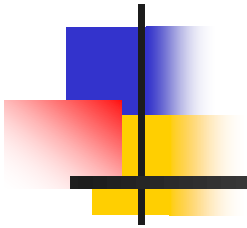
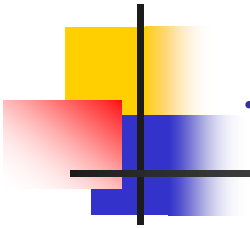


spring 3강-mybatis

양 명 숙
[now4ever7@gmail.com]

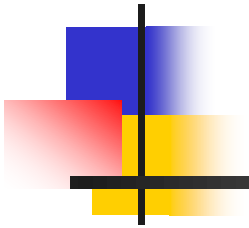


목차

- mybatis
- mybatis-스프링 연동

스프링의 데이터베이스 연동지원

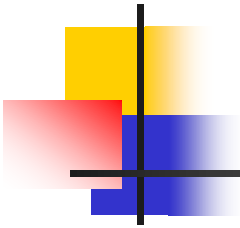
- 스프링은 다양한 데이터 액세스 기술들과의 통합에 필요한 데이터 액세스용 프레임워크 집합을 제공함
- 데이터를 저장할 때 직접 JDBC API를 쓰거나 mybatis를 쓸 수도 있다. 또는 하이버네이트(Hibernate)와 같은 객체 관계 매핑(ORM: Object-Relational Mapping) 프레임워크를 쓸 수도 있다.
- 스프링은 모든 경우에 지저분하고 반복적인 데이터 액세스 코드를 쓰지 않게 해준다
- 저수준의 데이터 액세스 관련 작업은 스프링에게 맡기고, 개발자는 애플리케이션의 데이터 관리 로직에만 집중할 수 있다.
- 스프링은 JDBC, 하이버네이트, mybatis 등의 다양한 기술을 이용해서 손쉽게 DAO 클래스를 구현할 수 있도록 지원하고 있는데, 지원하는 내용은 다음과 같다
  - 템플릿 클래스를 통한 데이터 접근 지원
  - 의미 있는 예외 클래스 제공
  - 트랜잭션 처리



mybatis

mybatis

- mybatis란
  - JDBC를 대체하는 퍼시스턴스 프레임워크
  - 데이터베이스 프로그래밍을 하는 데 기존 JDBC API를 사용할 때의 불편함을 없애줌
  - 마이바티스 내부에서는 JDBC API를 사용하고 개발자가 직접 JDBC를 사용할 때의 중복 작업 대부분을 없애줌
  - SQL을 별도의 XML이나 애노테이션으로 정의하기 때문에 SQL을 관리하기 편함
- 특징
  - JDBC를 작성할 때의 try/catch 구문을 사용할 필요가 없다
  - 객체 프로퍼티를 Prepared 구문의 파라미터로 자동으로 매핑함
  - 조회 결과를 객체로 자동으로 매핑함
  - 트랜잭션을 관리함
  - 스프링과 같은 외부 트랜잭션 관리자를 사용할 수도 있다.
  - 스프링 연동 모듈을 제공해서 스프링과 연동할 수도 있다.



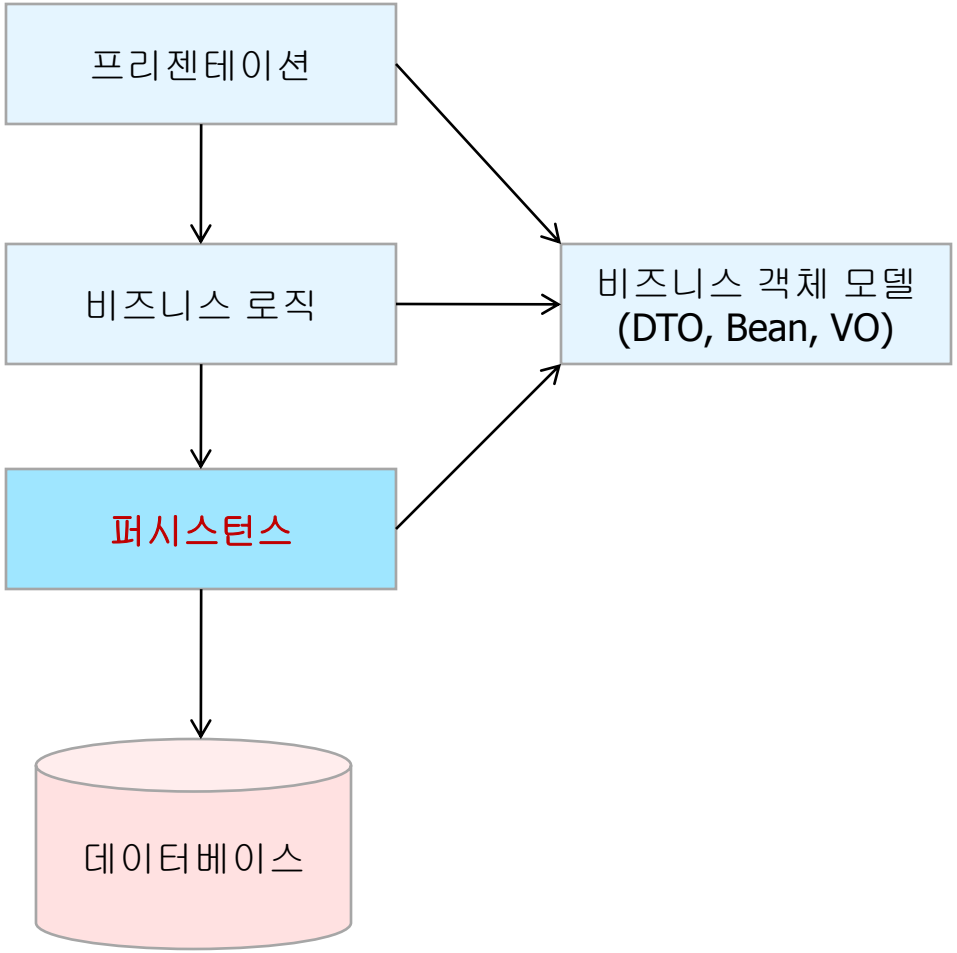
mybatis

- mybatis
  - SQL 을 이용하여 관계형 데이터베이스를 관리하는 Persistence Layer의 Framework
  - 더 빠른 JDBC 코딩을 위한 일반화된 프레임워크
  - 데이터 매퍼임

Data Mapping – SQL 구문을 자바 객체로 매핑

- 장점
  - 기존에 이미 sql을 습득하고 있는 개발자들이 배우기 쉬우며, 프로젝트에 빠르게 적용할 수 있음
  - 데이터베이스의 설계나 의존관계 등에 영향을 끼치지 않음 => 대규모 어플리케이션일 경우에도 적용에 무리가 없음

계층화된 설계



Data Mapper

- 객체와 데이터베이스 그리고 매퍼 자체를 독립적으로 유지하면서 객체와 데이터베이스 간에 데이터를 이동시킴
- SQL 구문 작성에 필요한 데이터를 클래스의 필드에서 추출하고, SQL 실행결과를 클래스에 매핑함

Value Object(VO, Bean)에 해당하는 부분이 SQL구문에 매핑됨
SQL 실행결과도 VO(Bean)객체로 매핑됨

JDBC를 이용하여 SQL 구문을 만들기 위하여 VO(Bean)객체의 속성을 직접 접근하거나
또 실행 결과를 VO(Bean) 객체로 담기 위해 일일이 ResultSet 에서 꺼내는 수고를 덜어줌

예

- member테이블

```
create table member(
        id number primary key,
        pwd  varchar2(10),
        name  varchar2(20),
        tel  varchar2(20)
);
```

- MemberVO 클래스

```
public class MemberVO{
        private int id;
        private String pwd;
        private String name;
        private String tel;
}
```

- mybatis 매핑 파일

```
[member-mapping.xml]
<select id="getMember" resultType="MemberVO" parameterType="java.lang.String">
                select id, pwd, name, tel from member where id=#{id}
</select>
```

• <select> 태그의 where 절에 #{id}로 표시된 String 형 값이 파라미터로 설정됨
• 이 쿼리의 결과가 MemberVO 클래스의 객체로 매핑됨

• MemberVO 클래스에는 select 명령어의 결과로 추출된 각 컬럼의 이름과 동일한 이름의 프로퍼티가 포함되어 있어 **mybatis가 MemberVO 객체를 생성하고 각각의 필드를 찾아 자동으로 매핑함**

• 필요한 데이터를 얻기 위한 쿼리를 자바 소스에 두지 않고 XML 파일로 관리하기 위해서임
• xml 을 사용하여 외부로 쿼리를 분리하게 되면, 변경사항이 발생할 때 신속하게 수정,
•Persistence Layer 를 완벽하게 독립시킬 수 있어 유지 보수성을 향상시킬 수 있음

예

```
[member-mapping.xml]

<insert id="insertMember" parameterType="MemberVO">
            insert into member(id, pwd, name, tel)
            values(member_seq.nextval , #{pwd}, #{name}, #{tel})
</insert>
```

```
sqlSession =  sqlSessionFactory.openSession();

int n= (int)sqlSession.insert("insertMember", memberVo);
```

여러개 중 ^ 와 같은 이름을 불러옴

mybatis 프로그램 절차

- jdbc는 필요 이상의 많은 코딩과 자원관리를 요구함 : 해결방법
- => mybatis를 사용하면 jdbc와 관련한 많은 코드들을 개발자가 직접 작성하지 않고, mybatis에게 위임할 수 있음
- [1] 매핑 파일을 작성함

```
<select id="getMember" resultType="MemberVO" parameterType="java.lang.String">
            select id, pwd, name, tel from member where id=#{id}
</select>
```

- [2] 매핑 파일을 이용해서 데이터를 조회함

```
sqlSession =  sqlSessionFactory.openSession();

MemberVO  vo= sqlSession.selectOne("getMember", "guest");     <---이게 #{id}의 id에 들감
```

