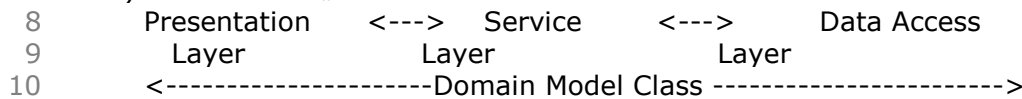


1. Spring JDBC Architecture

1)개요

- 대부분의 중/대규모 웹 어플리케이션은 효율적인 개발 및 유지 보수를 위하여 계층화(Layering)하여 개발하는 것이 원칙이다.
- 사용자관리 프로젝트 아키텍처에서 기본적으로 가지는 계층은 **Presentation Layer, Service Layer, Data Access Layer** 3계층과 모든 계층에서 사용되는 **Domain Model Class**로 구성되어 있다.
- 각각의 계층은 계층마다 독립적으로 분리하여 구현하는 것이 가능해야 하며, 각 계층에서 담당해야 할 기능들이 있다.

2)Architecture 개요



- 위의 3가지 계층은 독립적으로 분리할 수 있도록 구현해야 하며, 일반적으로 각 계층 사이에서는 **interface**를 이용하여 통신하는 것이 일반적이다.

2. Presentation Layer

- 1)Browser상의 웹 클라이언트의 요청 및 응답을 처리
- 2)상위계층(서비스 계층, 데이터 액세스 계층)에서 발생하는 **Exception**에 대한 처리
- 3)최종 UI에서 표현해야 할 도메인 모델을 사용
- 4)최종 UI에서 입력한 데이터에 대한 유효성 검증(**Validation**) 기능을 제공
- 5)비즈니스 로직과 최종 UI를 분리하기 위한 컨트롤러 기능을 제공
- 6)**@Controller** 어노테이션을 사용하여 작성된 **Controller** 클래스가 이 계층에 속함

3. Service Layer

- 1)어플리케이션 비즈니스 로직 처리와 비즈니스와 관련된 도메인 모델의 적합성 검증
- 2)**Transaction** 처리
- 3)**Presentation Layer**와 **Data Access Layer** 사이를 연결하는 역할로서 두 계층이 직접적으로 통신하지 않게 하여 어플리케이션의 유연성을 증가
- 4)다른 계층들과 통신하기 위한 인터페이스 제공
- 5)**Service** 인터페이스와 **@Service** 어노테이션을 사용하여 작성된 **Service** 구현 클래스가 이 계층에 속함.

4. Data Access Layer

- 1)영구 저장소(관계형 데이터베이스)의 데이터를 조작하는 데이터 액세스 로직을 객체화
- 2)영구 저장소의 데이터를 조회, 등록, 수정, 삭제함
- 3)**ORM(Object Relational Mapping)** 프레임워크(**MyBatis, Hibernate**)를 주로 사용하는 계층
- 4)**DAO** 인터페이스와 **@Repository** 어노테이션을 사용하여 작성된 **DAO** 구현 클래스가 이 계층에 속함.

5. Domain Model Class

- 1)관계형 데이터 베이스의 엔티티와 비슷한 개념을 가지는 것으로 실제 **VO(Value Object)** 혹은 **DTO(Data Transfer Object)** 객체에 해당
- 2)도메인 모델 클래스는 3개의 계층 전체에 걸쳐 사용
- 3)**private**으로 선언된 멤버변수가 있고, 그 변수에 대한 **getter**와 **setter** 메소드를 가진 클래스를 말함.

6. 데이터 액세스 공통 개념

1)DAO(Data Access Object) Pattern

- 데이터 액세스 계층은 **DAO** 패턴을 적용하여 비즈니스 로직과 데이터 액세스 로직을 분리하는 것이 원칙이다.
- 데이터베이스 접속과 **SQL** 발행 같은 데이터 액세스 처리를 **DAO**라고 불리는 오브젝트로 분리하는 패턴이다.
- 비즈니스 로직이 없거나 단순하면 **DAO**와 서비스 계층을 통합할 수도 있지만, 의미 있는 비즈니스 로직을 가진 엔터프라이즈 어플리케이션이라면 데이터 액세스 계층을 **DAO** 패턴으로 분리해야 한다.
- DAO** 패턴은 서비스계층에 영향을 주지 않고 데이터 액세스 기술을 변경할 수 있는 장점을 가지고 있다.
- 비즈니스 로직 -> **Dao** 인터페이스 -> **XxxDao(CRUD구현)** -> **Database**
- Dao**클래스에 데이터 액세스 처리를 기술하겠지만, 그 처리를 구현하는 **Java**기술은 여러 가지다.
- Dao Interface** -> **XxxDao(CRUD)** -> **JDBC/Hibernate/MyBatis(iBATIS)/JPA/JDO**등등등 -> **Database**
- Spring**에서는 새로운 데이터 액세스 기술을 제공하는 것이 아니라 기존의 5가지 방법 기술들을 좀 더 쉽게 만드는 기능을 제공한다.
- 즉 **JDBC**는 **Spring JDBC**로, **Hibernate**는 **Hibernate** 연계로, **JPA**는 **JPA** 연계로, **MyBatis**는 **MyBatis** 연계로, **JDO**는 **JDO** 연계이다.
- Spring**의 기능을 이용해서 얻을 수 있는 장점을 아래 3가지 이다.
- 데이터 액세스 처리를 간결하게 기술할 수 있다.

- 스프링이 제공하는 범용적이고 체계적인 데이터 액세스 예외를 이용할 수 있다.
- 스프링의 트랙잭션 기능을 이용할 수 있다.
- 여기서는 5가지 기술 중 Spring JDBC를 다룬다.

2)Connection Pooling을 지원하는 DataSource

- Connection Pooling은 미리 정해진 갯수만큼의 DB Connection을 Pool에 준비해두고, 어플리케이션이 요청할 때마다 Pool에서 꺼내서 하나씩 할당해주고 다시 돌려받아서 Pool에 넣는 식의 기법이다.
- 다중 사용자를 갖는 엔터프라이즈 시스템에서라면 반드시 DB Connection Pooling 기능을 지원하는 DataSource를 사용해야 한다.
- Spring에서는 DataSource를 공유 가능한 Spring Bean으로 등록해 주어 사용할 수 있도록 해준다.

7. DataSource 구현 클래스의 종류

1)테스트 환경을 위한 DataSource

-SimpleDriverDataSource

- Spring이 제공하는 가장 단순한 DataSource 구현 클래스
- getConnection()을 호출할 때마다 매번 DB Connection을 새로 만들고 따로 Pool을 관리하지 않으므로 단순한 테스트용으로만 사용해야 한다.

-SingleConnectionDriverDataSource

- 순차적으로 진행되는 통합 테스트에서는 사용 가능하다.
- 매번 DB Connection을 생성하지 않기 때문에 SimpleDriverDataSource보다 빠르게 동작한다.

2)OpenSource DataSource

-Apache Commons DBCP

- 가장 유명한 오픈소스 DB Connection Pool Library이다.
- Apache의 Commons Project(<http://commons.apache.org/dbcp>)

-c3p0 JDBC/DataSource Resource Pool

- c3p0는 JDBC 3.0 스펙을 준수하는 Connection과 Statement Pool을 제공하는 라이브러리이다.
- <http://www.mchange.com/projects/c3p0/>
- 두 가지 모두 수정자(setter) 메소드를 제공하므로 Spring Bean으로 등록해서 사용하기 편리.

8. Spring JDBC

1)JDBC란?

- 모든 Java의 Data Access 기술의 근간
- Entity Class와 Annotation을 이용하는 최신 ORM 기술도 내부적으로는 DB와의 연동을 위해 JDBC를 이용
- 안정적이고 유연한 기술이지만, Low level 기술로 인식되고 있다.
- 간단한 SQL을 실행하는데도 중복된 코드가 반복적으로 사용되며, DB에 따라 일관성 없는 정보를 가진 Checked Exception으로 처리한다.
- 장점
 - 대부분의 개발자가 잘 알고 있는 친숙한 데이터 액세스 기술로 별도의 학습 없이 개발이 가능
- 단점
 - Connection과 같은 공유 리소스를 제대로 릴리즈 해주지 않으면 시스템의 자원이 바닥나는 버그 발생.

