

프레임워크 프로그래밍

4. 스프링 JDBC

(데이터 액세스 층의 설계와 구현)

2020. 3. 10

컴퓨터공학

Learning Objectives



1. 데이터 액세스 층에 대해 이해한다.
2. 스프링 JDBC를 이해한다.
3. JUnit 을 이용한 스프링 유닛 테스트를 이해한다.

4. 스프링 JDBC

4.1 데이터 액세스 층

➤ 데이터 액세스 층의 역할

- 데이터 액세스 처리를 비즈니스 로직 층에서 분리하는 것

➤ DAO(Data Access Object)

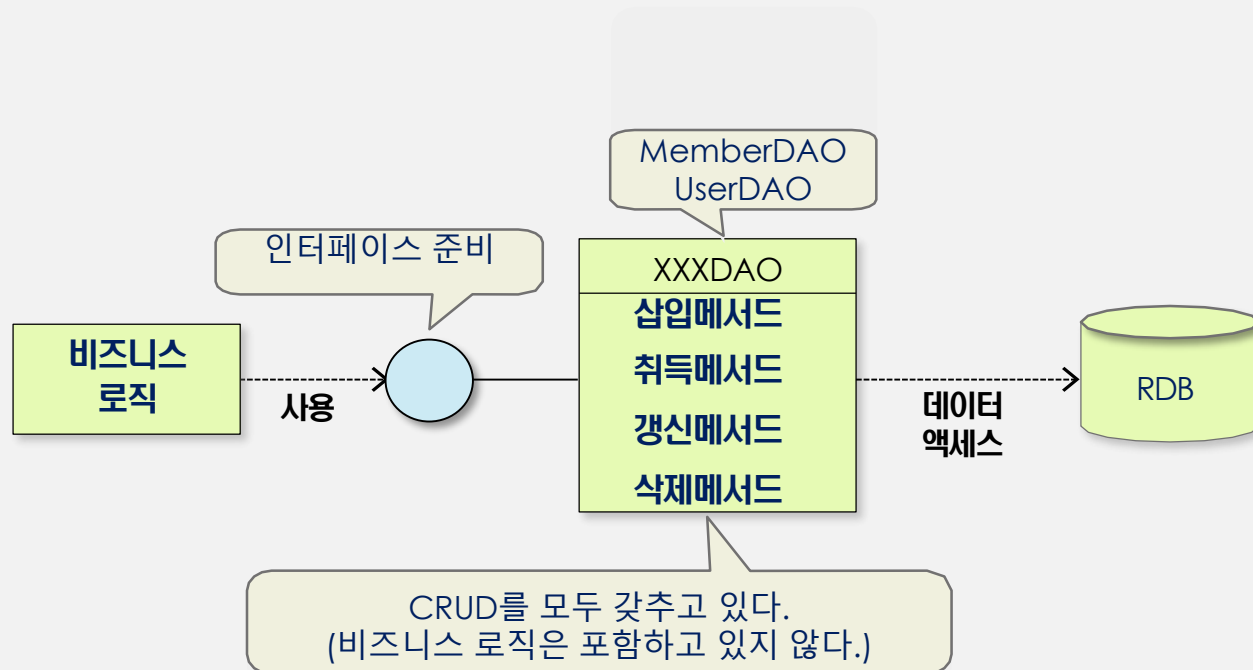
- 데이터 액세스 처리에 특화된 오브젝트
- 선 마이크로시스템즈 (현재의 오라클)가 제창한 J2EE 패턴 중 하나인 DAO 패턴 용어