list 받을 때는 selectList

데이터 매퍼 마이바티스의 역사

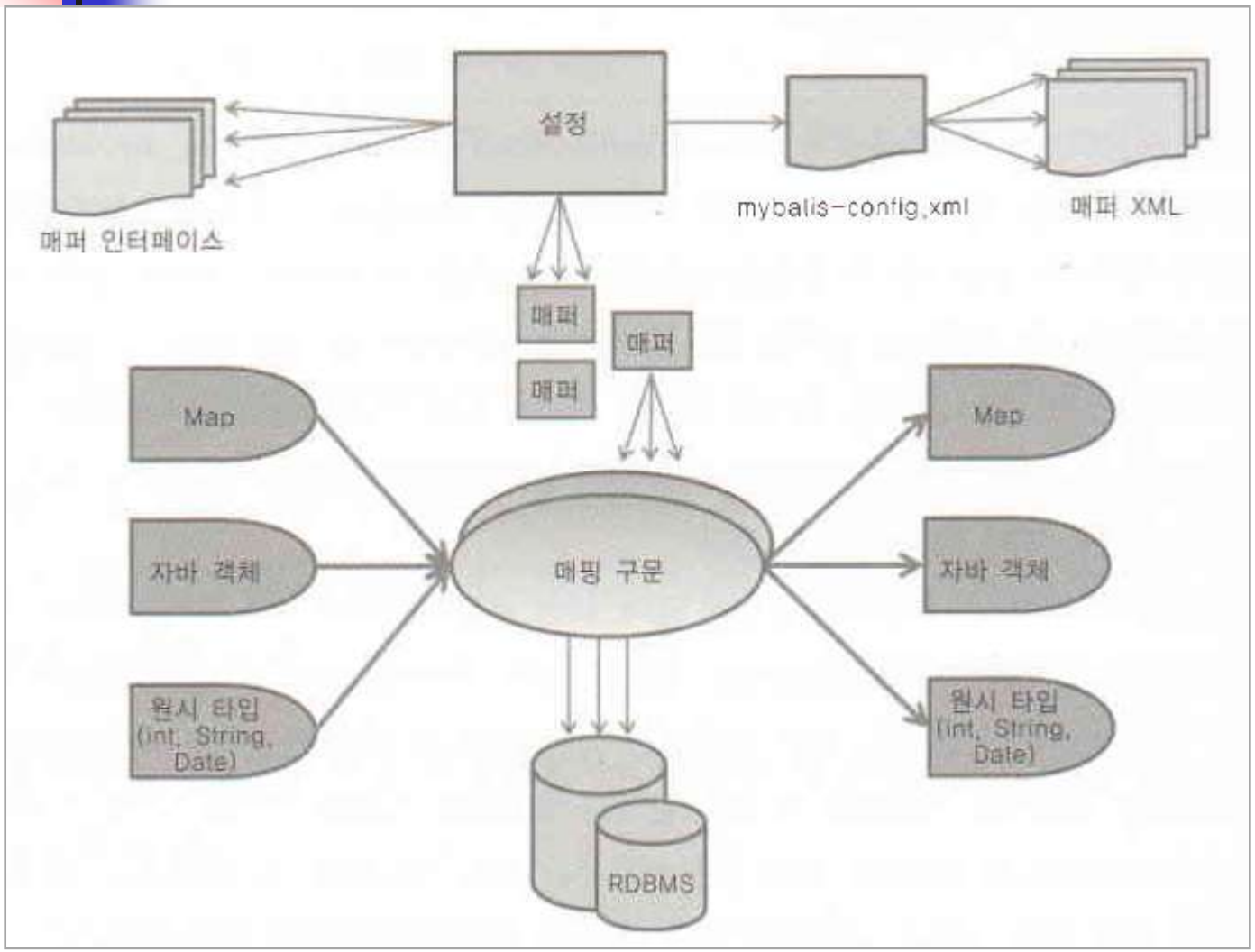
- 마이바티스(mybatis)의 전신인 아이바티스(ibatis)는 클린턴 비긴(Clinton Begin)이 2001년 시작한 암호 및 보안 관련 프로젝트였다
- 2003년 2월 - 아이바티스 데이터베이스 레이어 1.1.0을 정식 릴리스
- 2.x 버전 - 2004년 6월 아이바티스 SQLMaps와 DAO 두 가지 모듈을 묶어 릴리스
- 2006년 7월 - 루비를 위한 아이바티스를 릴리스
- 2006년 12월 2.3버전 릴리스- SQLMaps와 DAO모듈 중 DAO 모듈을 더이상 함께 릴리스하지 않기로 결정
- 2010년 4월 - 3.0버전인 마이바티스를 처음으로 릴리스
- iBatis - internet + abatis 를 합친 용어, 인터넷을 위한 암호 및 보안 관련 제품을 시작하면서 사용했다
- 프로젝트를 구글로 옮기면서 이름을 변경, abatis만 유지해서 mybatis로 명명
- 마이바티스는 클린턴 비긴, 래리 메더스, 브랜든 구딘의 메인 관리자 3명과 9명의 기여자 등 총 12명이 참여

데이터 매퍼 마이바티스의 역사

- 마이바티스(mybatis)는 아이바티스(ibatis)에서 시작했고, 몇 가지 차이점이 있지만, 기본 사용법은 아이바티스와 거의 동일함
- 마이바티스는 아이바티스에 한두 가지 제약 사항이 추가되고, 많은 기능이 추가된 제품으로 보면 됨
- **아이바티스와 마이바티스 비교**

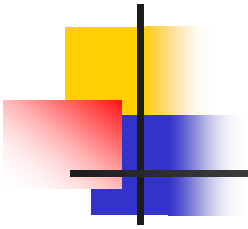
구분	아이바티스	마이바티스
네임스페이스	선택 사항	필수 사항
매핑 구문 정의	XML만 사용	XML과 애노테이션 사용
동적 SQL	XML 엘리먼트만 사용 동적 SQL을 위한 XML 엘리먼트는 16개 내외	XML 엘리먼트 및 구문 빌더 사용 동적 SQL을 위한 XML 엘리먼트는 4개 내외
스프링 연동	스프링 자체 구현체 사용	마이바티스 별도 모듈 사용
지원 계획	향후 아이바티스에 대한 공식적인 릴리스는 없을 것으로 보임	향후 계속 개선돼 릴리스될 예정

마이바티스 구조


설정
매퍼 인터페이스
mybatis-config.xml
매퍼 XML
매퍼
매퍼
매퍼
Map
자바 객체
원시 타입
(int, String,
Date)
매핑 구문
Map
자바 객체
원시 타입
(int, String,
Date)
RDBMS

마이바티스 구조

- 마이바티스를 구성하는 요소
  - 설정파일(mybatis-config.xml)
    - 데이터베이스 설정과 트랜잭션 등 마이바티스가 동작하는 규칙을 정의함
  - 매퍼 테이블 당 매퍼 1개씩 있어야함
    - SQL을 XML에 정의한 매퍼 XML 파일(1개 이상)과 SQL을 인터페이스마다 애노테이션으로 정의한 매퍼 인터페이스(1개 이상)를 의미함
  - 결과 매핑과 매핑 구문
    - 조회 결과를 자바 객체에 설정하는 규칙을 나타내는 결과 매핑과 SQL을 XML에 정의한 매핑구문을 말함
  - 지원하는 파라미터 타입
    - Map 객체, 자바 모델 클래스, 원시타입(int, String 등)
  - 지원하는 결과 타입
    - Map 객체, 자바 모델 클래스, 원시타입(int, String 등)



마이바티스 라이브러리

- http://code.google.com/p/mybatis/
  - https://github.com/mybatis/

- https://github.com/mybatis/mybatis-3/releases
  - mybatis-3.4.2.jar

jdbc코드를 마이바티스 코드로 변경하는 과정

- 1. jdbc에서 데이터베이스 연결을 생성하는 코드즉, getConnection() 은 다음 두 가지로 나뉜다
  - [1] JDBC 클래스명과 연결을 위한 URL 정보 및 계정 정보는 마이바티스에서 설정 파일이 됨
    - mybatis-config.xml
  - [2] DriverManager.getConnection() 과 같이 데이터베이스 연결을 생성하는 API 호출은 마이바티스가 제공하는 객체를 생성하는 것으로 바뀐다
    - 이 작업에서는 SqlSessionFactory 에서 SqlSession 객체를 생성한다
- 2. SQL을 사용하는 CRUD 에 대해서 처리
  - [1] SQL은 마이바티스의 매핑 구문으로 만든다
  - [2] 데이터베이스 연결에 대한 API 호출은 마이바티스 API를 호출하는 것으로 바뀐다
    - 이 호출은 대개 마이바티스의 SqlSession이 제공하는 selectOne, selectList, insert, update, delete를 호출한다

jdbc코드를 마이바티스 코드로 변경하는 과정

- 3. 데이터베이스 자원을 해제하는 작업 – 마이바티스에서 SqlSession 객체의 close() 메서드를 호출하는 것으로 바뀐다

- 대부분의 프레임워크는 설정 파일을 한 개 이상 가지며, 포맷은 대부분 xml 을 채택
  - 마이바티스도 xml 설정 파일을 한 개 만들어야 함
  - 마이바티스의 설정 파일은 마이바티스의 전반적인 환경 설정을 가진다.
    - 환경 설정의 대표적인 값은 데이터베이스 연결 정보와 매퍼(SQL을 선언한 별도의 XML이나 애노테이션을 가진 인터페이스)의 위치를 지정함
  - 설정 파일이 준비되면 설정 파일의 내용을 갖는 객체를 생성해야 함
    - 이 객체를 생성하기 위해서는 설정 파일을 파싱하고 로드하는 과정을 거쳐야 함
      - 설정 파일을 파싱하고 로드하는 것은 마이바티스 객체가 알아서 처리함
  - 이렇게 생성한 객체는 데이터베이스 연결 정보와 매퍼의 정보를 갖기 때문에 이후 데이터베이스 연동 과정에서 필요한 다양한 작업을 처리한다

마이바티스 설정 파일

- JDBC 코드를 마이바티스 코드로 변환하기 위해서는 먼저 마이바티스 설정 파일을 만든다
  - mybatis-config.xml
  - 설정 파일은 getConnection 메서드의 데이터베이스 연결 정보를 대체한다

[1] 트랜잭션 관리자
- JDBC 코드를 대체하기 때문에 type은 JDBC 로 지정
- JDBC 외에도 MANAGED 를 지정할 수 있다

[2] 데이터베이스 설정
- 데이터베이스 연결 정보를 설정한다

[3] 매퍼 정보 설정
- sql을 선언해둔 XML 이나 인터페이스 형태의 매퍼 위치를 지정해줘야 한다
- XML의 위치는 클래스 패스를 기준으로 지정하면 됨

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>
    <typeAliases>
        <typeAlias type="ldg.mybatis.model.Comment" alias="Comment" />
    </typeAliases>
    <environments default="development">
        <environment id="development">
            <transactionManager type="JDBC" />
            <dataSource type="POOLED">
                <property name="driver" value="oracle.jdbc.driver.OracleDriver" />
                <property name="url" value="jdbc:oracle:thin:@user-00:1521:orcl" />
                <property name="username" value="herb" />
                <property name="password" value="herb123" />
            </dataSource>
        </environment>
    </environments>
    <mappers>
        <mapper resource="ldg/mybatis/repository/mapper/CommentMapper.xml" />
    </mappers>
</configuration>
```

마이바티스 객체 생성하기 (SqlSessionFactory)

- 마이바티스 코드로 변환하기 위해 가장 먼저 설정 파일을 만들고,
- 두 번째, **설정 파일을 로드해 마이바티스 객체를 생성**한다
  - 이 마이바티스 객체는 SQL을 선언하는 것을 제외하고 JDBC 코드가 처리했던 대부분을 내부적으로 처리한다고 봐도 무방하다