2)Spring JDBC?

- JDBC의 장점과 단순성을 그대로 유지하면서도 기존 JDBC의 단점을 극복
- 간결한 형태의 API 사용법을 제공
- JDBC API에서 지원되지 않는 편리한 기능 제공
- 반복적으로 해야 하는 많은 작업들을 대신 해줌.
- Spring JDBC를 사용할 때는 실행할 SQL과 바인딩 할 파라미터를 넘겨주거나, 쿼리의 실행 결과를 어떤 객체에서 넘겨 받을지를 지정하는 것만 하면 된다.
- Spring JDBC를 사용하려면 먼저, DB Connection을 가져오는 DataSource를 Bean으로 등록해야 한다.

3)개발자가 JDBC방식으로 연결시의 문제점들

- 직접 개발자가 JDBC를 사용하면 소스 코드가 너무 길어지고 또한 커백션이나 PreparedStatement를 얻고 나면 반드시 연결 해제를 처리해야 하지만 깜빡 잊어버리는 개발자도 있을 수 있다.
- 그래서 연결이 해제되지 않으면 데이터베이스의 리소스 고갈이나 메모리 누수의 원인이 되어 최악의 경우에는 시스템이 정지할 가능성도 있다.
- 데이터 액세스 오류시 오류 원인을 특정하고 싶을 때는 SQLException의 오류 코드를 가져와 값을 조사할 필요가 있다.
- 더욱이 오류 코드는 데이터베이스 제품마다 값이 다르므로 데이터베이스 제품이 바뀌면 다시 수정해야만 한다.
- 또한 SQLException은 컴파일 시 예외 처리 유무를 검사하므로 소스 코드 상에서 반드시 catch 문을 기술해야만 한다.

4)Spring JDBC가 해주는 작업들

- Connection 열기와 닫기
 - Connection과 관련된 모든 작업을 Spring JDBC가 필요한 시점에서 알아서 진행한다.
 - 진행 중에 예외가 발생했을 때도 열린 모든 Connection 객체를 닫아준다.
- Statement 준비와 닫기
 - SQL 정보가 담긴 Statement 또는 PreparedStatement를 생성하고 필요한 준비 작업을 한다.
 - Statement도 Connection과 마찬가지로 사용이 끝나면 Spring JDBC가 알아서 닫아준다.
- Statement 실행
 - SQL이 담긴 Statement를 실행
 - Statement의 실행결과를 다양한 형태로 가져올 수 있다.
- ResultSet Loop 처리
 - ResultSet에 담긴 쿼리 실행 결과가 한 건 이상이면 ResultSet 루프를 만들어서 반복한다.
- Exception 처리와 반환
 - JDBC 작업 중 발생하는 모든 예외는 Spring JDBC 예외 변환기가 처리한다.
 - Checked Exception인 SQLException을 Runtime Exception인 DataAccessException 타입으로 변환
- Transaction 처리
 - Transaction과 관련된 모든 작업에 대해서는 신경쓰지 않아도 된다.

5)Spring JDBC의 JdbcTemplate Class

- Spring JDBC가 제공하는 클래스 중 하나
- JDBC의 모든 기능을 최대한 활용할 수 있는 유연성을 제공하는 클래스
- 실행, 조회, 배치의 3가지 작업 제공
 - 실행 : Insert나 Update같이 DB의 데이터에 변경이 일어나는 쿼리를 수행하는 작업
 - 조회 : Select를 이용해 데이터를 조회하는 작업
 - 배치 : 여러 개의 쿼리를 한번에 수행해야 하는 작업

6)JdbcTemplate class 생성

- JdbcTemplate은 DataSource를 파라미터로 받아서 아래와 같이 생성한다.


```
JdbcTemplate template = new JdbcTemplate(dataSource);
```
- DataSource는 보통 Bean으로 등록해서 사용하므로 JdbcTemplate이 필요한 DAO class에서 DataSource Bean을 DI 받아서 JdbcTemplate을 생성할 때 인자로 넘겨주면 된다.
- JdbcTemplate은 멀티스레드 환경에서도 안전하게 공유해서 쓸 수 있기 때문에 DAO class의 인스턴스 변수에 저장해 두고 사용할 수 있다.
- 생성 예

```
public class UserDAOJdbc{
    JdbcTemplate jdbcTemplate;

    @Autowired
    public void setDataSource(DataSource dataSource){
        jdbcTemplate = new JdbcTemplate(dataSource);
    }
}
```

7)JdbcTemplate의 Update() 메소드

- INSERT, UPDATE, DELETE와 같은 SQL을 실행할 때 사용.


```
int update(String sql, [SQL 파라미터])
```
- 이 메소드를 호출할 때는 SQL과 함께 바인딩 할 파라미터는 Object 타입 가변인자(Object ... args)를 사용할 수 있다.
- 이 메소드의 리턴값은 SQL 실행으로 영향받은 레코드의 갯수이다.
- 사용 예

```
public int update(User user){
    StringBuffer updateQuery = new StringBuffer();
    updateQuery.append("UPDATE USERS SET ");
    updateQuery.append("password=?, name=? ");
    updateQuery.append("WHERE id=? ");

    int result = this.jdbcTemplate.update(updateQuery.toString(),
        user.getName(), user.getPassword(), user.getId());

    return result;
}
```

8)JdbcTemplate의 queryForObject() 메소드

- SELECT SQL을 실행하여 하나의 Row를 가져올 때 사용.


```
<T> T queryForObject(String sql, [SQL 파라미터], RowMapper<T> rm)
```
- SQL 실행 결과는 여러 개의 칼럼을 가진 하나의 Row

-T는 VO 객체의 타입에 해당

-SQL 실행 결과로 돌아온 여러 개의 Column을 가진 한 개의 Row를 RowMapper 콜백을 이용해 VO 객체로 매핑한다.

-사용 예

```
public User findUser(String id){
    return this.jdbcTemplate.queryForObject("SELECT * FROM users WHERE id=?",
        new Object [] {id},
        new RowMapper<User>(){
            public User mapRow(ResultSet rs, int rowNum) throws SQLException{
                User user = new User();
                user.setId(rs.getString("id"));
                user.setName(rs.getString("name"));
                user.setPassword(rs.getString("password"));
                return user;
            }
        }
    );
}
```

9)JdbcTemplate 클래스의 query() 메소드

-SELECT SQL을 실행하여 여러 개의 Row를 가져올 때 사용.

```
<T> List<T> query(String sql, [SQL 파라미터], RowMapper<T> rm)
```

-SQL 실행 결과로 돌아온 여러 개의 Column을 가진 여러 개의 Row를 RowMapper 콜백을 이용해 VO 객체로 매핑해준다.

-결과 값은 매핑 한 VO 객체를 포함하고 있는 List 형태로 받는다.

-List의 각 요소가 하나의 Row에 해당한다.

9. Spring JDBC 환경설정

1)Oracle Jdbc Driver 라이브러리 검색 및 설치

※Oracle의 경우 어떤 드라이버를 pom.xml에 넣어도 에러가 난다.

원래는 Oracle 12C인 경우 Maven Repository에서 'oracle ojdbc7'으로, Oracle 11g인 경우는 'oracle ojdbc6'로 검색해야 한다.

1)'oracle ojdbc7'으로 검색시 12.1.0.2

```
<!-- https://mvnrepository.com/artifact/com.github.noraui/ojdbc7 -->
<dependency>
    <groupId>com.github.noraui</groupId>
    <artifactId>ojdbc7</artifactId>
    <version>12.1.0.2</version>
</dependency>
```

2)'oracle ojdbc6'으로 검색시 11.1.0.7.0

```
<!-- https://mvnrepository.com/artifact/com.oracle/ojdbc6 -->
<dependency>
    <groupId>com.oracle</groupId>
    <artifactId>ojdbc6</artifactId>
    <version>11.1.0.7.0</version>
    <scope>test</scope>
</dependency>
```

하지만 어떤 버전도 Maven에서 에러가 난다. <http://suyou.tistory.com/68> 참조.