4. 스프링 JDBC

4.1 데이터 액세스 층

➤ DAO(Data Access Obejct) 패턴

- DAO 패턴은 데이터 취득과 변경에 데이터 처리를 DAO 오브젝트로 분리하는 패턴

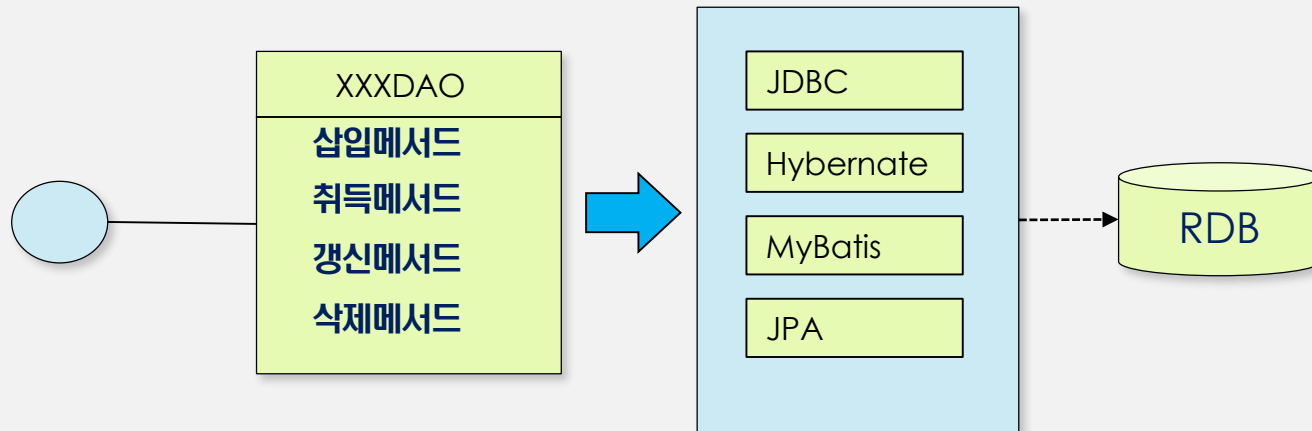


4. 스프링 JDBC

4.1 데이터 액세스 층

➤ 자바의 데이터 액세스 기술

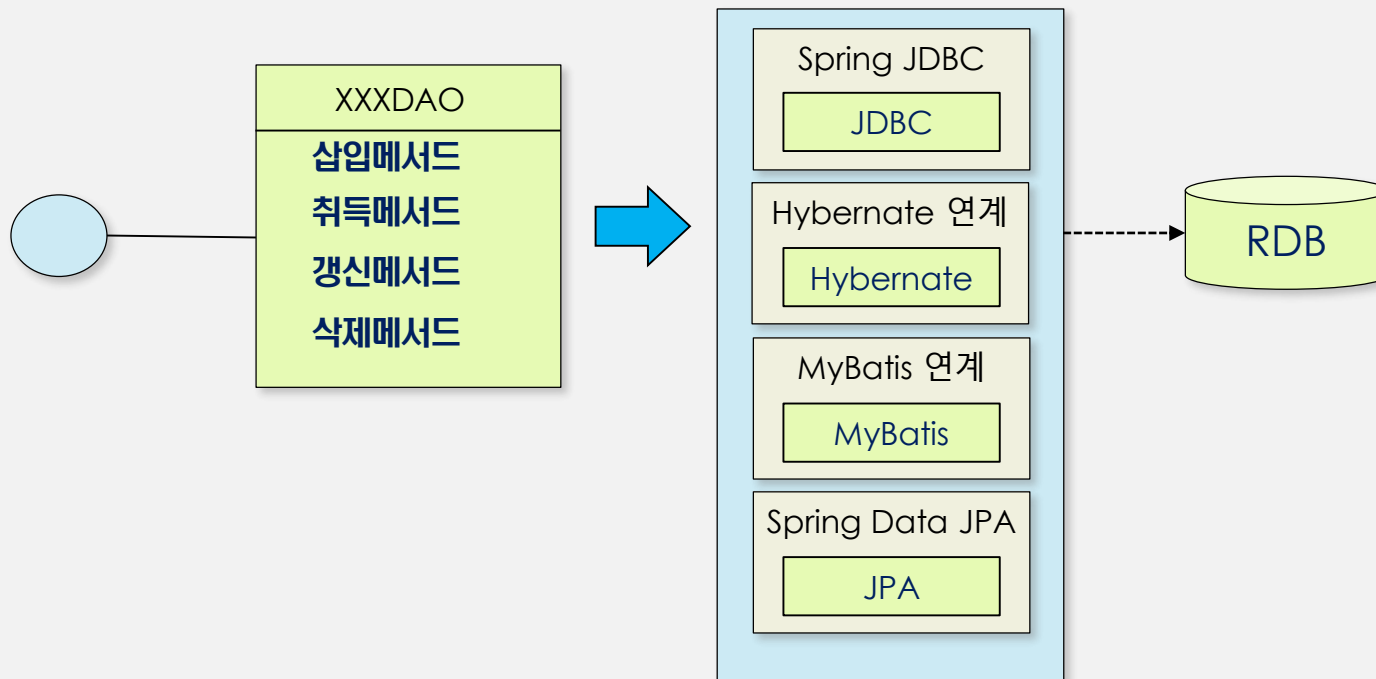
- 데이터 액세스 기술을 처리하는 자바 기술에는 여러가지 존재
- JDBC, Hibernate, MyBatis, JPA



4. 스프링 JDBC

4.1 데이터 액세스 층

- DAO 구현에서 스프링의 역할
 - 데이터 액세스 기술을 쉽게 사용하기 위한 연계 기능을 제공

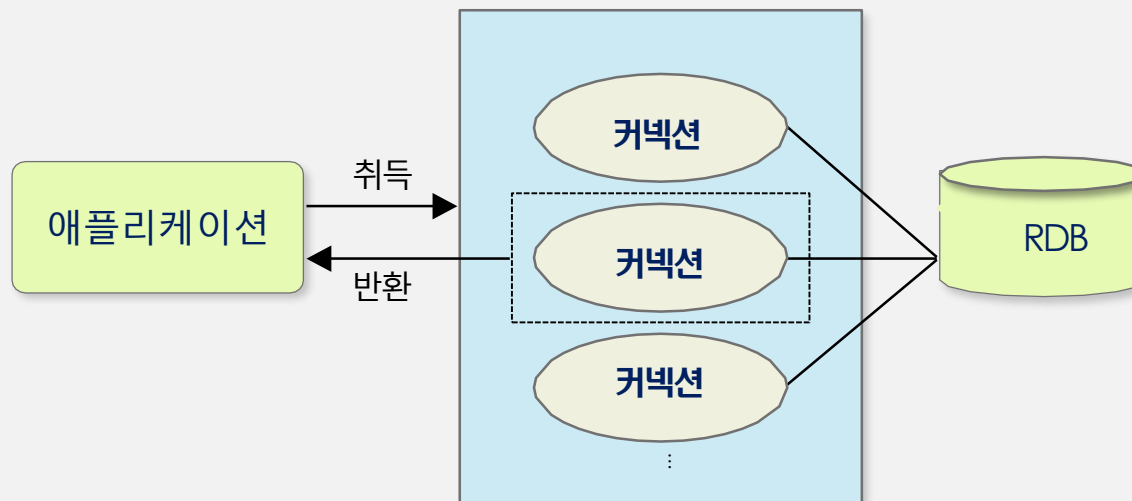


4. 스프링 JDBC

4.1 데이터 액세스 층

➤ 데이터 소스

- 데이터 액세스 기술 종류와 상관없이 데이터베이스 접속을 관리해주는 인터페이스
- 업무용 어플리케이션은 **커넥션 풀**에 의해 커넥션 오브젝트를 재사용