```
public class CommentSessionRepository {
        private SqlSessionFactory getSqlSessionFactory() {
                String resource = "mybatis-config.xml";
                InputStream inputStream;
                try {
                        inputStream = Resources.getResourceAsStream(resource);
                } catch (IOException e) {
                        throw new IllegalArgumentException(e);
                }
                return new SqlSessionFactoryBuilder().build(inputStream);
        }
```

설정 파일을 로드해서 마이바티스 객체를 생성하는 코드

• 마이바티스 API를 사용해 설정 파일을 읽고, 데이터베이스와 트랜잭션 관리자, 그리고 매퍼의 정보를 가진 객체를 생성하는 코드
• 마이바티스를 사용하려면 getSqlSessionFactory() 메서드를 사용해 마이바티스 객체를 생성하고 API를 호출하면 된다

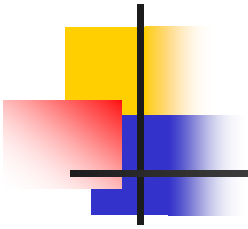
마이바티스 객체 생성하기

- [1] 마이바티스 설정 정보를 가진 객체 생성
  - inputStream = Resources.getResourceAsStream(resource);
  - 설정 파일 위치를 지정해 마이바티스 설정 정보에 대한 객체를 생성한다.
  - 설정파일의 이름은 mybatis-config.xml 이며, 클래스 패스를 기준으로 가장 상위에 있다
  - 설정 파일은 클래스 패스를 기준으로 파일명을 적어주면 됨
- [2] SqlSessionFactory 객체 생성
  - SqlSessionFactory 객체가 마이바티스의 전반적인 정보를 가지고 제어한다.
  - new SqlSessionFactoryBuilder().build(inputStream);
  - SqlSessionFactory 객체가 마이바티스의 전반적인 정보를 갖는 특성으로 인해 이 객체는 **애플리케이션 내에서 한 개만 생성**되어야 함

마이바티스 파라미터 표기법

- 마이바티스 파라미터 표기법
  - #{commentNo}
  - 값을 설정할 때 사용
  - 자바빈 – 프로퍼티명을 적어주면 됨
  - Map 객체 – key 값을 적어주면 됨
  - 파라미터의 값이 1개인 원시타입인 경우 #{}안에 아무 값이나 적어도 됨
  - 마이바티스는 파라미터 타입과 칼럼의 타입을 자동으로 처리하지 않기 때문에 Date 와 같은 일부 타입은 명시적으로 지정해줘야 개발자가 예상한 대로 동작함

```
#{connentNo, javaType=Date, jdbcType=TIMESTAMP}
```



데이터 조회

- 데이터를 조회하는 매핑 구문으로 분리
  - 마이바티스 코드로 변환하기 위해서는 먼저 sql 을 별도 XML에 분리하는 작업을 한다
  - 매핑구문 – 별도 XML 이나 애노테이션에서 선언한 SQL
  - 매퍼 XML – 매핑 구문을 XML 에 선언할 경우 이 XML 파일을 매퍼 XML 이라고 부른다

CommentVO

```
public class CommentVO  {
      private Long commentNo;
      private String userId;
      private Date regDate;
      private String commentContent;
…
```

```
<?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="ldg.mybatis.repository.mapper.CommentMapper">
            <select id="selectCommentByPrimaryKey" parameterType="long"
                                    resultType="ldg.mybatis.model.CommentVO">
                        SELECT
                        comment_no AS commentNo,
                        user_id AS userId,
                        comment_content AS commentContent,
                        reg_date AS regDate
                        FROM COMMENT
                        WHERE comment_no = #{commentNo}
            </select>
</mapper>
```

매퍼 XML
- CommentMapper.xml

매퍼 XML

본래 DAO 역은 여기서 다 처리함

- [1] xml 과 DOCTYPE 선언
- [2] 매퍼 네임스페이스
  - 매핑 구문들의 그룹
  - 마이바티스에서는 반드시 사용해야 함
  - 여러개의 매퍼에서 매핑 구문 아이디가 겹치더라도 마이바티스에서는 네임스페이스와 함께 매핑 구문 아이디를 사용하기 때문에 아이디가 겹치지 않게 분류할 수 있다
- [3] 매핑 구문
  - jdbc 코드에서 가져온 sql 과 XML 속성으로 구성
  - XML 속성
    - id – 마이바티스에서 이 SQL 을 사용하기 위한 id
    - parameterType – 파라미터 타입, 주로 자바빈이나 Map 사용
    - resultType – 결과 데이터 타입
    - SQL 종류에 따라 select/insert/update/delete 엘리먼트 사용

매핑 구문을 사용하는 마이바티스 코드 생성

- 자바코드에서 SQL 이 담긴 매핑 구문을 사용한다

```
public class CommentSessionRepository {
        private final String namespace = "ldg.mybatis.repository.mapper.CommentMapper";

        public Comment selectCommentByPrimaryKey(Long commentNo) {
                SqlSession sqlSession = getSqlSessionFactory().openSession();
                try {
                        return (Comment)sqlSession.selectOne(namespace +
".selectCommentByPrimaryKey", commentNo);
                } finally {
                        sqlSession.close();
                }
        }
```

[1] 마이바티스 객체 생성
- 마이바티스 매핑 구문을 호출해서 사용하려면 SqlSession 객체가 필요하며, 각 데이터베이스 작업별로 SqlSession 객체를 사용하면 됨

[2] 데이터 조회
- 데이터를 조회하기 위해 마이바티스 API를 사용
- SQL의 조회 조건에 파라미터를 설정한 뒤 SQL을 실행하고, 실행 후 조회 결과를 결과 객체에 설정하는 작업을 처리

[3] 데이터베이스 자원 해제

마이바티스 코드를 사용한 데이터 조회

```
public class CommentSessionRepositoryTest {
    private final CommentSessionRepository commentSessionRepository = new
    CommentSessionRepository();

    public void testSelectCommentByPrimaryKey() {
        Long commentNo = 1L;
        CommentVO comment =
    commentSessionRepository.selectCommentByPrimaryKey(commentNo);

        System.out.println(comment);
    }

    public static void main(String[] args) {
        CommentSessionRepositoryTest test = new CommentSessionRepositoryTest();

        test.testSelectCommentByPrimaryKey();

    }
}
```

데이터 입력, 수정, 삭제

```
public class CommentSessionRepository {
    private final String namespace = "ldg.mybatis.repository.mapper.CommentMapper";
    private SqlSessionFactory getSqlSessionFactory() {
        String resource = "mybatis-config.xml";
        InputStream inputStream;
        try {
            inputStream = Resources.getResourceAsStream(resource);
        } catch (IOException e) {
            throw new IllegalArgumentException(e);
        }
        return new SqlSessionFactoryBuilder().build(inputStream);
    }

    public Comment selectCommentByPrimaryKey(Long commentNo) {
        SqlSession sqlSession = getSqlSessionFactory().openSession();
        try {
            return (Comment)sqlSession.selectOne(namespace +
".selectCommentByPrimaryKey", commentNo);
        } finally {
            sqlSession.close();
        }
    }
```

```
public List<Comment> selectCommentByCondition(Map<String, Object> condition) {
        SqlSession sqlSession = getSqlSessionFactory().openSession();
        try {
                return sqlSession.selectList(namespace + ".selectCommentByCondition", condition);
        } finally {
                sqlSession.close();
        }
}
public Integer insertComment(Comment comment) {
        SqlSession sqlSession = getSqlSessionFactory().openSession();
        try {
                String statement = namespace + ".insertComment";
                int result = sqlSession.insert(statement, comment);
                if (result > 0) {
                        sqlSession.commit();
                } else {
                        sqlSession.rollback();
                }
                return result;
        } finally {
                sqlSession.close();
        }
}
```

```
    public Integer updateComment(Comment comment) {
        SqlSession sqlSession = getSqlSessionFactory().openSession();
        try {
                String statement = namespace + ".updateComment";
                int result = sqlSession.update(statement, comment);
                if (result > 0) {
                        sqlSession.commit();
                }
                return result;
        } finally {
                sqlSession.close();
        }
    }

    public Integer deleteComment(Long commentNo) {
        SqlSession sqlSession = getSqlSessionFactory().openSession();
        try {
                String statement = namespace + ".deleteComment";
                int result = sqlSession.delete(statement, commentNo);
                if (result > 0) {
                        sqlSession.commit();
                }
                return result;
        } finally {
                sqlSession.close();
        }
    }
}
```

CommentMapper.xml

```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="ldg.mybatis.repository.mapper.CommentMapper">
    <cache />
    <sql id="BaseColumns">
        comment_no AS commentNo,
        user_id AS userId,
        comment_content AS commentContent,
        reg_date AS regDate
    </sql>
    <select id="selectCommentByPrimaryKey" parameterType="long" resultType="ldg.mybatis.model.Comment">
        SELECT    <include refid="BaseColumns"/>
        FROM COMMENT
        WHERE comment_no = #{commentNo}
    </select>
    <select id="selectCommentByCondition" parameterType="hashmap"
                                resultType="ldg.mybatis.model.Comment">
        SELECT    comment_no AS commentNo,
                  user_id AS userId,
                  comment_content AS commentContent,
                  reg_date AS regDate
        FROM COMMENT
        <where>
                  <if test="commentNo != null">
                            comment_no = #{commentNo}
                  </if>
        </where>
    </select>
```

동적 SQL
-if 엘리먼트
- where 엘리먼트 : 하위 엘리먼트가 생성하는 내용이 있으면 자동으로 where를 붙여주는 역할을 함
- choose(when, otherwise), foreach, trim

#{commentNo }=> hashmap의 key

```
    <insert id="insertComment" parameterType="ldg.mybatis.model.Comment">
        INSERT INTO COMMENT(comment_no, user_id, comment_content, reg_date)
        VALUES (comment_seq.nextval, #{userId}, #{commentContent}, #{regDate})
    </insert>

    <update id="updateComment" parameterType="ldg.mybatis.model.Comment">
        UPDATE comment SET
                comment_content = #{commentContent}
        WHERE comment_no = #{commentNo};
    </update>

    <delete id="deleteComment" parameterType="long">
        DELETE FROM comment
        WHERE comment_no = #{commentNo};
    </delete>
</mapper>
```

```
public class CommentSessionRepositoryTest {
    private final CommentSessionRepository commentSessionRepository = new CommentSessionRepository();
    public void testInsertComment() {
            //Long commentNo = 1L;
            String userId = "fromm0";
            Date regDate = Calendar.getInstance().getTime();
            String commentContent = "test";

            Comment comment = new Comment();
            //comment.setCommentNo(commentNo);
            comment.setUserId(userId);
            comment.setCommentContent(commentContent);
            comment.setRegDate(regDate);

            Integer result = commentSessionRepository.insertComment(comment);

            System.out.println(result);
    }
    public void testUpdateComment() {
            Long commentNo = 1L;
            String commentContent = "수정 test";

            Comment comment = new Comment();
            comment.setCommentNo(commentNo);
            comment.setCommentContent(commentContent);
            Integer result = commentSessionRepository.updateComment(comment);
            System.out.println(result);
    }
```

```
    public void testDeleteComment() {
        Long commentNo = 1L;
        Integer result = commentSessionRepository.deleteComment(commentNo);