메이븐에서 Oracle 드라이버를 찾지 못하는 것은 아마도 저작권 문제로 보인다.

그래서 Oracle 사이트에서 직접 드라이버를 다운로드 받아서 Maven을 이용해서 Maven Local Repository에 인스톨을 하고

인스톨된 버전으로 pom.xml에 디펜던시 설정을 해야 한다.

① 오라클 홈페이지에서 jdbc드라이버를 다운로드 받는다. -->ojdbc6.jar

② 메이븐 인스톨러를 이용해서 메이븐 레포지토리에 설치한다.

```
mvn install:install-file -Dfile="파일이름(위치까지)" -DgroupId=그룹아이디 -DartifactId=파일이름
-Dversion=버전 -Dpackaging=jar
```

위에명령을 cmd에서 실행한다.

```

241 자기버전에 맞게 해당항목을 변경한다음 실행한다.
242 mvn install:install-file
-Dfile="C:\oraclexe\app\oracle\product\11.2.0\server\jdbc\lib\ojdbc6.jar"
-DgroupId=com.oracle -DartifactId=ojdbc6 -Dversion=11.1 -Dpackaging=jar

243
244 인스톨 명령을 실행하면 메이븐 depository에 해당 드라이버가 설치된다.
245 위에서는 C:\Users\webnbiz01\.m2\repository\com\oracle\ojdbc6\11.1 에 설치된 것이다.
246 해당 디렉토리로 이동하면 jar 파일과 pom 파일이 있다.
247 pom파일의 groupId, artifactId, version을 pom.xml에 디펜던시로 설정하면 된다.
248
249 ③ pom.xml에 디펜던시를 설정한다.
250 <dependency>
251 <groupId>com.oracle</groupId>
252 <artifactId>ojdbc6</artifactId>
253 <version>11.1</version>
254 </dependency>
255
256 pom.xml에 추가한다.
257 이후 pom.xml clean후 install 한다.
258 *****
259
260 2)Spring JDBC 설치
261 -Maven Repository에서 'Spring jdbc'라고 검색
262 -JdbcTemplate를 사용하기 위해 pom.xml에 다음 dependency를 추가해야 함.
263
264 <!-- https://mvnrepository.com/artifact/org.springframework/spring-jdbc -->
265 <dependency>
266 <groupId>org.springframework</groupId>
267 <artifactId>spring-jdbc</artifactId>
268 <version>4.3.13.RELEASE</version>
269 </dependency>
270
271
272 10. Lab
273 1)SpringJdbcDemo project 생성
274 -New > Java Project >
275 -Project name : SpringJdbcDemo > Finish
276
277 2)com.javasoft Package 생성
278 -/src > right-click > New > Package
279 -Name : com.javasoft > Finish
280
281 3)config 폴더 생성
282 -SpringJdbcDemo project > right-click > Build Path > Coinfigure Build Path
283 -Source Tab > Add Folder > Select SpringJdbcDemo project > Click [Create New Folder] button
284 -Folder name : config > Finish > OK
285
286 4)config/dbinfo.properties 파일 생성
287 -config > right-click > New > File
288 -File name : dbinfo.properties > Finish
289
290 db.driverClass=oracle.jdbc.driver.OracleDriver
291 db.url=jdbc:oracle:thin:@localhost:1521:XE
292 db.username=scott
293 db.password=tiger
294
295 5)/src/com.javasoft.UserClient.java 생성
296 -/src > com.javasoft > right-click > New > Class
297 -Name : UserClient
298
299 public class UserClient{
300     public static void main(String [] args){
301
302     }
303 }
304
305 6)Maven Project로 전환

```

-SpringJdbcDemo Project > right-click > Configure > Convert to Maven Project
-Finish

7)Spring Project로 전환

-SpringJdbcDemo Project > right-click > Spring Tools > Add Spring Project Nature

8)Oracle Jdbc Driver 라이브러리 검색 및 설치

9)Spring Context 설치

-Maven Repository 에서 'Spring Context'로 검색하여 디펜던시 추가하고 설치

```
<!-- https://mvnrepository.com/artifact/org.springframework/spring-context -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>4.3.13.RELEASE</version>
</dependency>
```

-pom.xml에 붙여 넣고 Maven Install 하기

10)Spring JDBC 설치

-JdbcTemplate를 사용하기 위해 pom.xml에 다음 dependency를 추가해야 함.

```
<!-- https://mvnrepository.com/artifact/org.springframework/spring-jdbc -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jdbc</artifactId>
    <version>4.3.13.RELEASE</version>
</dependency>
```

-pom.xml에 붙여 넣고 Maven Install 하기

11)Bean Configuration XML 작성

~/src/config > right-click > New > Other > Spring > Spring Bean Configuration File

-File name : beans.xml > Next

-Check [beans - <http://www.springframework.org/schema/beans>]

-Check [<http://www.springframework.org/schema/beans/spring-beans-4.3.xsd>]

-Finish

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd">

</beans>
```

-Namespace tab에서 context - <http://www.springframework.org/schema/context> check

```
<context:property-placeholder location="classpath:dbinfo.properties" />
<bean id="dataSource"
    class="org.springframework.jdbc.datasource.SimpleDriverDataSource">
    <property name="driverClass" value="${db.driverClass}" />
    <property name="url" value="${db.url}" />
    <property name="username" value="${db.username}" />
    <property name="password" value="${db.password}" />
</bean>
```

12)/src/com.javasoft.UserClient.java 코드 추가

```
package com.javasoft;

import java.sql.SQLException;

import javax.sql.DataSource;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.GenericXmlApplicationContext;
```

```

372
373 public class UserClient {
374     public static void main(String[] args) {
375         ApplicationContext ctx = new GenericXmlApplicationContext("classpath:beans.xml");
376
377         DataSource ds = (DataSource) ctx.getBean("dataSource");
378         try{
379             System.out.println(ds.getConnection());
380         }catch(SQLException ex){
381             System.out.println(ex);
382         }
383     }
384 }

```

13)Test

```

oracle.jdbc.driver.T4CConnection@51c8530f

```

11. Membership Project

1)Table 설계

```

CREATE TABLE users
(
    userid    VARCHAR2(12) NOT NULL PRIMARY KEY,
    name      VARCHAR2(20) NOT NULL,
    gender    VARCHAR2(10),
    city      VARCHAR2(30)
);

INSERT INTO users VALUES('jimin', '한지민', '여', '서울');
COMMIT;

```

2)Class Diagram

Membership Class Diagram.uml 파일 참조

3)각 Class의 역할

-Presentation Layer

--UserController<Class>

- UI계층과 서비스 계층을 연결하는 역할을 하는 클래스
- JSP에서 UserController를 통해서 서비스 계층의 UserService를 사용하게 된다.
- Service 계층의 UserService 인터페이스를 구현하나 객체를 IoC 컨테이너가 주입해 준다.

-Service Layer

--UserService<Interface>

- 서비스 계층에 속한 상위 인터페이스

--UserServiceImpl<Class>

- UserSerive 인터페이스를 구현한 클래스
- 복잡한 업무 로직이 있을 경우에는 이 클래스에서 업무 로직을 구현하면 된다.
- 데이터 액세스 계층의 userDao 인터페이스를 구현한 객체를 IoC 컨테이너가 주입해준다.

-Data Access Layer

--UserDao<Interface>

- 데이터 액세스 계층에 속한 상위 인터페이스

--UserDaoImplJDBC<Class> - Spring JDBC 구현

- UserDao 인터페이스를 구현한 클래스로 이 클래스에서는 데이터 액세스 로직을 구현하면 된다.
- Spring JDBC를 사용하는 경우에는 DataSource를 IoC 컨테이너가 주입해준다.
- MyBatis를 사용하는 경우에는 SqlSession을 IoC 컨테이너가 주입해준다.

4)New > Spring Legacy Project > Simple Spring Maven

Project name : Membership

5)/src/main/java > right-click > New > Package

Package name : com.javasoft.vo

6)UserVO.java 생성