4. 스프링 JDBC

4.1 데이터 액세스 층

➤ 데이터 소스 구현

- 서드 파티가 제공하는 데이터 소스

- [Apache Commons DBCP](#)
- 의존 관계 설정

```
POM.XML
<dependency>
    <groupId>commons-dbcp</groupId>
    <artifactId>commons-dbcp</artifactId>
    <version>1.4</version>
</dependency>
```

- 애플리케이션 서버가 제공하는 데이터 소스

- Tomcat, Oracle WebLogic, IBM WebSphere, JBoss 등
- JNDI를 이용하여 데이터 소스 오브젝트 취득

- 임베디드 데이터베이스가 제공하는 스프링 지원 데이터 소스

- HSQLDB(<http://www.hsqldb.org>)
- H2(<http://www.h2database.com>)
- Apache Derby(<http://db.apache.org/derby/>)

4. 스프링 JDBC

4.1 데이터 액세스 층

➤ 서드 파티가 제공하는 데이터 소스 (Apache Commons DBCP)

- Bean 정의 파일에 DBCP 데이터 소스 설정

applicationContext.xml

```
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">
  <property name="driverClassName" value="com.mysql.cj.jdbc.Driver" />
  <property name="url" value="jdbc:mysql://127.0.0.1:3306/springdb?
    allowPublicKeyRetrieval=true&useSSL=false&serverTimezone=UTC" />
  <property name="username" value="spring" />
  <property name="password" value="passwd" />
  <property name="maxActive" value="5" />
</bean>
```

4. 스프링 JDBC

4.2 스프링 JDBC

➤ JDBC 이용의 문제점

- 대량의 소스 코드를 기술
- 다양한 에러 원인을 파악하기 위한 코딩이 필요
- 데이터베이스 제품마다 에러 코드가 달라서 코드의 일관성 유지가 어려움

4. 스프링 JDBC

4.2 스프링 JDBC

```
public ArrayList<StudentVO> readList() {
    connect();
    ArrayList<StudentVO> studentlist = new ArrayList<StudentVO>();
    String sql = "select id,passwd,username,snum,depart,mobile,email from
student";
    try {
        pstmt = conn.prepareStatement(sql);
        ResultSet rs = pstmt.executeQuery();
        while(rs.next()) {
            StudentVO student = new StudentVO();
            student.setId(rs.getString("id"));
            student.setPasswd(rs.getString("passwd"));
            student.setUsername(rs.getString("username"));
            student.setSnum(rs.getString("snum"));
            student.setDepart(rs.getString("depart"));
            student.setMobile(rs.getString("mobile"));
            student.setEmail(rs.getString("email"));

            studentlist.add(student);
        }
        rs.close();
    } catch(SQLException e) {
        e.printStackTrace();
    } finally {
        disconnect();
    }
    return studentlist;
}
```

```
void connect() {
    try {
        Class.forName(jdbc_driver);
        conn = DriverManager.getConnection(jdbc_url, "spring","passwd");
    } catch(Exception e) {
        e.printStackTrace();
    }
}
```

JDBC 코드

4. 스프링 JDBC

4.2 스프링 JDBC

➤ 스프링 JDBC

- JDBC를 래핑한 API를 제공해 소스 코드를 단순화
- JDBC를 직접 사용할 때 발생하는 장황한 코드를 은닉
 - 커넥션 연결 종료
 - SQL 문의 실행
 - SQL 문 실행 결과 행에 대한 반복 처리
 - 예외 처리

➤ 스프링 JDBC기 제공하는 중요 템플릿

- JdbcTemplate
- NamedParameterTemplate

4. 스프링 JDBC

4.2 스프링 JDBC

➤ JdbcTemplate 클래스 제공 메서드

메서드 명	설명
queryForObject	하나의 결과 레코드 중에서 하나의 컬럼 값을 가져올 때 RowMapper와 함께 사용하면 하나의 레코드 정보를 객체에 매핑
queryForMap	하나의 결과 레코드 정보를 Map 형태로 매핑할 수 있음
queryForList	여러 개의 Map 형태의 결과 레코드를 다룰 수 있음
query	여러 개의 레코드를 객체로 변환하여 처리(ResultSetExtractor)
update	데이터의 변경(INSERT, UPDATE, DELETE)을 실행할 때

4. 스프링 JDBC

4.2 스프링 JDBC

- 템플릿 클래스의 오브젝트 생성과 인젝션
 - 템플릿 클래스를 XML 파일에 Bean으로 정의

```
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">
  <property name="driverClassName" value="com.mysql.cj.jdbc.Driver" />
  <property name="url" value="jdbc:mysql://127.0.0.1:3306/springdb?
    allowPublicKeyRetrieval=true&useSSL=false&serverTimezone=UTC" />
  <property name="username" value="spring" />
  <property name="password" value="passwd" />
  <property name="maxActive" value="5" />
</bean>

<bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
  <constructor-arg ref="dataSource" />
</bean>

<bean class="org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate">
  <constructor-arg ref="dataSource" />
</bean>
```

4. 스프링 JDBC

4.2 스프링 JDBC

➤ SELECT 문 : 취득 결과가 레코드 건수 또는 특정 컬럼만 취득할 경우

- queryForObject 메서드 사용 예제 1

- 제1인수 : SQL 문자열
- 제2인수 : 반환형 클래스 오브젝트 (int)

```
JdbcTemplate jdbcTemplate = ctx.getBean(JdbcTemplate.class);  
int count = jdbcTemplate.queryForObject(  
    "SELECT COUNT(*) FROM STUDENT", Integer.class);
```