        System.out.println(result);
    }
    public void testSelectCommentByCondition() {
        Map<String, Object> condition = new HashMap<String, Object>();
        condition.put("commentNo", 2L);
        condition.put("commentNoForeach", new ArrayList<Long>());
        Comment comment = commentSessionRepository.selectCommentByCondition(condition);
        System.out.println(comment);
    }
    public void testSelectCommentByPrimaryKey() {
        Long commentNo = 1L;
        Comment comment = commentSessionRepository.selectCommentByPrimaryKey(commentNo);
        System.out.println(comment);
    }
    public static void main(String[] args) {
        CommentSessionRepositoryTest test = new CommentSessionRepositoryTest();

        test.testSelectCommentByPrimaryKey();
        test.testInsertComment();
        test.testUpdateComment();
        test.testDeleteComment();
        test.testSelectCommentByCondition();
    }
}
```

트랜잭션 관리

- 마이바티스를 사용할 때 중요한 객체는 SqlSessionFactory 객체이고, **각각의 세부 작업은 SqlSessionFactory 에서 만들어지는 SqlSession 객체가 담당**함
- SqlSession 객체를 생성하는 시점에 트랜잭션에 관련한 속성을 설정함
- SqlSessionFactory 클래스가 제공하는 openSession() 메서드 – 마이바티스 설정 파일의 설정을 그대로 사용하는 마이바티스 객체를 생성함
- openSession() 메서드를 사용해 SqlSession 객체를 생성하면 생성된 객체는 다음의 특성을 가진다
  - [1] 객체를 생성할 때마다 트랜잭션을 시작한다
  - [2] 마이바티스 설정 파일의 데이터 소스를 사용한다
  - [3] 트랜잭션에 관련한 격리 레벨이나 전파 설정은 설정한 값을 사용해 설정한다
  - [4] PreparedStatement 는 재사용되지 않고, 배치 형태로 처리하지 않는다

트랜잭션 처리

- 커밋과 롤백을 처리하기 위해 JDBC 드라이버는 필요한 메서드를 제공하고, 마이바티스는 그 JDBC 메서드를 사용하는 별도의 메서드를 추가로 제공한다
  - SqlSession 클래스의 메서드
    - commit()
    - rollback()
- 마이바티스 객체를 생성하는 openSession() 메서드를 호출하면 트랜잭션이 이와 동시에 명시적으로 시작된다
- 데이터를 입력, 수정, 삭제하면 데이터베이스에 명시적으로 적용하기 위해 트랜잭션을 제어하는 메서드를 호출해야 함
- commit 이나 rollback 메서드를 호출하는 것과 동시에 트랜잭션이 종료됨
- 스프링을 사용하지 않고 마이바티스를 사용할 때는 SqlSession 클래스가 제공하는 commit, rollback 메서드를 사용
- 스프링과 연동해 스프링 연동 모듈을 사용하면 트랜잭션에 대한 제어는 마이바티스가 담당하지 않고, 스프링에 위임함

조회 결과를 자바 객체에 설정(결과 매핑)

- DB에서 데이터를 가져온 후 자바 객체에 값을 설정하기 위해 마이바티스는 결과 매핑이라는 기법을 제공
  - 마이바티스 결과 매핑은 칼럼명과 자바 모델 클래스의 필드명이 일치하면 자동으로 값을 설정해 줌
  - 칼럼명과 일치하지 않는다면 별도로 값을 설정하는 규칙을 정의해야만 정확히 값을 설정할 수 있다
  - resultMap, id, result
  - constructor,
  - association, collection
  - discriminator

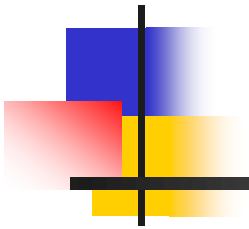
조회 결과를 자바 객체에 설정(결과 매핑)

```
<resultMap id="baseResultMap" type="Comment">
        <id column="comment_no" jdbcType="BIGINT" property="commentNo" />
        <result column="user_id" jdbcType="VARCHAR" property="userId" />
        <result column="reg_date" jdbcType="TIMESTAMP" property="regDate" />
        <result column="comment_content" jdbcType="VARCHAR" property="commentContent"
    />
</resultMap>
```

별칭을 사용하지 않는 칼럼값을 모델 객체에 설정하는 결과 매핑 설정

```
<select id="selectCommentByPrimaryKey" parameterType="long" resultMap=" baseResultMap">
        SELECT
                comment_no,
                user_id,
                comment_content,
                reg_date
        FROM comment
        WHERE comment_no = #{commentNo}
</select>
```

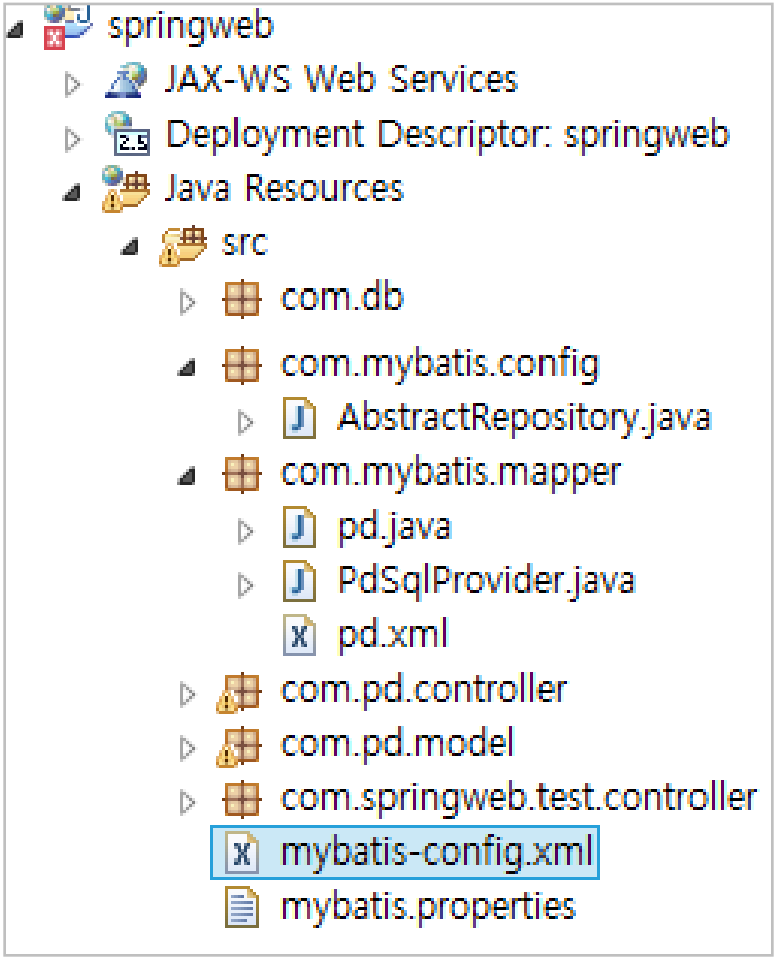
마이바티스 설정파일의 **mapUnderscoreToCamelCase** 설정을 **true** 로 설정하면 다음 규칙을 자동으로 적용함
comment_no => setCommentNo()



pd 테이블 – mybatis 예제

mybatis.properties

- jdbc.url=jdbc:oracle:thin:@303-0:1521:xe



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>
      <!-- 외부 프로퍼티 파일 로드 및 공통 프로퍼티 정의 -->
      <properties resource="mybatis.properties">
        <property name="jdbc.driver" value="oracle.jdbc.driver.OracleDriver" />
        <property name="jdbc.username" value="scott" />
        <property name="jdbc.password" value="tiger" />
      </properties>

      <!-- 타입별칭 -->
      <typeAliases>
            <typeAlias type="com.pd.model.PdBean" alias="PdBean" />
      </typeAliases>
```

```
    <!-- 데이터베이스및 트랜잭션 관리자 -->
    <environments default="development">
        <environment id="development">
            <transactionManager type="JDBC" />
            <dataSource type="POOLED">
                <property name="driver" value="${jdbc.driver}" />
                <property name="url" value="${jdbc.url}" />
                <property name="username" value="${jdbc.username}" />
                <property name="password" value="${jdbc.password}" />
            </dataSource>
        </environment>
    </environments>

    <!-- 매퍼정의 -->
    <mappers>
        <mapper resource="com/mybatis/mapper/pd.xml" />
    </mappers>
</configuration>
```

AbstractRepository

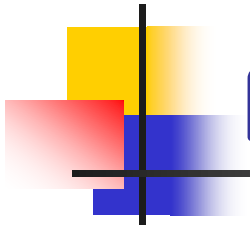
```
package com.mybatis.config;
import java.io.IOException;
import java.io.InputStream;
import org.apache.ibatis.io.Resources;
import org.apache.ibatis.session.SqlSessionFactory;
import org.apache.ibatis.session.SqlSessionFactoryBuilder;
public abstract class AbstractRepository {
	private static SqlSessionFactory sqlSessionFactory;

	static {
		setSqlSessionFactory();
	}

	private static void setSqlSessionFactory() {
		String resource = "mybatis-config.xml";
		InputStream inputStream;
		try {
			inputStream = Resources.getResourceAsStream(resource);
		} catch (IOException e) {
			throw new IllegalArgumentException(e);
		}
		sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);
	}

	protected SqlSessionFactory getSqlSessionFactory() {
		return sqlSessionFactory;
	}
}
```

pd.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.mybatis.mapper.pd">

    <insert id="pdInsert" parameterType="pdBean">
        <selectKey keyProperty="no" resultType="int" order="BEFORE">
            select pd_seq.nextval  as no from dual
        </selectKey>
        insert into pd(no, pdname, price)
        values(#{no}, #{pdName}, #{price})
    </insert>

    <select id="pdList" resultType="pdBean">
        select * from pd order by no desc
    </select>
```

pd.xml

```
    <select id="pdDetail" parameterType="int" resultType="pdBean">
            select * from pd where no=#{no}
    </select>

    <update id="pdUpdate" parameterType="pdBean">
            update pd set pdname=#{pdName}, price=#{price}
             where no=#{no}
    </update>

    <delete id="pdDelete" parameterType="int">
            delete from pd where no=#{no}
    </delete>
</mapper>
```



PdBean

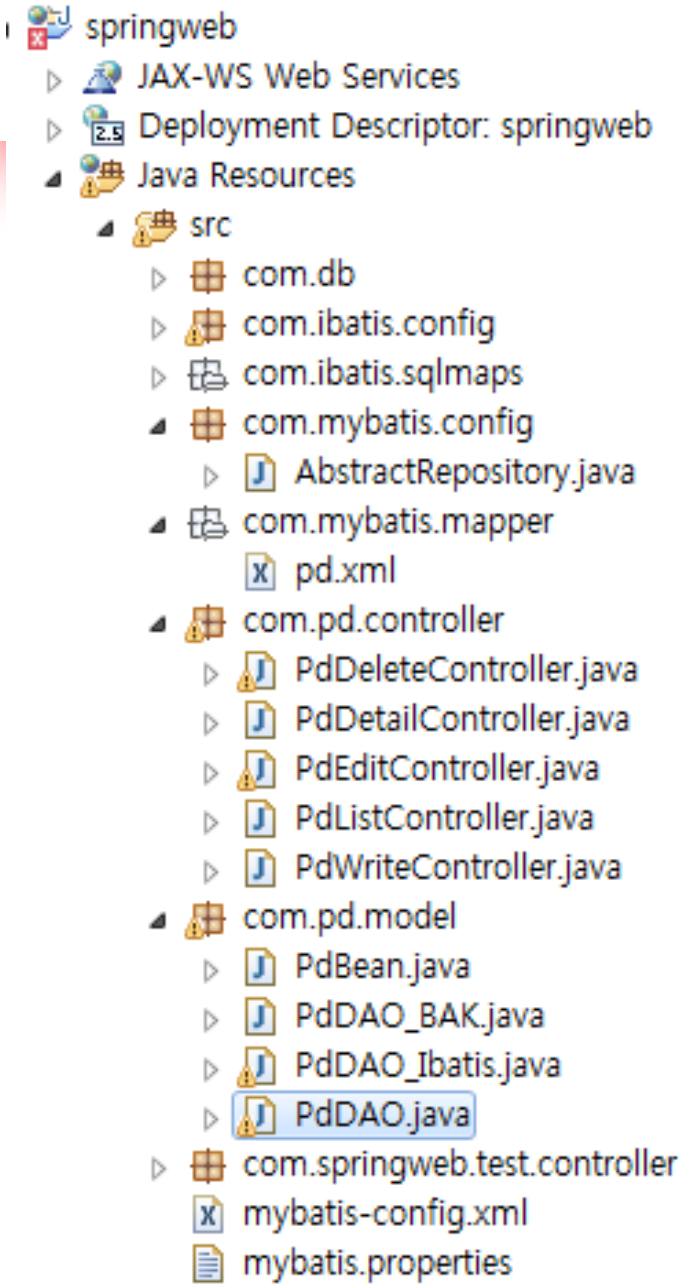
```
package com.pd.model;

import java.io.Serializable;
import java.sql.Timestamp;

public class PdBean {
	//멤버변수
	private int no;
	private String pdName;
	private int price;
	private Timestamp regdate;