```

package com.javasoft.vo;

```

```

439
440 public class UserVO {
441
442     private String userId;
443     private String name;
444     private String gender;
445     private String city;
446
447     public UserVO() {}
448
449     public UserVO(String userId, String name, String gender, String city) {
450         this.userId = userId;
451         this.name = name;
452         this.gender = gender;
453         this.city = city;
454     }
455
456     public String getUserId() {
457         return userId;
458     }
459
460     public void setUserId(String userId) {
461         this.userId = userId;
462     }
463
464     public String getName() {
465         return name;
466     }
467
468     public void setName(String name) {
469         this.name = name;
470     }
471
472     public String getGender() {
473         return gender;
474     }
475
476     public void setGender(String gender) {
477         this.gender = gender;
478     }
479
480     public String getCity() {
481         return city;
482     }
483
484     public void setCity(String city) {
485         this.city = city;
486     }
487
488     @Override
489     public String toString() {
490         return "User [userId=" + userId + ", name=" + name + ", gender="
491             + gender + ", city=" + city + "]\n";
492     }
493 }

```

7)/src/main/java > right-click > New > Package
 Package name : com.javasoft.service

8)UserService.java

```

500 package com.javasoft.service;
501
502 import java.util.List;
503 import com.javasoft.vo.UserVO;
504
505 public interface UserService {

```



```

506
507     void insertUser(UserVO user);
508
509     List<UserVO> getUserList();
510
511     void deleteUser(String id);
512
513     UserVO getUser(String id);
514
515     void updateUser(UserVO user);
516 }
517

```

9) UserServiceImpl.java

```

518     package com.javasoft.service;
519
520
521     import java.util.List;
522     import com.javasoft.vo.UserVO;
523
524     public class UserServiceImpl implements UserService {
525
526         @Override
527         public void insertUser(UserVO user) {
528             // TODO Auto-generated method stub
529
530         }
531
532         @Override
533         public List<UserVO> getUserList() {
534             // TODO Auto-generated method stub
535             return null;
536         }
537
538         @Override
539         public void deleteUser(String id) {
540             // TODO Auto-generated method stub
541
542         }
543
544         @Override
545         public UserVO getUser(String id) {
546             // TODO Auto-generated method stub
547             return null;
548         }
549
550         @Override
551         public void updateUser(UserVO user) {
552             // TODO Auto-generated method stub
553
554         }
555     }
556

```

10)/src/main/java > right-click > New > Package
Package name : com.javasoft.dao

11) UserDao.java

```

561     package com.javasoft.dao;
562
563
564     import java.util.List;
565     import com.javasoft.vo.UserVO;
566
567     public interface UserDao {
568         void insert(UserVO user);
569
570         List<UserVO> readAll();
571
572         void update(UserVO user);

```

```

573         void delete(String id);
574     }
575     UserVO read(String id);
576 }
577
578 12) UserDaoImplJDBC.java
579 package com.javasoft.dao;
580
581 import java.util.List;
582 import com.javasoft.vo.UserVO;
583
584 public class UserDaoImplJDBC implements UserDao {
585
586     @Override
587     public void insert(UserVO user) {
588         // TODO Auto-generated method stub
589     }
590
591     @Override
592     public List<UserVO> readAll() {
593         // TODO Auto-generated method stub
594         return null;
595     }
596
597     @Override
598     public void update(UserVO user) {
599         // TODO Auto-generated method stub
600     }
601
602     @Override
603     public void delete(String id) {
604         // TODO Auto-generated method stub
605     }
606
607     @Override
608     public UserVO read(String id) {
609         // TODO Auto-generated method stub
610         return null;
611     }
612 }
613
614 13) Oracle Jdbc Driver 설치
615 <dependency>
616     <groupId>com.oracle</groupId>
617     <artifactId>ojdbc6</artifactId>
618     <version>11.1</version>
619 </dependency>
620
621 <참고>
622 -MySQL일 경우에는 'spring mysql'로 검색하여 MySQL Connector/J를 설치한다.
623 <dependency>
624     <groupId>mysql</groupId>
625     <artifactId>mysql-connector-java</artifactId>
626     <version>6.0.6</version>
627 </dependency>
628
629 14) Spring JDBC 설치
630 -JdbcTemplate를 사용하기 위해 pom.xml에 다음 dependency를 추가해야 함.
631 <dependency>
632     <groupId>org.springframework</groupId>
633     <artifactId>spring-jdbc</artifactId>
634     <version>4.3.13.RELEASE</version>

```

</dependency>

-pom.xml에 붙여 넣고 Maven Install 하기

15)dbinfo.properties 파일 생성

-/src/main/resources/dbinfo.properties 파일 생성
-/src/main/resources > right-click > New > File
-File name : dbinfo.properties > Finish

```
db.driverClass=oracle.jdbc.driver.OracleDriver
db.url=jdbc:oracle:thin:@localhost:1521:XE
db.username=scott
db.password=tiger
```

<참고>

-MySQL일 경우에는 다음과 같이 설정한다.

```
db.driverClass=com.mysql.jdbc.Driver
db.url=jdbc:mysql://192.168.136.5:3306/world
db.username=root
db.password=javamysql
```

16)Bean Configuration XML 작성

-/src/main/resources > right-click > New > Spring Bean Configuration File
-File name : beans.xml > Finish
-Namespace Tab
-Check context - <http://www.springframework.org/schema/context>

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:context="http://www.springframework.org/schema/context"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.2.xsd">

  <context:property-placeholder location="classpath:dbinfo.properties" />
  <bean id="dataSource"
    class="org.springframework.jdbc.datasource.SimpleDriverDataSource">
    <property name="driverClass" value="${db.driverClass}" />
    <property name="url" value="${db.url}" />
    <property name="username" value="${db.username}" />
    <property name="password" value="${db.password}" />
  </bean>
</beans>
```

17)사용자 관리 프로젝트의 Bean 등록 및 의존 관계 설정

-<context:component-scan> 태그 사용
-@Service, @Repository 어노테이션을 선언한 클래스들과 @Autowired 어노테이션을 선언하여
의존관계를 설정한 클래스들이 위치한 패키지를 Scan하기 위한 설정을 XML에 해주어야 한다.
-beans.xml에 다음 코드 추가한다.

```
<context:component-scan base-package="com.javasoft" />
```

18)Spring TestContext Framework 사용하기

-/src/test/java > right-click > New > JUnit Test Case
-Name : MembershipTest > Finish

```
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

import com.javasoft.service.UserService;

@RunWith(SpringJUnit4ClassRunner.class)
```

```

703 @ContextConfiguration(locations="classpath:beans.xml")
704 public class MembershipTest {
705
706     @Autowired
707     UserService service;
708
709     @Test
710     public void test() {
711
712     }
713 }
714

```

19)Oracle JDBC Driver(ojdbc) Project BuildPath에 추가
-ojdbc6.jar

<참고>

-ojdbc8.jar([http://www.oracle.com/technetwork/database/features/jdbc/jdbc-ucp-122-3110062.h
tml](http://www.oracle.com/technetwork/database/features/jdbc/jdbc-ucp-122-3110062.html))

<참고>

-MySQL일 경우에는
mysql-connector-java-5.1.42-bin.jar(<https://dev.mysql.com/downloads/connector/j/>) 추가