- queryForObject 메서드 사용 예제 2

- 제1인수 : SQL 문자열, 제2인수 : 반환형 클래스 오브젝트 (String)
- 제3인수 : 파라미터 값

```
String name = jdbcTemplate.queryForObject(  
    "SELECT username FROM STUDENT WHERE id= ?", String.class, id);
```

4. 스프링 JDBC

4.2 스프링 JDBC

- SELECT 문 : 취득 결과가 한 레코드 값을 취득할 경우
 - `queryForMap` 메서드 : 한 레코드 값을 Map(컬럼 이름을 키로 값을 저장) 데이터로

```
Map<String, Object> student = jdbcTemplate.queryForMap(  
    "SELECT * FROM STUDENT WHERE id= ?", id);  
String name = (String)student.get("username");
```

- `queryForList` 메서드 : 여러 레코드 값을 Map 데이터로

```
List<Map<String, Object>> studentList = jdbcTemplate.queryForMap(  
    "SELECT * FROM STUDENT ");
```


4. 스프링 JDBC

4.2 스프링 JDBC

➤ SELECT 문 : 도메인으로 변환할 경우

- queryForObject 메서드와 query 메서드를 이용한다.
- queryForObject 메서드 – 한 레코드를 가져올 때
 - 제1인수 : SELECT 문, 제3인수 : SELECT 문의 파라미터
 - 제2인수 : 도메인 자동 변환을 위한 스프링 제공 클래스 `BeanPropertyRowMapper`
 - `BeanPropertyRowMapper`를 사용할 경우 StudentVO의 프로퍼티명과 테이블 컬럼명이 같아야 함. 그렇지 않을 경우는 `RoWMapper` 인터페이스를 구현해서 `StudentVO`로 변환 처리

```
public StudentVO read(String id) throws Exception {
    StudentVO vo = null;
    try {
        vo = jdbcTemplate.queryForObject(
            "SELECT * FROM STUDENT WHERE ID=?",
            new BeanPropertyRowMapper<StudentVO>(StudentVO.class), id);
    }
    catch(EmptyResultDataAccessException e) {
        return vo;
    }
    return vo;
}
```

4. 스프링 JDBC

4.2 스프링 JDBC

➤ SELECT 문 : 도메인으로 변환할 경우

- query 메서드 - 여러 레코드를 가져올 때

- RowMapper 인터페이스를 구현한 익명 클래스를 정의
- 클래스내 mapRow() 추상 메서드를 정의

```
public List<StudentVO> readList() throws Exception {
    List<StudentVO> studentlist = jdbcTemplate.query(
        "SELECT * FROM STUDENT",
        new RowMapper<StudentVO>() {
            public StudentVO mapRow(ResultSet rs, int rowNum) throws SQLException {
                StudentVO vo = new StudentVO();
                vo.setId(rs.getString("ID"));
                vo.setPasswd(rs.getString("PASSWD"));
                vo.setUsername(rs.getString("USERNAME"));
                vo.setSnum(rs.getString("SNUM"));
                vo.setDepart(rs.getString("DEPART"));
                vo.setMobile(rs.getString("MOBILE"));
                vo.setEmail(rs.getString("EMAIL"));
                return vo;
            }
        }
    );
    return studentlist;
}
```

4. 스프링 JDBC

4.2 스프링 JDBC

➤ INSERT/UPDATE/DELETE 문

- update 메서드만을 사용

• INSERT 문

```
StudentVO vo;  
jdbcTemplate.update(  
    "INSERT INTO STUDENT (ID, PASSWD, USERNAME, SNUM, DEPART, MOBILE, EMAIL)  
    VALUES (?, ?, ?, ?, ?, ?, ?) ", vo.getId(), vo.getPasswd(), vo.getUsername(),  
    vo.getSnum(), vo.getDepart(), vo.getMobile(), vo.getEmail()  
);
```

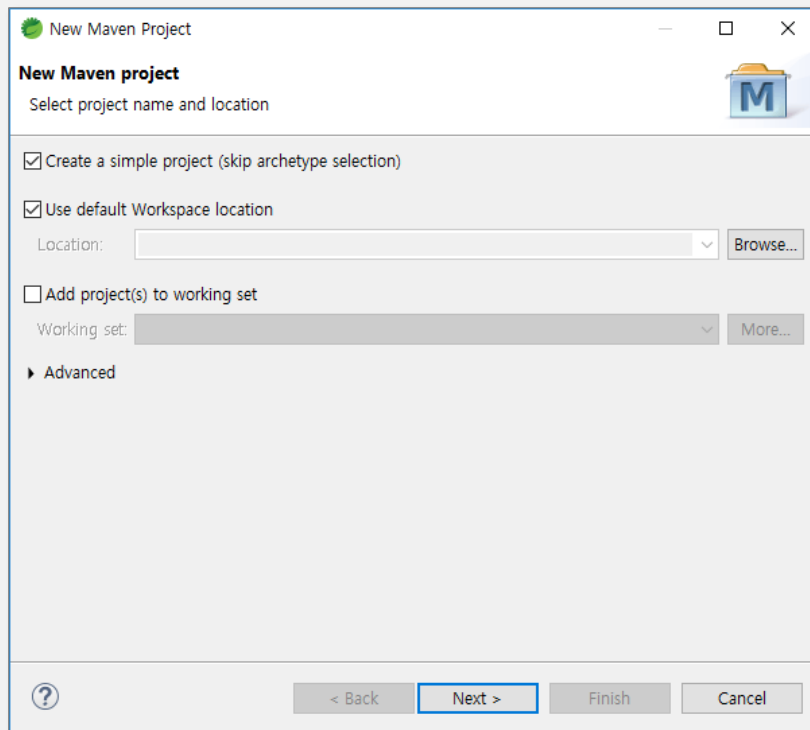
• DELETE 문

```
StudentVO vo;  
jdbcTemplate.update( "DELETE FROM STUDENT WHERE ID=? ", vo.getId() );
```