...
}
```

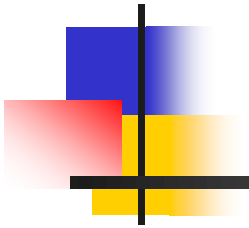
PdDAO

```
package com.pd.model;
import java.sql.SQLException;
import java.util.List;
import org.apache.ibatis.session.SqlSession;
import com.mybatis.config.AbstractRepository;

public class PdDAO extends AbstractRepository{
	private final String namespace = "com.mybatis.mapper.pd";
	private SqlSession sqlSession;

	public int insertPd(PdBean pdBean) {
		//pd 테이블에 insert하는 메서드
		sqlSession =  getSqlSessionFactory().openSession();
		try{
			int n = (int) sqlSession.insert(namespace + ".pdInsert", pdBean);
			if (n > 0) {
				sqlSession.commit();
			}
			return n;
		} finally {
			sqlSession.close();
		}
	}
```

```
public List<PdBean> selectAll()  {
	//전체 글을 조회하는 메서드
	sqlSession =  getSqlSessionFactory().openSession();
	try{
		List<PdBean> alist= sqlSession.selectList(namespace + ".pdList");

		return alist;
	} finally {
		sqlSession.close();
	}
}

public PdBean selectByNo(int no) {
	//no에 해당하는 글 조회
	sqlSession =  getSqlSessionFactory().openSession();
	try{
		PdBean bean = (PdBean) sqlSession.selectOne(namespace + ".pdDetail", no);
		return bean;
	} finally {
		sqlSession.close();
	}
}
```

```
    public int updatePd(PdBean bean) {
        sqlSession =  getSqlSessionFactory().openSession();
        try{
            int n= sqlSession.update(namespace + ".pdUpdate", bean);
            if (n > 0) {
                sqlSession.commit();
            }
            return n;
        } finally {
            sqlSession.close();
        }
    }

    public int deletePd(int no) {
        //no에 해당하는 상품 삭제
        sqlSession =  getSqlSessionFactory().openSession();
        try{
            int n=sqlSession.delete(namespace + ".pdDelete", no);
            if (n > 0) {
                sqlSession.commit();
            }
            return n;
        } finally {
            sqlSession.close();
        }
    }
}//class
```


springweb
JAX-WS Web Services
Deployment Descriptor: springweb
Java Resources
src
com.db
com.ibatis.config
com.ibatis.sqlmaps
com.mybatis.config
AbstractRepository.java
com.mybatis.mapper
pd.xml
com.pd.controller
PdDeleteController.java
PdDetailController.java
PdEditController.java
PdListController.java
PdWriteController.java
com.pd.model
PdBean.java
PdDAO_BAK.java
PdDAO_Ibatis.java
PdDAO.java
com.springweb.test.controller
mybatis-config.xml
mybatis.properties



마이바티스 설정파일

복잡한 마이바티스 설정파일

```
<?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>
    <!-- 외부 프로퍼티 파일 로드및 공통 프로퍼티 정의 -->
    <properties resource="mybatis.properties">
    <!--properties url="file:d:₩mybatis.properties"-->

            <property name="jdbc.driver" value="oracle.jdbc.driver.OracleDriver" />
            <!-- <property name="jdbc.url" value="jdbc:oracle:thin:@303-0:1521:xe" /> -->
            <property name="jdbc.username" value="herb" />
            <property name="jdbc.password" value="herb" />
    </properties>

    <!-- 마이바티스의 작동 규칙정의 -->
    <settings>
            <setting name="cacheEnabled" value="false" />
            <setting name="useGeneratedKeys" value="true" />
            <setting name="mapUnderscoreToCamelCase" value="true" />
            <setting name="autoMappingBehavior" value="PARTIAL" />
    </settings>
```

```
<!-- 타입별칭 -->
<typeAliases>
	<typeAlias type="ldg.mybatis.model.Comment" alias="Comment" />
	<typeAlias type="ldg.mybatis.model.User" alias="User" />
	<typeAlias type="ldg.mybatis.model.Reply" alias="Reply" />
	<typeAlias type="ldg.mybatis.model.CommentUser" alias="CommentUser" />
	<typeAlias type="ldg.mybatis.model.CommentReplies" alias="CommentReplies" />
</typeAliases>


<!-- 매퍼정의 -->
<mappers>
	<mapper resource="ldg/mybatis/repository/mapper/CommentMapper.xml" />
	<mapper resource="ldg/mybatis/repository/mapper/CommentMapperResultMap.xml" />
	<mapper resource="ldg/mybatis/repository/mapper/CommentMapperDynamicSql.xml" />
</mappers>
```

```
    <!-- 데이터베이스 및 트랜잭션 관리자 -->
    <environments default="development">
        <environment id="development">
            <transactionManager type="JDBC" />
            <dataSource type="POOLED">
                <property name="driver" value="${jdbc.driver}" />
                <property name="url" value="${jdbc.url}" />
                <property name="username" value="${jdbc.username}" />
                <property name="password" value="${jdbc.password}" />
            </dataSource>
        </environment>
        <environment id="release">
            <transactionManager type="JDBC" />
            <dataSource type="POOLED">
                <property name="driver" value="${jdbc.driver}" />
                <property name="url" value="${jdbc.url}" />
                <property name="username" value="${jdbc.username}" />
                <property name="password" value="${jdbc.password}" />
            </dataSource>
        </environment>
    </environments>

</configuration>
```

properties 엘리먼트

- properties
  - 공통적인 속성을 정의하거나 외부 파일에서 값을 가져와서 사용해야 하는 경우 사용
  - 프로젝트에서는 개발 장비와 운영 장비로 구분해서 사용하는 경우가 많은데, 서버 구분 없이 동일한 값은 그대로 설정 파일에 설정해서 사용하고, **서버별로 다른 값**은 **외부 프로퍼티 파일로 분리**한 후 서버별로 프로퍼티 파일을 선택해서 배포하는 형태를 주로 이용
  - properties 엘리먼트는 공통적인 환경 값을 정의해서 설정 파일에서 그 환경 값을 사용하거나
  - 외부 프로퍼티 파일에 환경 값을 설정하고 설정 파일에서 그 값을 사용할 수 있게 함
- 예) 외부 프로퍼티 파일 – mybatis.properties

```
jdbc.url=jdbc:oracle:thin:@yang-hp:1521:orcl
jdbc.driver=oracle.jdbc.driver.OracleDriver
jdbc.username=herb
jdbc.password=herb123
```

```
<properties resource="mybatis.properties" />
```

properties 엘리먼트

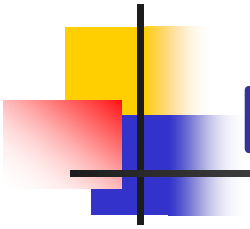
- 외부 프로퍼티 파일을 읽기 위해서는 properties 엘리먼트의 resource 속성에 위치를 지정
  - resource 속성은 클래스패스 기준으로 프로퍼티 파일을 찾는다
  - 서버별로 다른 값이 아닌 공통 속성은 외부 파일에 두지 않고, 하위 엘리먼트인 property 엘리먼트를 이용해 선언할 수 있다.

```
jdbc.url=jdbc:oracle:thin:@yang-hp:1521:orcl
```

```
<properties resource="mybatis.properties">
        <property name="jdbc.driver" value="oracle.jdbc.driver.OracleDriver" />
        <property name="jdbc.username" value="herb" />
        <property name="jdbc.password" value="herb123" />
</properties>
```

프로퍼티 파일을 클래스 패스 기준이 아닌 절대 경로로 잡아야 한다면 url 속성을 사용하면 됨

```
<properties url="file:d:\  mybatis.properties" />
```



properties 엘리먼트

- 이렇게 설정한 후 값을 사용할 때는 ${key} 형태로 사용하면 됨

```
<dataSource type="POOLED">
        <property name="driver" value="${jdbc.driver}" />
        <property name="url" value="${jdbc.url}" />
</dataSource>
```

settings 엘리먼트

- settings 엘리먼트를 사용해서 설정하는 각종 값은 SqlSessionFactory 객체가 SqlSession 객체를 만들 때 생성할 객체의 특성을 결정함
- settings 엘리먼트의 하위 엘리먼트들은 대부분 디폴트 값을 가진다
- 별도로 설정하지 않으면 디폴트값을 사용하는데, 특별한 경우가 아니면 디폴트값을 사용해도 문제없이 잘 작동함
- cacheEnabled – 캐시를 기본으로 사용할지를 결정, 디폴트값 : true
- useGeneratedKeys – 생성 키 사용여부를 결정, 디폴트값:false
  - MySql – auto_increment, 오라클 – sequence, SQL서버 – identity 를 생성키로 제공함
- mapUnderscoreToCamelCase – 언더 바 형태를 낙타 표현식으로 자동매핑 할지에 대한 옵션, 디폴트값:false(자동으로 매핑하지 않음)
  - 이 옵션을 사용하지 않으면서 테이블의 컬럼명은 언더 바로 구분하고 자바 모델 클래스는 낙타표기법을 사용할 경우 쿼리문에 칼럼별로 별칭을 사용하거나 별도의 결과 매핑을 사용해야 함

```
<!-- 마이바티스의 작동 규칙정의 -->
<settings>
  <setting name="cacheEnabled" value="false"/>
  <setting name="useGeneratedKeys" value="true"/>
  <setting name="mapUnderscoreToCamelCase" value="true"/>
</settings>
```

typeAliases 엘리먼트

- 매핑 구문의 파라미터나 결과 타입을 지정할 때 타입별 별칭을 설정할 수 있다
- typeAliases 엘리먼트를 사용해서 별칭을 설정하고 매핑 구문에서 파라미터 타입이나 결과 타입에 Comment 를 사용하면 ldg.mybatis.model.Comment 로 인식함
- 마이바티스는 애노테이션으로 설정하는 방법을 추가로 제공함

```
<typeAliases>
        <typeAlias type="ldg.mybatis.model.Comment" alias="Comment" />
        <typeAlias type="ldg.mybatis.model.User" alias="User" />
        <typeAlias type="ldg.mybatis.model.Reply" alias="Reply" />
        <typeAlias type="ldg.mybatis.model.CommentUser" alias="CommentUser" />
        <typeAlias type="ldg.mybatis.model.CommentReplies" alias="CommentReplies" />
</typeAliases>
```

```
package ldg.mybatis.model;