12. JDBC를 이용한 Membership Project

1)사용자 조회 테스트

-com.javasoft.dao.UserDaoImplJDBC.java 수정

```

727 @Repository("userDao")
728 public class UserDaoImplJDBC implements UserDao {
729     private DataSource dataSource;
730
731     @Autowired
732     public void setDataSource(DataSource dataSource) {
733         this.dataSource = dataSource;
734     }
735
736     ...
737     @Override
738     public UserVO read(String id) {
739         Connection conn = null;
740         PreparedStatement pstmt = null;
741         ResultSet rs = null;
742         UserVO userVO = null;
743         try {
744             conn = this.dataSource.getConnection();
745             pstmt = conn.prepareStatement("SELECT * FROM users WHERE userid = ?");
746             pstmt.setString(1, id);
747             rs = pstmt.executeQuery();
748             rs.next();
749             userVO = new UserVO(rs.getString("userid"), rs.getString("name"),
750                                 rs.getString("gender"), rs.getString("city"));
751         }catch(SQLException ex) {
752             System.out.println(ex);
753         }finally {
754             try {
755                 if(conn != null) conn.close();
756                 if(pstmt != null) pstmt.close();
757                 if(rs != null) rs.close();
758             }catch(SQLException ex) {
759                 System.out.println(ex);
760             }
761         }
762         return userVO;
763     }
764 }
765

```

-com.javasoft.service.UserServiceImpl.java 수정

```

766 @Service("userService")
767 public class UserServiceImpl implements UserService {
768
769     @Autowired
770     UserDao userDao;
771
772     ...
773     @Override
774     public UserVO getUser(String id) {
775         return userDao.read(id);
776     }
777

```

778 -/src/test/java/MembershipTest.java

```

779
780     @Test
781     public void test() {
782         //사용자 조회 테스트
783         UserVO user = service.getUser("jimin");
784         System.out.println(user);
785         assertEquals("한지민", user.getName());
786     }
787

```

788 *****

789 java.lang.NoClassDefFoundError: Could not initialize class
org.springframework.jdbc.core.StatementCreatorUtils

790 <에러 해결 방법>

```

791     <dependency>
792         <groupId>org.springframework</groupId>
793         <artifactId>spring-context</artifactId>
794         <version>4.3.13.RELEASE</version>
795     </dependency>
796

```

797 -버전 변경 후 Maven Clean -> Maven Install

798 2) 사용자 등록 및 목록 조회 테스트

800 -com.javasoft.dao.UserDaoImplJDBC.java 코드 수정

```

801     @Override
802     public void insert(UserVO user) {
803         Connection conn = null;
804         PreparedStatement pstmt = null;
805         try {
806             conn = this.dataSource.getConnection();
807             String sql = "INSERT INTO users (userid, name, gender,city) VALUES (?, ?, ?, ?)";
808             pstmt = conn.prepareStatement(sql);
809             pstmt.setString(1, user.getUserId());
810             pstmt.setString(2, user.getName());
811             pstmt.setString(3, user.getGender());
812             pstmt.setString(4, user.getCity());
813             pstmt.executeUpdate();
814             System.out.println("등록된 Record UserId=" + user.getUserId() + " Name=" +
                user.getName());
815         } catch (SQLException ex) {
816             System.out.println(ex);
817         } finally {
818             try {
819                 if(conn != null) conn.close();
820                 if(pstmt != null) pstmt.close();
821             } catch (SQLException ex) {
822                 System.out.println(ex);
823             }
824         }
825     }
826
827     @Override
828     public List<UserVO> readAll() {
829         Connection conn = null;
830         Statement stmt = null;

```

```

831     ResultSet rs = null;
832     List<UserVO> userList = null;
833     try {
834         conn = this.dataSource.getConnection();
835         stmt = conn.createStatement();
836         rs = stmt.executeQuery("SELECT * FROM users");
837         userList = new ArrayList<UserVO>();
838         while(rs.next()) {
839             UserVO userVO = new UserVO(rs.getString("userid"), rs.getString("name"),
840                 rs.getString("gender"), rs.getString("city"));
841             userList.add(userVO);
842         }
843     }catch(SQLException ex) {
844         System.out.println(ex);
845     }finally {
846         try {
847             if(conn != null) conn.close();
848             if(stmt != null) stmt.close();
849             if(rs != null) rs.close();
850         }catch(SQLException ex) {
851             System.out.println(ex);
852         }
853     }
854     return userList;
855 }

```

856 -com.javasoft.service.UserServiceImpl.java 코드 수정

```

858     @Override
859     public void insertUser(UserVO user) {
860         userDao.insert(user);
861     }
862
863     @Override
864     public List<UserVO> getUserList() {
865         return userDao.readAll();
866     }
867

```

868 -/src/test/java/MembershipTest.java

```

869
870     ...
871     @Test
872     public void test1() {
873         //사용자 등록 및 목록조회 테스트
874         this.service.insertUser(new UserVO("dooly", "둘리", "남", "경기"));
875         for(UserVO user : this.service.getUserList()){
876             System.out.println(user);
877         }
878     }
879

```

880 3)사용자 정보 수정 테스트

881 -com.javasoft.dao.UserDaoImplJDBC.java 코드 수정

```

882
883     @Override
884     public void update(UserVO user) {
885         Connection conn = null;
886         PreparedStatement pstmt = null;
887         try {
888             conn = this.dataSource.getConnection();
889             String sql = "UPDATE users SET name = ?, gender = ?, city = ? WHERE userid = ?";
890             pstmt = conn.prepareStatement(sql);
891             pstmt.setString(1, user.getName());
892             pstmt.setString(2, user.getGender());
893             pstmt.setString(3, user.getCity());
894             pstmt.setString(4, user.getUserId());
895             pstmt.executeUpdate();
896             System.out.println("갱신된 Record with ID = " + user.getUserId() );

```

```

897         }catch(SQLException ex) {
898             System.out.println(ex);
899         }finally {
900             try {
901                 if(conn != null) conn.close();
902                 if(pstmt != null) pstmt.close();
903             }catch(SQLException ex) {
904                 System.out.println(ex);
905             }
906         }
907     }

```

908
909 -com.javasoft.service.UserServiceImpl.java 코드 수정

```

910
911     @Override
912     public void updateUser(UserVO user) {
913         userDao.update(user);
914     }

```

915
916 -/src/test/java/MembershipTest.java

```

917
918     @Ignore @Test
919     public void test1() {
920         //사용자 등록 및 목록조회 테스트
921         this.service.insertUser(new UserVO("dooly", "둘리", "남", "경기"));
922         for(UserVO user : this.service.getUserList()){
923             System.out.println(user);
924         }
925     }
926
927     @Test
928     public void test2() {
929         //사용자 정보 수정 테스트
930         service.updateUser(new UserVO("dooly", "김둘리", "여", "부산"));
931         UserVO user = service.getUser("dooly");
932         System.out.println(user);
933     }

```

934
935 4)사용자 정보 삭제 테스트

936 -com.javasoft.dao.UserDaoImplJDBC.java 코드 수정

```

937
938     @Override
939     public void delete(String id) {
940         Connection conn = null;
941         PreparedStatement pstmt = null;
942         try {
943             conn = this.dataSource.getConnection();
944             pstmt = conn.prepareStatement("DELETE FROM users WHERE userid = ?");
945             pstmt.setString(1, id);
946             pstmt.executeUpdate();
947             System.out.println("삭제된 Record with ID = " + id );
948         }catch(SQLException ex) {
949             System.out.println(ex);
950         }finally {
951             try {
952                 if(conn != null) conn.close();
953                 if(pstmt != null) pstmt.close();
954             }catch(SQLException ex) {
955                 System.out.println(ex);
956             }
957         }
958     }

```

959
960 -com.javasoft.service.UserServiceImpl.java 코드 수정

```

961
962     @Override
963     public void deleteUser(String id) {

```

```

964         userDao.delete(id);
965     }
966
967 -/src/test/java/MembershipTest.java
968     @Test
969     public void test() {
970         UserVO user = this.service.getUser("jimin");
971         System.out.println(user);
972         assertEquals("한지민", user.getName());
973     }
974     @Ignore @Test
975     public void test1() {
976         this.service.insertUser(new UserVO("dooly", "둘리", "남", "경기"));
977         for(UserVO user : this.service.getUserList()){
978             System.out.println(user);
979         }
980     }
981
982     @Ignore @Test
983     public void test2() {
984         service.updateUser(new UserVO("dooly", "김둘리", "여", "부산"));
985         UserVO user = service.getUser("dooly");
986         System.out.println(user);
987     }
988
989     @Test
990     public void test3() {
991         //사용자 정보 삭제 테스트
992         service.deleteUser("dooly");
993         for(UserVO user : service.getUserList()){
994             System.out.println(user);
995         }
996     }
997