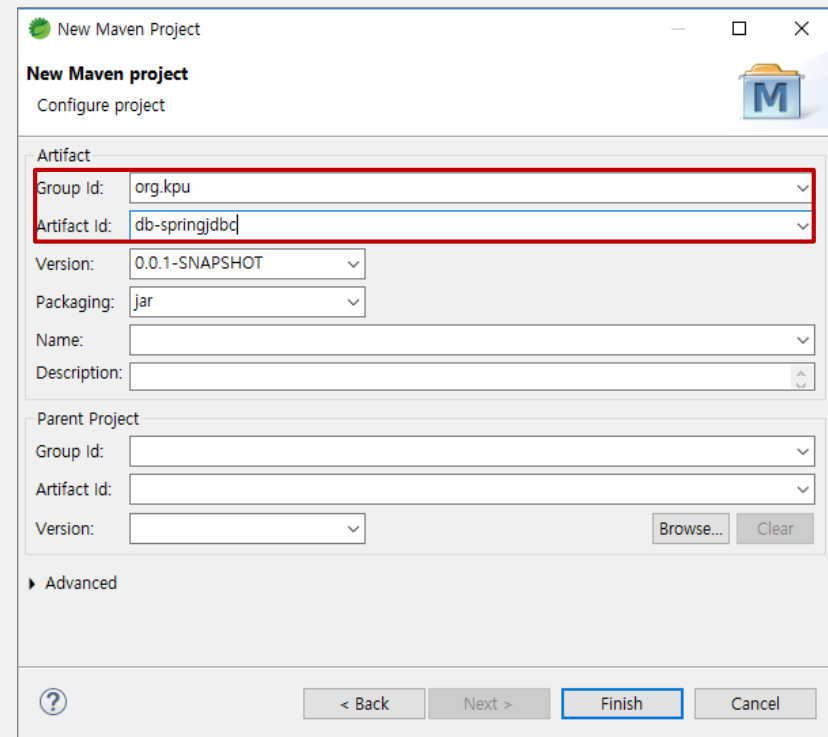
4. 스프링 JDBC

4.2 스프링 JDBC

- db-springjdbc 예제 프로젝트 만들기
 - [File]-[New]-[Maven Project] 메뉴



The 'New Maven Project' dialog box, Step 1: Select project name and location. It features a title bar with a green icon and standard window controls. The main area has a header 'New Maven project' and a subtitle 'Select project name and location'. Below this, there are three checked options: 'Create a simple project (skip archetype selection)', 'Use default Workspace location', and 'Add project(s) to working set'. The 'Location' field is empty with a 'Browse...' button. The 'Working set' field is also empty with a 'More...' button. An 'Advanced' section is collapsed. At the bottom, there are navigation buttons: '< Back', 'Next >', 'Finish', and 'Cancel'.



The 'New Maven Project' dialog box, Step 2: Configure project. It features a title bar with a green icon and standard window controls. The main area has a header 'New Maven project' and a subtitle 'Configure project'. Below this, there are several fields for project configuration: 'Artifact' section with 'Group Id' (org.kpu) and 'Artifact Id' (db-springjdbc) highlighted by a red box, 'Version' (0.0.1-SNAPSHOT), 'Packaging' (jar), 'Name', and 'Description'. The 'Parent Project' section has 'Group Id', 'Artifact Id', and 'Version' fields, with 'Browse...' and 'Clear' buttons. An 'Advanced' section is collapsed. At the bottom, there are navigation buttons: '< Back', 'Next >', 'Finish', and 'Cancel'.

4. 스프링 JDBC

4.2 스프링 JDBC

➤ db-springjdbc 예제 프로젝트 구성

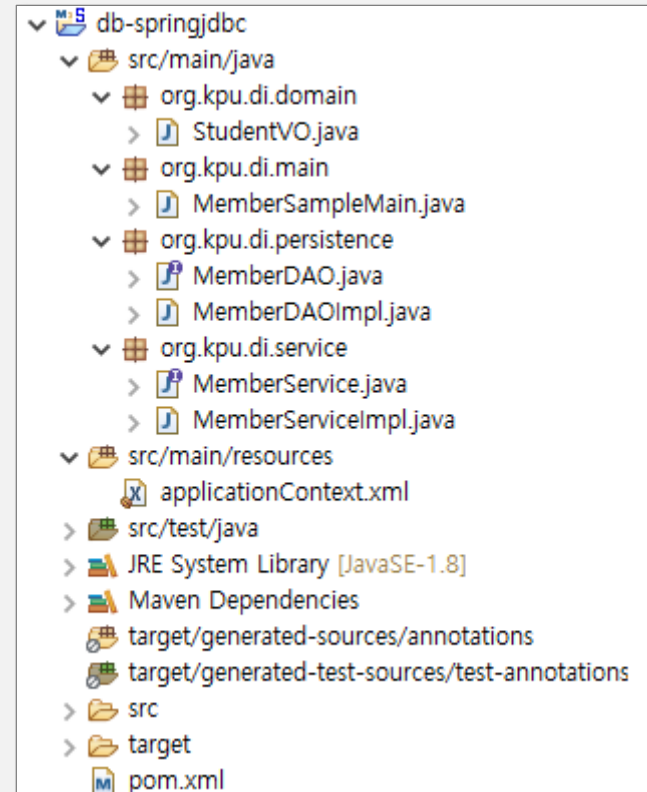
- [Properties]-[Java Build Path] : 1.8 확인 및 필요시 변경