@Alias("Comment")
public class Comment {
}
```

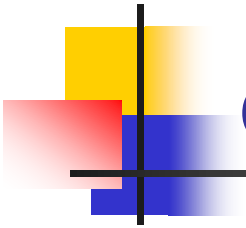
마이바티스가 미리 정의한 타입 별칭

- 원시타입이나 흔히 사용되는 자바 타입에 대해서는 마이바티스 내부에 미리 정의된 별칭이 있다.

별칭	매핑된 타입
_byte	byte
_long	long
_short	short
_int	int
_integer	int
_double	double
_float	float
_boolean	boolean
string	String
byte	Byte
long	Long
short	Short
int	Integer
integer	Integer
double	Double
float	Float
boolean	Boolean

마이바티스가 미리 정의한 타입 별칭

별칭	매핑된 타입
date	Date
decimal	BigDecimal
bigdecimal	BigDecimal
object	Object
map	Map
hashmap	HashMap
list	List
arraylist	ArrayList
collection	Collection
iterator	Iterator



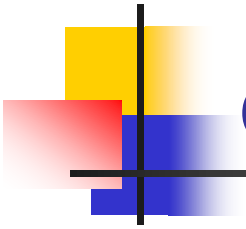
environments 엘리먼트

- environments 엘리먼트를 사용해서 마이바티스의 트랜잭션 관리자와 데이터 소스 2가지를 설정할 수 있다
- 트랜잭션 관리자와 데이터 소스는 마이바티스만 사용할 때 필요하며, 스프링 연동 모듈을 사용할 경우에는 필요 없다
- 트랜잭션 관리자 (transactionManager 엘리먼트)
  - transactionManager 엘리먼트 – 트랜잭션 관리자 클래스를 설정함
    - 트랜잭션 관리자는 트랜잭션을 어떻게 처리할 지 결정하는 객체다
  - type 속성
    - JDBC – 마이바티스 API에서 제공하는 commit, rollback 메서드 등을 사용해서 트랜잭션을 관리하는 방식
    - MANAGED – 마이바티스 API 보다는 컨테이너가 직접 트랜잭션을 관리하는 방식

environments 엘리먼트

- 데이터소스 (dataSource 엘리먼트)
  - dataSource 엘리먼트는 데이터 소스를 설정함
  - 데이터 소스는 데이터베이스의 정보를 갖는 객체다
  - type 속성 – UNPOOLED, POOLED, JNDI
    - POOLED – 일정 수의 데이터베이스 연결을 풀이라는 메모리 영역에 넣어두고 필요할 때마다 가져다가 사용하고, 사용하고 나면 다시 풀에 넣는다.
      - 대부분 서버를 시작할 때 애플리케이션이 올라가면서 설정된 수만큼의 데이터베이스 연결을 만들어 둠

```
<environments default="development">
        <environment id="development">
                <transactionManager type="JDBC" />
                <dataSource type="POOLED">
                        <property name="driver" value="${jdbc.driver}" />
                        <property name="url" value="${jdbc.url}" />
                        <property name="username" value="${jdbc.username}" />
                        <property name="password" value="${jdbc.password}" />
                </dataSource>
        </environment>
</environments>
```



environments 엘리먼트

- UNPOOLED – 데이터베이스에 요청할 때마다 데이터베이스 연결을 새롭게 생성하고 처리 후 완전히 해제한다.
  - 데이터베이스 연결을 매번 생성하는 것이 성능상 좋지 않기 때문에 실제 운영하는 애플리케이션에는 사용하지 않는다
- JNDI – 컨테이너의 JNDI 컨텍스트를 참조한다.
  - JNDI(Java Naming and Directory Interface) – 디렉토리 서비스를 위해 자바가 제공하는 인터페이스
  - 디렉토리 서비스 – 전화번호부처럼 산재된 정보를 쉽게 찾을 수 있도록 해주는 서비스, 여기서는 데이터 소스를 찾는다.
  - JNDI 설정은 톰캣과 같은 대부분의 웹 애플리케이션 서버가 제공함

mappers 엘리먼트

- 마이바티스에서 가장 중요한 매퍼를 지정하는 엘리먼트
- 매퍼는 매핑 구문과 파라미터나 결과 타입등을 지정하는 역할을 함
- 매퍼 위치를 지정하는 방법
  - [1] 클래스 패스에 위치한 XML 매퍼 파일 지정(resource 속성)
  - [2] URL을 사용한 XML 매퍼 파일 지정(url 속성)
  - [3] 매퍼 인터페이스를 사용하는 인터페이스 위치 지정 (class 속성)
  - [4] 패키지 지정으로 패키지 내 자동으로 매퍼 검색(name 속성)

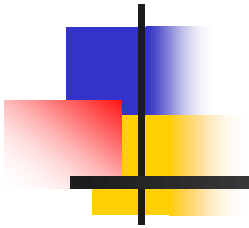
```
<mappers>
        <mapper resource="ldg/mybatis/repository/mapper/CommentMapper.xml" />
        <mapper url="file:///ldg/mybatis/repository/mapper/CommentMapperResultMap.xml" />
        <mapper class="ldg.mybatis.repository.mapper.CommentMapper" />
        <packgae name="ldg.mybatis" />
</mappers>
```

mappers 엘리먼트

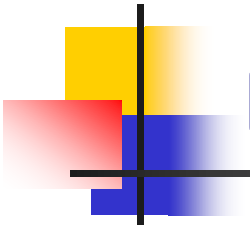
- 스프링 연동 모듈을 사용하면 mapperLocations 프로퍼티를 사용해 매퍼 위치를 지정할 수 있기 때문에 mappers 엘리먼트를 사용하지 않을 수도 있다

스프링 설정 파일

```
<bean id="sqlSessionFactoryBean" class="org.mybatis.spring.SqlSessionFactoryBean">
        <property name="typeAliasesPackage" value="ldg.mybatis.model" />
        <property name="dataSource" ref="dataSource" />
        <property name="configLocation" value="classpath:/mybatis-config.xml" />
        <property name="mapperLocations">
            <array>
                <value>classpath:/ldg/mybatis/repository/mapper/CommentMapper.xml</value>
                <!-- <value>classpath*:/ldg/mybatis/repository/mapper/**/*.xml</value> -->
            </array>
        </property>
</bean>
```

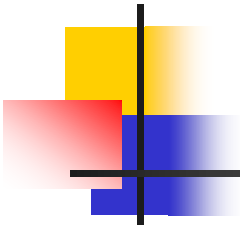


애퍼 XML



매퍼 XML

- 매퍼 XML을 구성할 수 있는 각 엘리먼트
  - resultMap
  - sql
  - insert, update, delete
  - selectKey
  - select



resultMap 엘리먼트

- 속성
  - id
    - 매핑 구문에서 결과 매핑을 사용할 때 구분하기 위한 아이디
  - type
    - 결과 매핑을 적용하는 대상 객체 타입
    - 매핑 구문의 결과 데이터를 갖는 자바 타입을 지정
    - 대개는 Map 이나 자바 모델 클래스를 지정함
- resultMap 엘리먼트의 하위 엘리먼트
  - id, result
  - constructor, association, collection, discriminator

```
<resultMap id="baseResultMap" type="Comment">
        <id column="comment_no" jdbcType="BIGINT" property="commentNo" />
        <result column="user_id" jdbcType="VARCHAR" property="userId" />
        <result column="reg_date" jdbcType="TIMESTAMP" property="regDate" />
        <result column="comment_content" jdbcType="VARCHAR" property="commentContent" />
</resultMap>
```

```
<select id="selectCommentByPrimaryKey" parameterType="long" resultMap="baseResultMap">
        SELECT
                comment_no,
                user_id,
                comment_content,
                reg_date
        FROM comment2
        WHERE comment_no = #{commentNo}
</select>
```

sql 엘리먼트

- sql 엘리먼트
  - 각각의 매핑 구문에서 공통으로 사용할 수 있는 SQL 문자열의 일부를 정의하고 재사용하기 위해 사용함
  - 별도로 빼둔 SQL의 일부는 각각의 매핑 구문에서 include 엘리먼트를 사용해서 처리
  - sql 엘리먼트에는 정적인 내용뿐 아니라 동적 SQL도 넣을 수 있다

```
<sql id="BaseColumns">
        comment_no AS commentNo,
        user_id AS userId,
        comment_content AS commentContent,
        reg_date AS regDate
</sql>

<select id="selectCommentByPrimaryKey" parameterType="long"
                        resultType="ldg.mybatis.model.Comment">
        SELECT
        <include refid="BaseColumns"/>
        FROM comment2
        WHERE comment_no = #{commentNo}
</select>
```

insert, update, delete 엘리먼트

- insert, update, delete 엘리먼트
  - SQL에서 각각 입력, 수정, 삭제를 위해 사용하는 엘리먼트

```
<?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="ldg.mybatis.repository.mapper.CommentMapper">
        <insert id="insertComment" parameterType="ldg.mybatis.model.Comment">
                INSERT INTO comment2(comment_no, user_id, comment_content, reg_date)
                VALUES (comment2_seq.nextval, #{userId}, #{commentContent}, #{regDate})
        </insert>

        <update id="updateComment" parameterType="ldg.mybatis.model.Comment">
                UPDATE comment2 SET comment_content = #{commentContent}
                WHERE comment_no = #{commentNo};
        </update>

        <delete id="deleteComment" parameterType="long">
                DELETE FROM comment2
                WHERE comment_no = #{commentNo};
        </delete>
</mapper>
```

insert, update, delete 엘리먼트

- insert, update, delete 엘리먼트가 사용하는 속성

속성	설명
id	매핑 구문을 구분하는 아이디다. id 속성에 지정한 값은 SqlSession 객체의 구문 아이디 파라미터로 사용한다. 같은 네임스페이스 내에서는 유일해야 한다.
parameterType	파라미터 객체의 타입이다. 원시 타입과 자바의 래퍼 객체에 대해서는 이미 정의된 별칭이 있으므로 별칭을 사용하면 되고, 개발자가 별도로 정의한 객체를 설정할 수도 있다. 개발자가 정의한 객체는 패키지를 포함한 전체 클래스명을 적어주거나 타입 별칭을 사용할 수 있다.
flushCache	매핑 구문을 실행할 때 캐시를 지울지 여부를 설정한다. boolean(true, false) 타입의 값으로 지정해야 한다.
timeout	SQL을 실행한 후 응답을 기다리는 최대 시간이다. 대개는 설정하지 않고 JDBC 드라이버 자체의 타임아웃 값을 그대로 사용한다.
statementType	JDBC의 구문 타입을 지정한다. STATEMENT, PREPARED, CALLABLE 중 하나를 선택할 수 있다. 디폴트는 PREPARED다.

insert, update, delete 엘리먼트

- insert 엘리먼트만 추가로 사용하는 속성
  - 대부분 생성키(관계형 데이터베이스의 자동 증가 필드)에 관련된 속성들

속성	설명
useGeneratedKeys	생성 키 값을 만들기 위해 JDBC의 getGeneratedKeys 메소드를 호출할지 여부를 설정한다. 디폴트 값은 false다.
keyProperty	JDBC의 getGeneratedKeys 메소드가 반환한 값이나 insert 구문의 하위 엘리먼트인 selectKey 엘리먼트에 의해 반환된 값을 설정할 프로퍼티를 지정한다. 디폴트는 설정하지 않는 것이다.
keyColumn	생성 키를 가진 테이블의 칼럼명을 설정한다. PostgreSQL처럼 키 칼럼이 테이블이 첫 번째 칼럼이 아닌 데이터베이스에서만 필요하다.

selectKey 엘리먼트

- selectKey 엘리먼트
  - 자동 생성 키의 값을 가져오기 위해 사용함
  - 자동 생성 키는 MySql의 auto_increment 속성과 sql 서버의 indentity 속성을 사용해 데이터를 입력할 때마다 증가된 값을 설정할 수 있다
  - 방금 입력한 자동 생성 키가 무슨 값인지 입력과 동시에 알아내기 위한 기능이 selectKey의 기능
  - 아이바티스와 달리 마이바티스에서 자동 생성 키는 파라미터로 받은 객체에 설정하는 형태로 처리함