```

13. iBATIS를 이용한 Membership Project

1) 사용자 조회 테스트

-Project Build path에 ibatis-2.3.4.726.jar 등록

-src/main/java/SqlMapConfig.xml 생성

```

1003 <?xml version="1.0" encoding="UTF-8"?>
1004 <!DOCTYPE sqlMapConfig
1005     PUBLIC "-//ibatis.apache.org//DTD SQL Map Config 2.0//EN"
1006     "http://ibatis.apache.org/dtd/sql-map-config-2.dtd">
1007 <sqlMapConfig>
1008     <properties resource="dbinfo.properties" />
1009     <settings useStatementNamespaces="true"/>
1010     <transactionManager type="JDBC">
1011         <dataSource type="SIMPLE">
1012             <property name="JDBC.Driver" value="${db.driverClass}"/>
1013             <property name="JDBC.ConnectionURL" value="${db.url}"/>
1014             <property name="JDBC.Username" value="${db.username}"/>
1015             <property name="JDBC.Password" value="${db.password}"/>
1016         </dataSource>
1017     </transactionManager>
1018     <sqlMap resource="com/javasoft/dao/Users.xml"/>
1019 </sqlMapConfig>

```

-com.javasoft.dao/User.xml 생성

```

1022 <?xml version="1.0" encoding="UTF-8"?>
1023 <!DOCTYPE sqlMap
1024     PUBLIC "-//ibatis.apache.org//DTD SQL Map 2.0//EN"
1025     "http://ibatis.apache.org/dtd/sql-map-2.dtd">
1026 <sqlMap namespace="Users">
1027     <typeAlias alias="userVO" type="com.javasoft.vo.UserVO"/>
1028     <resultMap id="result" class="userVO">
1029         <result property="userId" column="userid"/>
1030         <result property="name" column="name"/>

```



```

1031         <result property="gender" column="gender"/>
1032         <result property="city" column="city"/>
1033     </resultMap>
1034     <select id="useResultMap" resultMap="result">
1035         SELECT * FROM users WHERE userid=#id#
1036     </select>
1037 </sqlMap>
1038
1039 -com.javasoft.dao.UserDaoImplJDBC1.java 생성
1040 @Repository("userDao1")
1041 public class UserDaoImplJDBC1 implements UserDao {
1042     @Override
1043     public UserVO read(String id) {
1044         Reader rd = null;
1045         SqlMapClient smc = null;
1046         UserVO userVO = null;
1047         try {
1048             rd = Resources.getResourceAsReader("SqlMapConfig.xml");
1049             smc = SqlMapClientBuilder.buildSqlMapClient(rd);
1050             userVO = (UserVO)smc.queryForObject("Users.useResultMap", id);
1051         } catch (IOException | SQLException e) {
1052             // TODO Auto-generated catch block
1053             e.printStackTrace();
1054         }
1055         return userVO;
1056     }
1057 }

```

1058 -com.javasoft.service.UserServiceImpl.java 수정

```

1060 @Service("userService")
1061 public class UserServiceImpl implements UserService {
1062
1063     @Autowired
1064     UserDao userDao1; //userDao에서 userDao1로 변경
1065
1066     ...
1067     @Override
1068     public UserVO getUser(String id) {
1069         return userDao1.read(id);
1070     }
1071 }

```

1072 -/src/test/java/MembershipTest.java

```

1074 @Test
1075 public void test() {
1076     //사용자 조회 테스트
1077     UserVO user = service.getUser("jimin");
1078     System.out.println(user);
1079     assertEquals("한지민", user.getName());
1080 }
1081
1082
1083

```

1084 2)사용자 등록 및 목록 조회 테스트

1085 -Users.xml

```

1086 <insert id="insert" parameterClass="userVO">
1087     INSERT INTO USERS(userid, name, gender, city)
1088     VALUES (#userId#, #name#, #gender#, #city#)
1089 </insert>
1090
1091 <select id="getAll" resultClass="userVO">
1092     SELECT * FROM USERS
1093 </select>
1094

```

1095 -UserDaoImplJDBC1.java

```

1096 @Override
1097 public void insert(UserVO user) {

```

```

1098     Reader rd = null;
1099     SqlMapClient smc = null;
1100     UserVO userVO = null;
1101     try {
1102         rd = Resources.getResourceAsReader("SqlMapConfig.xml");
1103         smc = SqlMapClientBuilder.buildSqlMapClient(rd);
1104         smc.insert("Users.insert", user);
1105         System.out.println("등록된 Record UserId=" + user.getUserId() + " Name=" +
            user.getName());
1106     } catch (IOException | SQLException e) {
1107         // TODO Auto-generated catch block
1108         e.printStackTrace();
1109     }
1110 }
1111
1112 @Override
1113 public List<UserVO> readAll() {
1114     Reader rd = null;
1115     SqlMapClient smc = null;
1116     List<UserVO> userList = null;
1117     try {
1118         rd = Resources.getResourceAsReader("SqlMapConfig.xml");
1119         smc = SqlMapClientBuilder.buildSqlMapClient(rd);
1120         userList = (List<UserVO>)smc.queryForList("Users.getAll", null);
1121     } catch (IOException | SQLException e) {
1122         // TODO Auto-generated catch block
1123         e.printStackTrace();
1124     }
1125     return userList;
1126 }

```

-MembershipTest.java

```

1129 @Test
1130 public void test1() {
1131     this.service.insertUser(new UserVO("dooly", "둘리", "남", "경기"));
1132     for(UserVO user : this.service.getUserList()){
1133         System.out.println(user);
1134     }
1135 }

```

3) 사용자 정보 수정 테스트

-Users.xml

```

1139 <update id="update" parameterClass="userVO">
1140     UPDATE USERS
1141     SET    name = #name#, gender = #gender#, city = #city#
1142     WHERE  userId = #userId#
1143 </update>

```

-com.javasoft.dao.UserDaoImplJDBC1.java 코드 수정

```

1146 @Override
1147 public void update(UserVO user) {
1148     Reader rd = null;
1149     SqlMapClient smc = null;
1150     UserVO userVO = null;
1151     try {
1152         rd = Resources.getResourceAsReader("SqlMapConfig.xml");
1153         smc = SqlMapClientBuilder.buildSqlMapClient(rd);
1154         smc.update("Users.update", user);
1155         System.out.println("갱신된 Record with ID = " + user.getUserId() );
1156     } catch (IOException | SQLException e) {
1157         // TODO Auto-generated catch block
1158         e.printStackTrace();
1159     }
1160 }

