-spring에 XML Mapper 설정**

|  |
| --- |
| **root-context.xml 설정 변경** |
| <bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">  <property name="dataSource" ref="dataSource"></property>  <property name="configLocation" value="classpath:/mybatis-config.xml"></property>  <property name="mapperLocations" value="classpath:/mappers/\*\*/\*Mapper.xml"></property>  </bean> |

* + - 1. **DAO 인터페이스의 구현**
* DAO인터페이스와 Mapper를 이용한 구현클래스 작성
* MyBatis에서 DAO를 이용하는 경우에는 SqlSessionTemplate를 이용한다.
  + - * 1. **SqlSessionTemplate 설정**
* 기본적인 트랜잭션의 관리
* 쓰레드 처리의 안전성을 보장
* 데이터베이스의 연결과 종료를 책임진다.
* SqlSessionTemplate는 SqlSessionFactory를 생성자로 주입해서 설정

|  |
| --- |
| **root-context.xml에 SqlSessionTemplate 설정** |
| <bean id="sqlSession" class="org.mybatis.spring.SqlSessionTemplate" destroy-method="clearCache">  <constructor-arg name="sqlSessionFactory" ref="sqlSessionFactory"></constructor-arg>  </bean> |

* + - * 1. **구현 클래스 작성**
* SqlSessionTemplate을 주입하여 사용
* DAO를 스프링에 인식시키기 위한 어노테이션 @Repository

|  |
| --- |
| **MemberDAOImpl** |
| @Repository  public class MemberDAOImpl implements MemberDAO{  @Inject  private SqlSession sqlSession;  /\*\*  \* SqlSession의 기능  \* selectOne  \* selectList  \* selectMap  \* insert  \* update  \* delete  \*  \*/  private static final String namespace = "com.spring.web.MemberMapper";    @Override  public String getTime() {  return sqlSession.selectOne(namespace + ".getTime");  }  @Override  public void insertMember(MemberVO vo) {  sqlSession.insert(namespace + ".insertMember", vo);  }  } |

* + - 1. **스프링에 빈으로 등록**

|  |
| --- |
| **root-context.xml에 컴포넌트 스캔 처리** |
| <!-- 스프링에서 해당 패키지를 스캔, @Repository 어노테이션을 읽음 -->  <context:component-scan base-package="com.spring.persistence"></context:component-scan> |

* + - 1. **테스트 코드 작성**

|  |
| --- |
| **MemberDAOTest** |
| @RunWith(SpringJUnit4ClassRunner.class)  @ContextConfiguration(locations = {"file:src/main/webapp/WEB-INF/spring/\*\*/root-context.xml"})  public class MemberDAOTest {    @Inject  private MemberDAO dao;  @Test  public void testTime() {  System.out.println(dao.getTime());  }    @Test  public void testInsertMember() {  MemberVO vo = new MemberVO();  vo.setUserid("user05");  vo.setUserpw("user05");  vo.setUsername("USER05");  vo.setEmail("user05@aaa.com");    dao.insertMember(vo);  }  } |
|  |

* + - 1. **MyBatis의 로그 log4jdbc-log4j2설정**
* 잘못된 SQL이나 잘못된 속성의 이름으로 예외발생을 위한 로그 설정 라이브러리

|  |
| --- |
| **MyBatis 로그 설정을 위한 log4jdbc-log4j2 라이브러리 추가** |
| <!-- https://mvnrepository.com/artifact/org.bgee.log4jdbc-log4j2/log4jdbc-log4j2-jdbc4 -->  <dependency>  <groupId>org.bgee.log4jdbc-log4j2</groupId>  <artifactId>log4jdbc-log4j2-jdbc4</artifactId>  <version>1.16</version>  </dependency> |

|  |
| --- |
| **root-context.xml에 데이터베이스와 연결되는 드라이버 클래스와 연결 URL 수정** |
| <bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">  <property name="driverClassName" value ="net.sf.log4jdbc.sql.jdbcapi.DriverSpy"></property>  <property name="url" value="jdbc:log4jdbc:mysql://mysqlinstance.c4otsabdm6jt.ap-northeast-2.rds.amazonaws.com:3306/snsdb?useSSL=false"></property>  <property name="username" value="sdba"></property>  <property name="password" value="1234qwer"></property>  </bean> |

* log4jdbc-log4j2가 올바르가 동작하게 하기 위한 로그 관련 설정 파일 추가

|  |
| --- |
| **log4jdbc.log4j2.properties 파일** |
| log4jdbc.spylogdelegator.name=net.sf.log4jdbc.log.slf4j.Slf4jSpyLogDelegator |

|  |
| --- |
| **logback.xml 파일** |
| <?xml version="1.0" encoding="UTF-8"?>  <configuration>  <include resource="org/springframework/boot/logging/logback/base.xml"/>    <!-- log4jdbc-log4j2 -->  <logger name="jdbc.sqlonly" level="DEBUG"/>  <logger name="jdbc.sqltiming" level="INFO"/>  <logger name="jdbc.audit" level="WARN"/>  <logger name="jdbc.resultest" level="ERROR"/>  <logger name="jdbc.resultsettalbe" level="ERROR"/>  <logger name="jdbc.connection" level="INFO"/>    </configuration> |

|  |
| --- |
| **log4jdbc 설정 후 결과** |
|  |

* + - 1. **MyBatis의 #{} 문법**
* 사용자의 등록과 조회 처리

|  |
| --- |
| **MemberDAO 인터페이스** |
| public MemberVO readMember(String userid) throws Exception;    public MemberVO readWithePW(String userid, String userpw) throws Exception; |

|  |
| --- |
| **MemberDAOImpl** |
| @Repository  public class MemberDAOImpl implements MemberDAO{  @Inject  private SqlSession sqlSession;  /\*\*  \* SqlSession의 기능  \* selectOne  \* selectList  \* selectMap  \* insert  \* update  \* delete  \*  \*/  private static final String namespace = "com.spring.web.MemberMapper";    @Override  public String getTime() {  return sqlSession.selectOne(namespace + ".getTime");  }  @Override  public void insertMember(MemberVO vo) {  sqlSession.insert(namespace + ".insertMember", vo);  }  @Override  public MemberVO readMember(String userid) throws Exception {  return (MemberVO) sqlSession.selectOne(namespace + ".selectMember", userid);  }  @Override  public MemberVO readWithePW(String userid, String userpw) throws Exception {  Map<String, Object> paramMap = new HashMap<String, Object>();    paramMap.put("userid", userid);  paramMap.put("userpw", userpw);  return sqlSession.selectOne(namespace + ".readWithPW", paramMap);  }  } |

* 사용자 조회 쿼리문 등록

|  |
| --- |
| **memberMapper.xml** |
| <select id="selectMember" resultType="com.spring.domain.MemberVO">  SELECT  \*  FROM tbl\_member  WHERE userid = #{userid}  </select>  <select id="readWithPW" resultType="com.spring.domain.MemberVO">  SELECT  \*  FROM tbl\_member  WHERE userid = #{userid}  AND userpw = #{userpw}  </select> |

* + 1. **정리**

|  |
| --- |
| 1. 개발환경 설정 2. MySQL 설정 3. MyBatis 설정 4. Spring + MyBatis 설정 |