```
<insert id="insertBoard" parameterType="boardVO">
        <selectKey resultType="int" keyProperty="no" order="BEFORE">
                select board_seq.nextval as no from dual
        </selectKey>

        insert into board (no, name, pwd, title, email, content)
        values(#{no},#{name},#{pwd},#{title},#{email}, #{content})
</insert>
```

selectKey 엘리먼트

속성	설명
keyProperty	selectKey 구문의 결과를 설정할 대상 프로퍼티다.
resultType	결과 타입을 지정한다.
order	생성 키를 가져오는 시점을 결정한다. BEFORE 또는 AFTER를 설정할 수 있는데, BEFORE로 설정하면 키를 먼저 조회하고 그 값을 keyProperty에 설정한 후 insert 구문을 실행한다. AFTER로 설정하면 insert 구문을 실행한 후 selectKey 구문을 실행한다.
statementType	마이바티스는 Statement, PreparedStatement, CallableStatement를 매핑하기 위해 STATEMENT, PREPARED, CALLABLE 구문 타입을 지원한다. Statement는 정적인 SQL을 실행하는 구문 객체다. PreparedStatement는 Statement와 비슷하기는 하지만 SQL 파싱과 같은 일부 단계를 거친 정보가 저장돼 있어 재사용 시 SQL을 실행함에 있어 몇 가지 단계가 생략되기 때문에 Statement에 비해 좀더 빠르다. 대개는 Statement보다는 PreparedStatement를 주로 사용한다. CallableStatement는 저장 프로시저를 실행하기 위해 제공하는 구문 객체다.

select 엘리먼트

```
<select id="selectCommentByPrimaryKey" parameterType="long"  resultType="Comment">
        SELECT
                comment_no AS commentNo,
                user_id AS userId,
                comment_content AS commentContent,
                reg_date AS regDate
        FROM comment2
        WHERE comment_no = #{commentNo}
</select>
```

속성	설명
id	매핑 구문을 구분하는 아이디다. id 속성에 지정한 값은 SqlSession 객체의 구문 아이디 파라미터로 사용한다. 같은 네임스페이스 내에서는 유일해야 한다.
parameterType	파라미터 객체의 타입이다. 원시 타입과 자바의 래퍼 객체에 대해서는 이미 정의된 별칭이 있으므로 별칭을 사용하면 되고, 개발자가 정의한 객체를 설정할 수도 있다. 개발자가 정의한 객체의 경우 패키지를 포함한 전체 클래스명을 적어줘야 한다.
resultType	매핑 구문의 결과 타입이다. 타입 별칭을 사용하지 않는다면 패키지 경로까지 모두 명시해야 한다. 결과 형태가 collection이면 collection을 구성하는 타입을 명시해야 한다.
resultMap	결과 매핑은 4.4절의 결과 매핑에서 주로 다뤘다.

속성	설명
flushCache	구문을 호출할 때마다 캐시를 지울지 여부를 설정한다. boolean(true, false) 타입의 값으로 지정해야 한다.
useCache	이 값을 true로 설정하면 구문의 결과를 캐시한다. 디폴트는 true다.
timeout	데이터베이스에 구문을 실행하고 응답을 기다리는 최대 시간이다. 대개는 설정하지 않고 JDBC 드라이버 자체의 타임아웃 값을 그대로 사용한다.
fetchSize	지정된 수만큼의 결과를 반환하게 하는 드라이버 힌트 형태의 값이다. 디폴트는 설정하지 않는 것이고 일부 드라이버는 지원하지 않는다.
statementType	JDBC의 구문 타입을 지정한다. STATEMENT, PREPARED, CALLABLE 중 하나를 선택할 수 있다. 디폴트는 PREPARED다.
resultSetType	데이터베이스 조회 결과의 데이터들을 하나씩 사용할 때 커서를 사용한다. JDBC를 사용하다 보면 ResultSet 객체에서 next 메소드를 사용해서 다음 데이터의 존재 여부를 판단하거나 다음 데이터를 가져오는데, 이 메소드는 커서를 다음 데이터의 위치로 이동시키는 역할을 한다. 즉, 데이터베이스에서 커서는 데이터의 위치를 가리키는 수단이라고 볼 수 있는데, resultSetType 속성에서 지정하는 값은 이 커서의 이동을 어떻게 할지 규칙을 정하는 것이다. 속성으로 설정할 수 있는 값은 FORWARD_ONLY|SCROLL_SENSITIVE|SCROLL_INSENSITIVE 중 하나를 선택한다. FORWARD_ONLY는 커서가 앞으로만 이동한다. SCROLL_SENSITIVE는 커서가 앞뒤로 이동할 수 있고, ResultSet 객체 생성 후 추가 및 삭제된 데이터도 볼 수 있다. 마지막으로 SCROLL_INSENSITIVE는 앞뒤로 이동할 수 있고, ResultSet 객체 생성 후 추가, 삭제된 데이터는 볼 수 없다. 디폴트는 설정하지 않는 것이고, 일부 드라이버는 지원하지 않는다.
databaseId	마이바티스는 데이터베이스 벤더에 따른 다양한 구문을 실행하게 지원한다. 특정 데이터베이스별로 구문을 선택해서 사용하고자 할 때 사용하면 된다. 먼저 마이바티스 설정의 databaseIdProvider를 설정해야 하며, databaseId 속성이 없다면 모두 사용한다.

SqlSession API

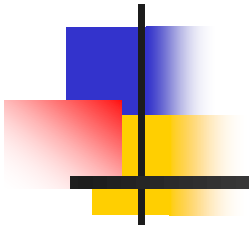
반환 타입	메소드 시그니처	설명
int	delete(String statement)	데이터를 삭제하는 매핑 구문을 호출한다. 단, 파라미터가 없기 때문에 조건을 만들지 않는다.
int	delete(String statement, Object parameter)	데이터를 삭제하는 매핑 구문을 호출한다. 삭제할 데이터를 한정하기 위해 파라미터를 사용해서 조건을 만든다.
int	insert(String statement)	데이터를 입력하는 매핑 구문을 호출한다.
int	insert(String statement, Object parameter)	데이터를 입력하는 매핑 구문을 호출한다. 파라미터에 설정한 값을 사용해서 데이터를 입력한다.
void	select(String statement, Object parameter, ResultHandler handler)	데이터를 조회하는 매핑 구문을 호출한다. 파라미터를 사용해서 조회 조건을 만들 수 있고, 핸들러를 사용해서 조회 결과에 대해 처리를 추가할 수 있다.
void	select(String statement, Object parameter, RowBounds rowBounds, ResultHandler handler)	데이터를 조회하는 매핑 구문을 호출한다. 파라미터를 사용해서 조회 조건을 만들 수 있고, 로우바운드를 사용해서 데이터 개수도 한정할 수 있다. 핸들러를 사용해서 조회 결과에 대해 처리를 추가할 수 있다.
void	select(String statement, ResultHandler handler)	데이터를 조회하는 매핑 구문을 호출한다. 핸들러를 사용해서 조회 결과에 대해 처리를 추가할 수 있다.
List	selectList(String statement)	데이터를 조회하는 매핑 구문을 호출해서 List 타입으로 반환한다.

반환 타입	메소드 시그니처	설명
List	selectList(String statement, Object parameter)	데이터를 조회하는 매핑 구문을 호출해서 List 타입으로 반환한다. 파라미터를 사용해서 조회 조건을 만들 수 있다.
List	selectList(String statement, Object parameter, RowBounds rowBounds)	데이터를 조회하는 매핑 구문을 호출해서 List 타입으로 반환한다. 파라미터를 사용해서 조회 조건을 만들 수 있고, 로우바운드를 사용해서 데이터 개수도 한정할 수 있다.
Map	selectMap(String statement, Object parameter, String mapKey)	데이터를 조회하는 매핑 구문을 호출해서 Map 타입으로 반환한다. 파라미터를 사용해서 조회 조건을 만들 수 있고 mapKey 파라미터를 사용해서 Map의 키에 해당하는 프로퍼티를 정할 수 있다.
Map	selectMap(String statement, Object parameter, String mapKey, RowBounds rowBounds)	데이터를 조회하는 매핑 구문을 호출해서 List 타입으로 반환한다. 파라미터를 사용해서 조회 조건을 만들 수 있고, mapKey 파라미터를 사용해서 Map의 키에 해당하는 프로퍼티를 정할 수 있다. 로우바운드를 사용해서 데이터 개수도 한정할 수 있다.
Map	selectMap(String statement, String mapKey)	데이터를 조회하는 매핑 구문을 호출해서 Map 타입으로 반환한다. mapKey 파라미터를 사용해서 Map의 키에 해당하는 프로퍼티를 정할 수 있다.

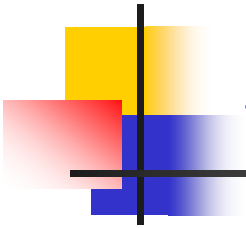
Object	selectOne(String statement)	데이터를 조회하는 매핑 구문을 호출해서 객체로 반환한다. 조회 조건이 없고 데이터 개수가 2개 이상이면 예외를 발생시킨다.
Object	selectOne(String statement, Object parameter)	데이터를 조회하는 매핑 구문을 호출해서 객체로 반환한다. 파라미터를 사용해서 조회 조건을 만들고, 데이터가 개수가 2개 이상이면 예외를 발생시킨다.
int	update(String statement)	데이터를 수정하는 매핑 구문을 호출한다. 조회 조건이 없다.
int	update(String statement, Object parameter)	데이터를 수정하는 매핑 구문을 호출한다. 조회 조건이 있다.

• 입력, 수정, 삭제에 대해서 각기 한 가지 메서드를 제공하고, 조회를 위해 4가지 메서드를 제공

• 조회의 경우에는 반환타입에 따라 메서드가 달리 제공됨
-조회 결과의 개수가 여러 개여서 반환타입이 List 일때는 selectList
- 조회 결과가 한 개일때는 selectOne을 사용함

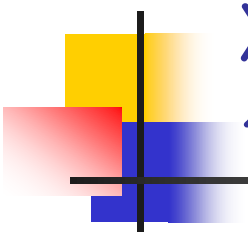


동적 SQL



동적 SQL

- 데이터를 다루다 보면 여러 가지 분기 처리에 의해 SQL을 동적으로 만드는 작업은 빈번하다
- 마이바티스는 동적 SQL을 처리하기 위해
  - XML에서 동적 SQL을 위한 엘리먼트를 사용해 생성



XML에서 동적 SQL을 위한 엘리먼트를 사용해 생성

- 동적 SQL을 만들기 위해 마이바티스가 제공하는 XML 엘리먼트
  - [1] if
  - [2] choose(when, otherwise)
  - [3] trim(where, set)
  - [4] foreach
- 마이바티스는 XML 엘리먼트를 줄이고, 다양한 조건을 처리하기 위해 OGNL 표현식을 사용한다.
  - jsp에서 사용하는 JSTL의 표현식이 OGNL이기 때문에 그대로 마이바티스의 조건문에 적용하면 됨

OGNL의 기본 문법

- OGNL(Object Graph Navigation Language)
  - 프로퍼티의 값을 가져오거나 설정하기 위한 언어
  - OGNL를 사용했던 곳은 오픈 심포니의 웹워크와 JSTL 등이 있고, JSTL의 core라이브러리에서 if 태그의 조건문을 표현할 때 가장 많이 사용함