```

-MembershipTest.java 수정

```

1163 @Ignore @Test

```

```

1164     public void test1() {
1165         this.service.insertUser(new UserVO("dooly", "둘리", "남", "경기"));
1166         for(UserVO user : this.service.getUserList()){
1167             System.out.println(user);
1168         }
1169     }
1170
1171     @Test
1172     public void test2() {
1173         service.updateUser(new UserVO("dooly", "김둘리", "여", "부산"));
1174         UserVO user = service.getUser("dooly");
1175         System.out.println(user);
1176     }
1177
1178 4)사용자 정보 삭제 테스트
1179 -Users.xml
1180     <delete id="delete" parameterClass="String">
1181         DELETE FROM USERS WHERE  userid = #id#
1182     </delete>
1183
1184 -com.javasoft.dao.UserDaoImplJDBC.java 코드 수정
1185     @Override
1186     public void delete(String id) {
1187         Reader rd = null;
1188         SqlMapClient smc = null;
1189         UserVO userVO = null;
1190         try {
1191             rd = Resources.getResourceAsReader("SqlMapConfig.xml");
1192             smc = SqlMapClientBuilder.buildSqlMapClient(rd);
1193             smc.delete("Users.delete", id);
1194             System.out.println("삭제된 Record with ID = " + id );
1195         } catch (IOException | SQLException e) {
1196             // TODO Auto-generated catch block
1197             e.printStackTrace();
1198         }
1199     }
1200
1201 -MembershipTest.java 수정
1202     @Test
1203     public void test() {
1204         UserVO user = this.service.getUser("jimin");
1205         System.out.println(user);
1206         assertEquals("한지민", user.getName());
1207     }
1208     @Ignore @Test
1209     public void test1() {
1210         this.service.insertUser(new UserVO("dooly", "둘리", "남", "경기"));
1211         for(UserVO user : this.service.getUserList()){
1212             System.out.println(user);
1213         }
1214     }
1215
1216     @Ignore @Test
1217     public void test2() {
1218         service.updateUser(new UserVO("dooly", "김둘리", "여", "부산"));
1219         UserVO user = service.getUser("dooly");
1220         System.out.println(user);
1221     }
1222
1223     @Test
1224     public void test3() {
1225         //사용자 정보 삭제 테스트
1226         service.deleteUser("dooly");
1227         for(UserVO user : service.getUserList()){
1228             System.out.println(user);
1229         }
1230     }

```

14. MyBatis를 이용한 Membership Project

1) 사용자 조회 테스트

-Project Build path에 mybatis-3.4.5.jar 등록

-src/main/resources/dbinfo.properties

db.driverClass=oracle.jdbc.driver.OracleDriver

db.url=jdbc:oracle:thin:@localhost:1521:XE

db.username=scott

db.password=tiger

-src/main/java/mybatis-config.xml

```
<?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>
    <properties resource="dbinfo.properties" />
    <typeAliases>
        <typeAlias type="com.javasoft.vo.UserVO" alias="userVO" />
    </typeAliases>
    <environments default="development">
        <environment id="development">
            <transactionManager type="JDBC"/>
            <dataSource type="POOLED">
                <property name="driver" value="${db.driverClass}"/>
                <property name="url" value="${db.url}"/>
                <property name="username" value="${db.username}"/>
                <property name="password" value="${db.password}"/>
            </dataSource>
        </environment>
    </environments>
    <mappers>
        <mapper resource="com/javasoft/dao/mybatis-mapper.xml"/>
    </mappers>
</configuration>
```

-com.javasoft.dao/mybatis-mapper.xml

```
<?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="com.javasoft.vo.UserVO">
    <resultMap id="userVOResult" type="userVO">
        <result property="userId" column="userid" />
        <result property="name" column="name" />
        <result property="gender" column="gender" />
        <result property="city" column="city" />
    </resultMap>
    <select id="select" parameterType="String" resultType="userVO"
        resultMap="userVOResult">
        SELECT * FROM USERS WHERE userid = #{id}
    </select>
</mapper>
```

-UserServiceImpl.java 수정

@Service("userService")

public class UserServiceImpl implements UserService {

@Autowired

UserDao userDao2;

-UserDaoImplJDBC2.java 수정

@Repository("userDao2")

```

1297 public class UserDaoImplJDBC2 implements UserDao {
1298     ...
1299     @Override
1300     public UserVO read(String id) {
1301         Reader rd = null;
1302         SqlSession session = null;
1303         UserVO userVO = null;
1304         try {
1305             rd = Resources.getResourceAsReader("mybatis-config.xml");
1306             session = session = new SqlSessionFactoryBuilder().build(rd).openSession();
1307             userVO = (UserVO)session.selectOne("select", id);
1308         } catch (IOException e) {
1309             // TODO Auto-generated catch block
1310             e.printStackTrace();
1311         }
1312         return userVO;
1313     }
1314 }

```

1315 -MembershipTest.java

```

1316 @RunWith(SpringJUnit4ClassRunner.class)
1317 @ContextConfiguration(locations="classpath:beans.xml")
1318 public class MembershipTest {
1319     ...
1320     @Autowired
1321     UserService service;
1322     ...
1323     @Test
1324     public void test() {
1325         UserVO user = this.service.getUser("jimin");
1326         System.out.println(user);
1327         assertEquals("한지민", user.getName());
1328     }
1329 }

```

1330 2)사용자 등록 및 목록 조회 테스트

1331 -mybatis-mapper.xml

```

1332 <insert id="insert" parameterType="userVO">
1333     INSERT INTO USERS(userid, name, gender, city)
1334     VALUES ({userId}, {name}, {gender}, {city})
1335 </insert>
1336
1337 <select id="selectAll" resultType="userVO" resultMap="userVOResult">
1338     SELECT * FROM USERS
1339 </select>

```

1340 -UserDaoImplJDBC2.java

```

1341 @Override
1342 public void insert(UserVO user) {
1343     Reader rd = null;
1344     SqlSession session = null;
1345     UserVO userVO = null;
1346     try {
1347         rd = Resources.getResourceAsReader("mybatis-config.xml");
1348         session = new SqlSessionFactoryBuilder().build(rd).openSession();
1349         session.insert("insert", user);
1350         session.commit();
1351         System.out.println("등록된 Record UserId=" + user.getUserId() + " Name=" +
1352             user.getName());
1353     } catch (IOException e) {
1354         e.printStackTrace();
1355     }
1356 }
1357
1358 @Override
1359 public List<UserVO> readAll() {

```

```

1363         Reader rd = null;
1364         SqlSession session = null;
1365         List<UserVO> userList = null;
1366         try {
1367             rd = Resources.getResourceAsReader("mybatis-config.xml");
1368             session = new SqlSessionFactoryBuilder().build(rd).openSession();
1369             userList = session.selectList("selectAll");
1370         } catch (IOException e) {
1371             e.printStackTrace();
1372         }
1373         return userList;
1374     }

```

1375 -MembershipTest.java

```

1376
1377
1378     @Autowired
1379     UserService service;
1380
1381     @Test
1382     public void test() {
1383         UserVO user = this.service.getUser("jimin");
1384         System.out.println(user);
1385         assertEquals("한지민", user.getName());
1386     }
1387     @Test
1388     public void test1() {
1389         this.service.insertUser(new UserVO("dooly", "둘리", "남", "경기"));
1390         for(UserVO user : this.service.getUserList()){
1391             System.out.println(user);
1392         }
1393     }
1394

```

1395 3)사용자 정보 수정 테스트

1396 -mybatis-mapper.xml

```

1397
1398     <update id="update" parameterType="userVO">
1399         UPDATE USERS SET name = #{name}, gender = #{gender}, city = #{city}
1400         WHERE userid = #{userId}
1401     </update>
1402

```

1403 -UserDaoImplJDBC2.java

```

1404
1405     @Override
1406     public void update(UserVO user) {
1407         Reader rd = null;
1408         SqlSession session = null;
1409         UserVO userVO = null;
1410         try {
1411             rd = Resources.getResourceAsReader("mybatis-config.xml");
1412             session = new SqlSessionFactoryBuilder().build(rd).openSession();
1413             session.update("update", user);
1414             session.commit();
1415             System.out.println("갱신된 Record with ID = " + user.getUserId() );
1416         } catch (IOException e) {
1417             e.printStackTrace();
1418         }
1419     }
1420

```

1421 -MembershipTest.java

```

1422
1423     @Autowired
1424     UserService service;
1425
1426     @Test
1427     public void test() {
1428         UserVO user = this.service.getUser("jimin");
1429         System.out.println(user);

```

```

1430         assertEquals("한지민", user.getName());
1431     }
1432     @Ignore @Test
1433     public void test1() {
1434         this.service.insertUser(new UserVO("dooly", "둘리", "남", "경기"));
1435         for(UserVO user : this.service.getUserList()){
1436             System.out.println(user);
1437         }
1438     }
1439
1440     @Test
1441     public void test2() {
1442         service.updateUser(new UserVO("dooly", "김둘리", "여", "부산"));
1443         UserVO user = service.getUser("dooly");
1444         System.out.println(user);
1445     }
1446