- POM.xml 설정

- spring-context -> 5.2.3
- spring-jdbc -> 5.2.3 (뒷 페이지 참조)
 - commons-dbcp -> 1.4
 - mysql-connector-java -> 8.0.18
- sl4j-api -> 1.7.30
- logback-classic -> 1.2.3

- src/main/resources/applicationContext.xml 생성
네임스페이스 추가 (beans, context, c, jdbc)

- StudentVO.java 기존 프로젝트에서 복사



4. 스프링 JDBC

4.2 스프링 JDBC

➤ POM.xml 추가 설정

POM.XML

```
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-jdbc</artifactId>
  <version>${spring-framework.version}</version>
</dependency>
<dependency>
  <groupId>commons-dbcp</groupId>
  <artifactId>commons-dbcp</artifactId>
  <version>1.4</version>
</dependency>
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>8.0.16</version>
</dependency>
```

```
<spring-framework.version>5.2.3.RELEASE</spring-framework.version>
```

4. 스프링 JDBC

4.2 스프링 JDBC

➤ db-springjdbc 예제 프로젝트

- MemberDAO.java

```
package org.kpu.di.persistence;
import java.util.List;
import org.kpu.di.domain.StudentVO;

public interface MemberDAO {
    public void add(StudentVO student) throws Exception;
    public StudentVO read(String id) throws Exception;
    public List<StudentVO> readList() throws Exception;
}
```


4. 스프링 JDBC

4.2 스프링 JDBC

➤ db-springjdbc 예제 프로젝트

- MemberDAOImpl.java

```
package org.kpu.di.persistence;

import java.util.List;
import java.sql.ResultSet;
import java.sql.SQLException;

import org.kpu.di.domain.StudentVO;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.dao.EmptyResultDataAccessException;
import org.springframework.jdbc.core.BeanPropertyRowMapper;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.RowMapper;
import org.springframework.stereotype.Component;

@Component
public class MemberDAOImpl implements MemberDAO {

    @Autowired
    JdbcTemplate jdbcTemplate;

    public void add(StudentVO vo) throws Exception {
        jdbcTemplate.update("INSERT INTO STUDENT (ID, PASSWD, USERNAME, SNUM,
        DEPART, MOBILE, EMAIL) VALUES (?, ?, ?, ?, ?, ?, ?)", vo.getId(), vo.getPasswd(),
        vo.getUsername(), vo.getSnum(), vo.getDepart(), vo.getMobile(), vo.getEmail());
    }
}
```


4. 스프링 JDBC

4.2 스프링 JDBC

```
public StudentVO read(String id) throws Exception {
    StudentVO vo = null;
    try {
        vo = jdbcTemplate.queryForObject( "SELECT * FROM STUDENT WHERE ID=?"
            , new BeanPropertyRowMapper<StudentVO>(StudentVO.class), id);
    }
    catch(EmptyResultDataAccessException e) {
        return vo;
    }
    return vo;
}

public List<StudentVO> readList() throws Exception {
    List<StudentVO> studentlist = jdbcTemplate.query(
        "SELECT * FROM STUDENT",
        new RowMapper<StudentVO>() {
            public StudentVO mapRow(ResultSet rs, int rowNum) throws SQLException {
                StudentVO vo = new StudentVO();
                vo.setId(rs.getString("ID"));
                vo.setPasswd(rs.getString("PASSWD"));
                vo.setUsername(rs.getString("USERNAME"));
                vo.setSnum(rs.getString("SNUM"));
                vo.setDepart(rs.getString("DEPART"));
                vo.setMobile(rs.getString("MOBILE"));
                vo.setEmail(rs.getString("EMAIL"));
                return vo;
            }
        }
    );
    return studentlist;
}
```

4. 스프링 JDBC

4.2 스프링 JDBC

- db-springjdbc 예제 프로젝트
 - MemberService.java

```
package org.kpu.di.service;

import java.util.List;

import org.kpu.di.domain.StudentVO;

public interface MemberService {
    public StudentVO readMember(String id) throws Exception;
    public void addMember(StudentVO student) throws Exception;
    public List<StudentVO> readMemberList() throws Exception;
}
```

4. 스프링 JDBC

4.2 스프링 JDBC

➤ db-springjdbc 예제 프로젝트

- MemberServiceImpl.java

```
package org.kpu.di.service;
import java.util.List;
import org.kpu.di.domain.StudentVO;
import org.kpu.di.persistence.MemberDAO;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

@Component
public class MemberServiceImpl implements MemberService {
    @Autowired
    private MemberDAO memberDAO;

    public StudentVO readMember(String id) throws Exception {
        return memberDAO.read(id);
    }
    public void addMember(StudentVO student) throws Exception {
        memberDAO.add(student);
    }
    public List<StudentVO> readMemberList() throws Exception{
        return memberDAO.readList();
    }
}
```

4. 스프링 JDBC

4.2 스프링 JDBC

- db-springjdbc 예제 프로젝트
 - applicationContext.xml

applicationContext.xml

```
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">
    <property name="driverClassName" value="com.mysql.cj.jdbc.Driver" />
    <property name="url" value="jdbc:mysql://127.0.0.1:3306/springdb?
        allowPublicKeyRetrieval=true&useSSL=false&serverTimezone=UTC" />
    <property name="username" value="spring" />
    <property name="password" value="passwd" />
    <property name="maxActive" value="5" />
</bean>

<bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
    <constructor-arg ref="dataSource" />
</bean>

<bean class="org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate">
    <constructor-arg ref="dataSource" />
</bean>

<context:component-scan base-package="org.kpu.di.persistence"></context:component-scan>
<context:component-scan base-package="org.kpu.di.service"></context:component-scan>
```

4. 스프링 JDBC

4.2 스프링 JDBC

➤ db-springjdbc 예제 프로젝트

- MemberSampleMain.java

```
package org.kpu.di.main;
import java.util.List;
import org.kpu.di.domain.StudentVO;
import org.kpu.di.service.MemberService;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.dao.DataAccessException;
import org.springframework.jdbc.core.JdbcTemplate;

public class MemberSampleMain {

    private static ApplicationContext ctx = null;
    public static void main(String[] args) throws Exception {
        System.out.println("안녕하세요 DB-SPRINGJDBC");
        ctx = new ClassPathXmlApplicationContext("classpath:applicationContext.xml");
        MemberService memberService = ctx.getBean(MemberService.class); // by Class name
        String strID = "hansol"; StudentVO vo = new StudentVO();
        vo.setID(strID); vo.setPasswd(strID); vo.setUsername(strID); vo.setSnum(strID);



서비스 코드



    }
}
```

4. 스프링 JDBC

4.2 스프링 JDBC

➤ db-springjdbc 예제 프로젝트

- MemberSampleMain.java (서비스 코드)

```
try {  
    // memberService.addMember(vo);  
    StudentVO member = memberService.readMember(strID);  
    System.out.println(member);  
  
    List<StudentVO> list = memberService.readMemberList();  
    for(StudentVO svo : list) {  
        System.out.println(svo);  
    }  
  
} catch(DataAccessException e) {  
    System.out.println(e);  
  
} finally { //Check Count  
    JdbcTemplate jdbcTemplate = ctx.getBean(JdbcTemplate.class);  
    int count = jdbcTemplate.queryForObject("SELECT COUNT(*) FROM STUDENT", Integer.class);  
    System.out.println(count);  
}
```

4.3 JUnit 을 이용한 스프링 유닛 테스트

➤ 스프링 테스트(Spring-Test)

- 스프링 프레임워크에서 만든 클래스(@Controller, @Service, @Repository, @Component 등이 붙은 클래스)를 테스트하는 모듈
- 단위 테스트, 통합 테스트를 지원하기 위한 매커니즘이나 편리한 기능을 제공
 - Junit 테스트 프레임워크를 사용하여 스프링 DI 컨테이너를 동작시키는 기능
 - 트랜잭션 테스트를 상황에 맞게 제어하는 기능
 - 애플리케이션 서버를 사용하지 않고 스프링 MVC 동작을 재현하는 기능
 - RestTemplate을 이용해 HTTP 요청에 대한 임의 응답을 보내는 기능

4. 스프링 JDBC

4.3 JUnit 을 이용한 스프링 유닛 테스트

➤ 스프링 테스트(Spring-Test) – POM.xml 설정

POM.XML

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-test</artifactId>
    <version>${spring-framework.version}</version>
    <scope>test</scope>
</dependency>

<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>${junit.version}</version>
    <scope>test</scope>
</dependency>
```

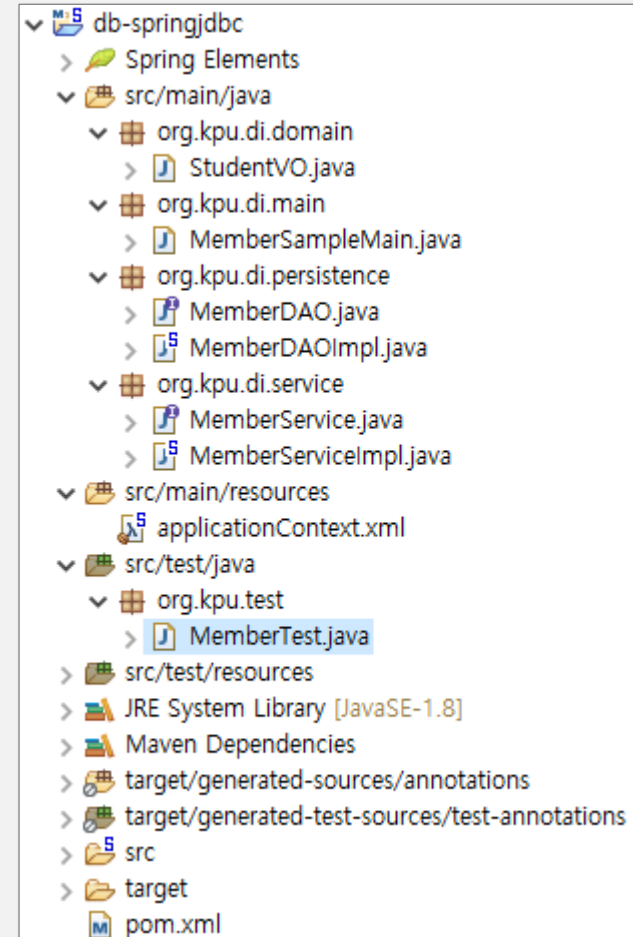
```
<spring-framework.version>5.2.3.RELEASE</spring-framework.version>
<junit.version>4.13</junit.version>
```


4. 스프링 JDBC

4.3 JUnit 을 이용한 스프링 유닛 테스트

➤ 빈 테스트 환경

- src/test/java 디렉토리 내 테스트 클래스 작성
- `MemberTest.java`



4.3 JUnit 을 이용한 스프링 유닛 테스트

➤ 테스트 케이스 작성 : `memberService.readMember("hansol");` 테스트