```

4) 사용자 정보 삭제 테스트

-mybatis-mapper.xml

```

1450     <delete id="delete" parameterType="String">
1451         DELETE FROM USERS WHERE  userid = #{id}
1452     </delete>
1453

```

-UserDaoImplJDBC2.java

```

1456     @Override
1457     public void delete(String id) {
1458         Reader rd = null;
1459         SqlSession session = null;
1460         UserVO userVO = null;
1461         try {
1462             rd = Resources.getResourceAsReader("mybatis-config.xml");
1463             session = new SqlSessionFactoryBuilder().build(rd).openSession();
1464             session.delete("delete", id);
1465             session.commit();
1466             System.out.println("삭제된 Record with ID = " + id );
1467         } catch (IOException e) {
1468             e.printStackTrace();
1469         }
1470     }
1471

```

-MembershipTest.java

```

1474     @Autowired
1475     UserService service;
1476
1477     @Test
1478     public void test() {
1479         UserVO user = this.service.getUser("jimin");
1480         System.out.println(user);
1481         assertEquals("한지민", user.getName());
1482     }
1483     @Ignore @Test
1484     public void test1() {
1485         this.service.insertUser(new UserVO("dooly", "둘리", "남", "경기"));
1486         for(UserVO user : this.service.getUserList()){
1487             System.out.println(user);
1488         }
1489     }
1490
1491     @Ignore @Test
1492     public void test2() {
1493         service.updateUser(new UserVO("dooly", "김둘리", "여", "부산"));
1494         UserVO user = service.getUser("dooly");
1495         System.out.println(user);
1496     }

```

```

1497
1498 @Test
1499 public void test3() {
1500     //사용자 정보 삭제 테스트
1501     service.deleteUser("dooly");
1502     for(UserVO user : service.getUserList()){
1503         System.out.println(user);
1504     }
1505 }
1506

```

15. JdbcTemplate를 이용한 Membership Project

1) 사용자 조회 테스트

-com.javasoft.dao.UserDaoImplJDBC.java 수정

```

1511 @Repository("userDao")
1512 public class UserDaoImplJDBC implements UserDao {
1513     private JdbcTemplate jdbcTemplate;
1514
1515     @Autowired
1516     public void setDataSource(DataSource dataSource) {
1517         this.jdbcTemplate = new JdbcTemplate(dataSource);
1518     }
1519
1520     class UserMapper implements RowMapper<UserVO> {
1521         public UserVO mapRow(ResultSet rs, int rowNum) throws SQLException {
1522             UserVO user = new UserVO();
1523             user.setUserId(rs.getString("userid"));
1524             user.setName(rs.getString("name"));
1525             user.setGender(rs.getString("gender"));
1526             user.setCity(rs.getString("city"));
1527             return user;
1528         }
1529     }
1530
1531     ...
1532     @Override
1533     public UserVO read(String id) {
1534         String SQL = "SELECT * FROM users WHERE userid = ?";
1535         try {
1536             UserVO user = jdbcTemplate.queryForObject(SQL,
1537                 new Object[] { id }, new UserMapper());
1538             return user;
1539         } catch (EmptyResultDataAccessException e) {
1540             return null;
1541         }
1542     }
1543

```

-com.javasoft.service.UserServiceImpl.java 수정

```

1546 @Service("userService")
1547 public class UserServiceImpl implements UserService {
1548
1549     @Autowired
1550     UserDao userDao;
1551
1552     ...
1553     @Override
1554     public UserVO getUser(String id) {
1555         return userDao.read(id);
1556     }
1557

```

~/src/test/java/MembershipTest.java

```

1560 @Test
1561 public void test() {
1562     //사용자 조회 테스트
1563     UserVO user = service.getUser("jimin");

```



```

1564         System.out.println(user);
1565         assertEquals("한지민", user.getName());
1566     }
1567     *****
1568 java.lang.NoClassDefFoundError: Could not initialize class
org.springframework.jdbc.core.StatementCreatorUtils
<에러 해결 방법>
    <dependency>
1571     <groupId>org.springframework</groupId>
1572     <artifactId>spring-context</artifactId>
1573     <version>4.3.13.RELEASE</version>
1574     </dependency>

```

1575
1576 -버전 변경 후 Maven Clean -> Maven Install

1577 2)사용자 등록 및 목록 조회 테스트

1578 -com.javasoft.dao.UserDaoImplJDBC.java 코드 수정

```

1581     @Override
1582     public void insert(UserVO user) {
1583         String SQL = "INSERT INTO users (userid, name, gender,city) VALUES (?, ?, ?, ?)";
1584         jdbcTemplate.update(SQL, user.getUserId(), user.getName(), user.getGender(),
1585             user.getCity());
1586
1587         System.out.println("등록된 Record UserId=" + user.getUserId() + " Name=" +
1588             user.getName());
1589     }
1590
1591     @Override
1592     public List<UserVO> readAll() {
1593         String SQL = "SELECT * FROM users";
1594         List<UserVO> userList = jdbcTemplate.query(SQL, new UserMapper());
1595         return userList;
1596     }

```

1597 -com.javasoft.service.UserServiceImpl.java 코드 수정

```

1598
1599     @Override
1600     public void insertUser(UserVO user) {
1601         userDao.insert(user);
1602     }
1603
1604     @Override
1605     public List<UserVO> getUserList() {
1606         return userDao.readAll();
1607     }
1608

```

1609 -/src/test/java/MembershipTest.java

```

1610
1611     @Test
1612     public void test1() {
1613         //사용자 등록 및 목록조회 테스트
1614         this.service.insertUser(new UserVO("dooly", "둘리", "남", "경기"));
1615         for(UserVO user : this.service.getUserList()){
1616             System.out.println(user);
1617         }
1618     }
1619

```

1620 3)사용자 정보 수정 테스트

1621 -com.javasoft.dao.UserDaoImplJDBC.java 코드 수정

```

1622
1623     @Override
1624     public void update(UserVO user) {
1625         String SQL = "UPDATE users SET name = ?, gender = ?, city = ? WHERE userid = ?";
1626         jdbcTemplate.update(SQL, user.getName(), user.getGender(),
1627             user.getCity(),user.getUserId());

```

```

1627         System.out.println("갱신된 Record with ID = " + user.getUserId() );
1628     }
1629
1630 -com.javasoft.service.UserServiceImpl.java 코드 수정
1631
1632     @Override
1633     public void updateUser(UserVO user) {
1634         userDao.update(user);
1635     }
1636

```

1637 -/src/test/java/MembershipTest.java

```

1638
1639     @Ignore @Test
1640     public void test1() {
1641         //사용자 등록 및 목록조회 테스트
1642         this.service.insertUser(new UserVO("dooly", "둘리", "남", "경기"));
1643         for(UserVO user : this.service.getUserList()){
1644             System.out.println(user);
1645         }
1646     }
1647
1648     @Test
1649     public void test2() {
1650         //사용자 정보 수정 테스트
1651         service.updateUser(new UserVO("dooly", "김둘리", "여", "부산"));
1652         UserVO user = service.getUser("dooly");
1653         System.out.println(user);
1654     }
1655

```

1656 4)사용자 정보 삭제 테스트

1657 -com.javasoft.dao.UserDaoImplJDBC.java 코드 수정

```

1658
1659     @Override
1660     public void delete(String id) {
1661         String SQL = "DELETE FROM users WHERE userid = ?";
1662         jdbcTemplate.update(SQL, id);
1663         System.out.println("삭제된 Record with ID = " + id );
1664     }
1665

```

1666 -com.javasoft.service.UserServiceImpl.java 코드 수정

```

1667
1668     @Override
1669     public void deleteUser(String id) {
1670         userDao.delete(id);
1671     }
1672

```

1673 -/src/test/java/MembershipTest.java

```

1674
1675     @Test
1676     public void test3() {
1677         //사용자 정보 삭제 테스트
1678         service.deleteUser("dooly");
1679         for(UserVO user : service.getUserList()){
1680             System.out.println(user);
1681         }
1682     }

```