```
package org.kpu.test;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.kpu.di.domain.StudentVO;
import org.kpu.di.service.MemberService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations = "classpath:/applicationContext.xml")

public class MemberTest {

    @Autowired
    MemberService memberService;

    @Test
    public void testReadMember( ) throws Exception {
        StudentVO member = memberService.readMember("hansol");
        System.out.println(member);
    }
}
```

MemberTest.java

테스트용 DI 컨테이너를 동작시키기 위한 Runner 클래스

테스트용 DI 컨테이너가 사용하는 설정 파일

DI 컨테이너에 등록할 테스트 대상 빈을 주입

테스트 메서드 선언

테스트 메서드 실행

4. 스프링 JDBC

4.3 JUnit 을 이용한 스프링 유닛 테스트

➤ 테스트 케이스 실행

- MemberTest.java

db-springjdbc

- Spring Elements
- src/main/java
- src/main/resources
- src/test/java
 - org.kpu.test
 - MemberTest.java
 - src/test/resources
 - JRE System Library [JavaSE-1.8]
 - Maven Dependencies
 - target/generated-sources/annotations
 - target/generated-test-sources/test-annotations
 - src
 - target

Run As

- 1 Run on Server Alt+Shift+X, R
- 2 JUnit Test Alt+Shift+X, T
- Run Configurations...

Package Explorer JUnit

Finished after 0.533 seconds

Runs: 1/1 Errors: 0 Failures: 0

org.kpu.test.MemberTest [Runner: JUnit 4] (0.163 s)

testReadMember (0.163 s)

<terminated> MemberTest [JUnit] C:\Program Files\Java\jdk1.8.0_241\bin\javaw.exe (2020. 3. 10. 오후 2:27:50)

```
14:27:51.624 [main] DEBUG org.springframework.jdbc.core.BeanPropertyRowMapper - Mapping column 'username' to property 'username' of type 'java.lang.String'
14:27:51.624 [main] DEBUG org.springframework.jdbc.core.BeanPropertyRowMapper - Mapping column 'snum' to property 'snum' of type 'java.lang.String'
14:27:51.624 [main] DEBUG org.springframework.jdbc.core.BeanPropertyRowMapper - Mapping column 'depart' to property 'depart' of type 'java.lang.String'
14:27:51.624 [main] DEBUG org.springframework.jdbc.core.BeanPropertyRowMapper - Mapping column 'mobile' to property 'mobile' of type 'java.lang.String'
14:27:51.624 [main] DEBUG org.springframework.jdbc.core.BeanPropertyRowMapper - Mapping column 'email' to property 'email' of type 'java.lang.String'
StudentVO [id=hansol, passwd=hansol, username=정환솔, snum=2012152040, depart=컴퓨터(卒), mobile=01111119999, email=2012152040@kpu.ac.kr]
14:27:51.628 [main] DEBUG org.springframework.test.context.cache.DefaultCacheAwareContextLoaderDelegate - Retrieved ApplicationContext [527446182] from cache with key [[MergedContextConfigur
14:27:51.628 [main] DEBUG org.springframework.test.context.cache - Spring test ApplicationContext cache statistics: [DefaultContextCache@2aceadd4 size = 1, maxSize = 32, parentContextCount =
14:27:51.629 [main] DEBUG org.springframework.test.context.cache.DefaultCacheAwareContextLoaderDelegate - Retrieved ApplicationContext [527446182] from cache with key [[MergedContextConfigur
14:27:51.629 [main] DEBUG org.springframework.test.context.cache - Spring test ApplicationContext cache statistics: [DefaultContextCache@2aceadd4 size = 1, maxSize = 32, parentContextCount =
14:27:51.630 [main] DEBUG org.springframework.test.context.support.AbstractDirtiesContextTestExecutionListener - After test method: context [DefaultTestContext@61f8bee4 testClass = MemberTes
14:27:51.631 [main] DEBUG org.springframework.test.context.cache.DefaultCacheAwareContextLoaderDelegate - Retrieved ApplicationContext [527446182] from cache with key [[MergedContextConfigur
14:27:51.631 [main] DEBUG org.springframework.test.context.cache - Spring test ApplicationContext cache statistics: [DefaultContextCache@2aceadd4 size = 1, maxSize = 32, parentContextCount =
14:27:51.632 [main] DEBUG org.springframework.test.context.support.AbstractDirtiesContextTestExecutionListener - After test class: context [DefaultTestContext@61f8bee4 testClass = MemberTest
14:27:51.636 [SpringContextShutdownHook] DEBUG org.springframework.context.support.GenericApplicationContext - Closing org.springframework.context.support.GenericApplicationContext@1f7030a6,
```

4. 스프링 JDBC

4.3 JUnit 을 이용한 스프링 유닛 테스트

➤ 다른 테스트 케이스

- memberService.addMember(); memberService.readMember(); 테스트

```
public class MemberTest {  
    @Autowired  
    MemberService memberService;  
    // @Test  
    public void testAddMember( ) throws Exception {  
        String strID = "JUnit";  
        StudentVO vo = new StudentVO();  
        vo.setID(strID);  
        vo.setPasswd(strID);  
        vo.setUsername(strID);  
        vo.setSnum(strID);  
        memberService.addMember(vo);  
        StudentVO member = memberService.readMember("JUnit");  
        System.out.println(member);  
    }  
    @Test  
    public void testReadMember( ) throws Exception {  
        StudentVO member = memberService.readMember("hansol");  
        System.out.println(member);  
    }  
}
```