```
<c:if test="${obj.val != null}">
```

  - OGNL의 기본 문법

문법 타입	예
comment 객체의 userid 필드	comment.userid
현재 객체의 hashcode() 메서드 호출	hashCode()
댓글 배열(comments)의 첫번째 인덱스 값	comments[0]

• 첫번째 문법인 특정 객체의 특정 필드 값을 가져오거나 설정하는 문법을 가장 많이 사용함
• OGNL의 기본 문법만 사용하면 대부분의 동적 SQL 처리가 가능함

if 엘리먼트

- if 엘리먼트

```
<select id="selectCommentByCondition" parameterType="hashmap" resultType="Comment">
        SELECT
                comment_no,
                user_id,
                comment_content,
                reg_date
        FROM comment2
        <if test="commentNo != null">
                WHERE comment_no = #{commentNo}
        </if>
</select>
```

Hashmap 객체에 commentNo 키의 값이 있다면 commentNo 키의 값에 해당하는 레코드를 가져오지만, commentNo 키의 값이 없다면 조회 조건이 추가되지 않기 때문에 모든 댓글의 목록을 가져옴

자바코드로 표현 => if(hashmap.get("commentNo") !=null)

if 엘리먼트

```
HashMap<String,Object> condition
   = new HashMap<String,Object>();
condition.put("commentNo", 10);
```

- HashMap 타입의 파라미터 객체에 다음과 같이 작성자 정보를 추가

```
User user = new User();
user.setUserId("fromm0");
condition.put("user", user);
```

```
<select id="selectCommentByConditionIf" parameterType="hashmap" resultType="Comment">
            SELECT     comment_no,  user_id,  comment_content,  reg_date
            FROM comment2
            <if test="commentNo != null">
                        where  comment_no = #{commentNo}
            </if>
            <if test="user != null and user.userId != null">
                        where user_id = #{user.userId}
            </if>
</select>
```

• 댓글 번호와 작성자 정보의 유무로 조회 조건이 결정됨
• HashMap 객체에 작성자 정보가 있고 작성자 아이디가 null 이 아니면 해당 작성자의 댓글만 가져오는 SQL

```
자바코드로 표현 =>   if(hashmap.get("user") !=null &&
                                    ((User)hashmap.get("user")).getUserId() != null)
```

choose(when, otherwise) 엘리먼트

```
<select id="selectCommentByConditionChoose" parameterType="hashmap" resultType="Comment">
        SELECT    comment_no,  user_id,  comment_content,   reg_date
        FROM comment2
        <choose>
                <when test="commentNo != null">
                        WHERE comment_no = #{commentNo}
                </when>
                <when test="user != null and user.userId != null">
                        WHERE user_id = #{user.userId}
                </when>
                <otherwise>
                        WHERE comment_no = 1
                        AND user_id = 'fromm0'
                </otherwise>
        </choose>
</select>
```

- 댓글 번호가 있다면 해당되는 댓글만 가져오는 동적 SQL
- 작성자 정보가 있고 사용자 아이디가 있다면 해당되는 작성자의 댓글만 가져오는 동적 SQL
- 위 두 가지 조건 중 하나도 만족하지 못하면 댓글번호는 1 이고, 작성자 아이디는 fromm0 인 댓글만 가져오는 동적 SQL

trim(where) 엘리먼트

trim 엘리먼트는 if 엘리먼트의 단점을 보완할 수있는 기능을 제공함
if 엘리먼트를 사용한 2번째 예제에서 댓글번호와 작성자 아이디가 모두 있을 때의 결과를 보자

```
SELECT    comment_no,  user_id,  comment_content,  reg_date
          FROM comment2
          where  comment_no = #{commentNo}
          where user_id = #{user.userId}
```

where 가 두번 나오는 에러를 피하기 위해
where 엘리먼트를 사용할 수 있다

```
<select id="selectCommentByConditionIf" parameterType="hashmap" resultType="Comment">
          SELECT    comment_no,  user_id,  comment_content,    reg_date
                    FROM comment2
          <where>
                    <if test="commentNo != null">
                              comment_no = #{commentNo}
                    </if>
                    <if test="user != null and user.userId != null">
                              AND user_id = #{user.userId}
                    </if>
          </where>
</select>
```

```
SELECT comment_no, user_id, comment_content, reg_date
          FROM comment2
          WHERE user_id = #{user.userId}
```

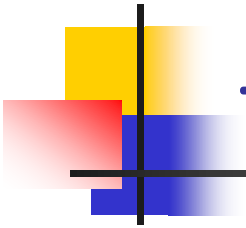
• where 엘리먼트를 사용하면 하위 엘리먼트에서 생성한 내용이 있을 경우 앞에 where를 붙여주고, 생성한 내용이 없으면 그대로 무시함
• 하위 엘리먼트에서 생성한 내용이 AND 나 OR로 시작할 경우 자동으로 이 단어들을 지워줌

trim(where) 엘리먼트

- 조회 조건에 붙는 내용이 AND 나 OR가 아니라 다른 문자로 시작한다고 할때는 where 엘리먼트가 처리하는 기능에 더해 추가로 규칙을 정의하기 위해 trim 엘리먼트를 제공함

```
<select id="selectCommentByConditionTrim" parameterType="hashmap" resultType="Comment">
        SELECT    comment_no, user_id, comment_content, reg_date
        FROM comment2
        <trim prefix="WHERE" prefixOverrides="AND |OR ">
                <if test="commentNo != null">
                        AND comment_no = #{commentNo}
                </if>
                <if test="user != null and user.userId != null">
                        AND user_id = #{user.userId}
                </if>
        </trim>
</select>
```

• trim - 하위 엘리먼트가 내용을 만들면 prefix 속성에 설정한 문자인 WHERE 를 붙이고, 하위 엘리먼트가 생성한 내용이 AND 나 OR로 시작하면 prefixOverrides 속성에 설정한 AND 나 OR를 자동으로 지워줌



trim(where) 엘리먼트

- trim 엘리먼트가 제공하는 속성
  - [1] prefix
    - 처리 후 엘리먼트의 내용이 있으면 가장 앞에 붙여준다
  - [2] prefixOverrides
    - 처리 후 엘리먼트 내용 중 가장 앞에 해당 문자들이 있다면 자동으로 지워준다
  - [3] suffix
    - 처리 후 엘리먼트 내에 내용이 있으면 가장 뒤에 붙여준다
  - [4] suffixOverrides
    - 처리 후 엘리먼트 내용 중 가장 뒤에 해당 문자들이 있다면 자동으로 지워준다
- trim 엘리먼트는 select 엘리먼트 뿐만 아니라 insert, update, delete 엘리먼트에도 사용 가능

trim(where) 엘리먼트

```
<update id="updateCommentIf" parameterType="Comment">
        UPDATE comment2
            set
                <if test="commentContent != null">comment_content = #{commentContent},</if>
                <if test="regDate != null">reg_date = #{regDate}</if>

        WHERE comment_no = #{commentNo};
</update>
```

```
UPDATE comment2  set
        comment_content = #{commentContent},
WHERE comment_no = #{commentNo};
```

- regdate 가 없는 경우 - where 앞에 ,(쉼표)가 붙기 때문에 SQL 문법 에러 발생
- update 엘리먼트에서 분기 처리를 할 때도 trim 엘리먼트를 사용하면 쉽게 해결할 수 있다

```
<update id="updateCommentTrim" parameterType="Comment">
        UPDATE comment2
        <trim prefix="SET" suffixOverrides=",">
                <if test="commentContent != null">comment_content = #{commentContent},</if>
                <if test="regDate != null">reg_date = #{regDate}</if>
        </trim>
        WHERE comment_no = #{commentNo};
</update>
```

foreach 엘리먼트

```
long[] cmtNos={1L,2L,3L};

HashMap<String,Object> hmap
   = new HashMap<String,Object>();
hmap.put("commentNos",  cmtNos);
```

- 여러 개의 값으로 구성된 파라미터를 설정하기 위해 foreach 엘리먼트를 사용
- SQL의 조회 조건에는 in 절로 조건을 추가하는 경우가 있다. in 절에는 대부분 어떠한 값들을 전달하게 되는데, 다음과 같이 값을 반복해서 설정한다

```
<select id="selectCommentByConditionForeach" parameterType="hashmap" resultType="Comment">
        SELECT comment_no,  user_id, comment_content, reg_date
        FROM comment2
        <trim prefix="WHERE" prefixOverrides="AND |OR ">
                <if test="commentNos != null">
                        comment_no   IN
                        <foreach collection="commentNos" item="commentNo"
                                index="index" open="(" close=")" separator=",">
                        #{commentNo}
                        </foreach>
                </if>
        </trim>
</select>
```

• 매핑 구문에 댓글 번호 목록인 commentNos 파라미터에 {1L, 2L, 3L} 을 설정할 때 생성되는 SQL

```
SELECT      comment_no, user_id, comment_content, reg_date
FROM comment2
WHERE comment_no IN (1, 2, 3)
```

foreach 엘리먼트

• foreach 엘리먼트에서는 댓글 번호 목록 각각을 ,(쉼표) 구분자로 합친다. 그리고 마지막에 합친 값의 앞뒤로 (와)를 붙이는 것이다

- foreach 엘리먼트의 속성
  - collection – 값 목록을 가진 객체를 설정하면 됨.
    - 파라미터의 타입은 배열이나 List 모두 처리 가능
  - item – 목록에서 각각의 값을 사용하고자 할 때 사용하는 속성
  - index – 몇 번째 값인지 나타내는 인덱스 값. 0 부터 시작
  - open – 목록에서 값을 가져와서 설정할 때 가장 앞에 붙여주는 문자를 지정한다. in 절에서는 대부분 (로 시작함
  - close – 목록에서 값을 가져와서 설정할 때 가장 뒤에 붙여주는 문자를 지정한다. in 절에서는 대부분) 로 끝남
  - separator – 목록에서 값을 가져와서 설정할 때 값들 사이에 붙여주는 문자를 지정한다. in 절에서는 값 사이에 쉼표를 붙여준다

set 엘리먼트

- if, choose, trim, foreach 엘리먼트는 대개 where 조건에 많이 사용함
- set 엘리먼트는 where 조건이 아닌 update 에서 마지막으로 명시된 컬럼 표기에서 쉼표를 제거하는 역할을 담당함

```
<update id="updateCommentIf" parameterType="Comment">
        UPDATE comment2
                <set>
                <if test="commentContent != null">comment_content = #{commentContent},</if>
                <if test="regDate != null">reg_date = #{regDate},</if>
                </set>
        WHERE comment_no = #{commentNo};
</update>
```

- set엘리먼트가 마지막의 쉼표를 자동으로 제거해 줌(regdate 뒤 쉼표 제거)
- set엘리먼트는 trim엘리먼트로도 대체가 가능

예제 – 동적 SQL

pd.xml

```
<mapper namespace="com.mybatis.mapper.pd">
    <sql id="colList">
        no, pdname, price, regdate
    </sql>

   <select id="selectAll" resultType="PdDTO"      parameterType="PdDTO">
        select
        <include refid="colList"/>
        from pd
        <if test="pdName!=null and pdName!=''">
                where pdName like '%' || #{pdName} || '%'
        </if>
        order by no desc
    </select>

.....
```

pdList.jsp

```
……
</table>
    <br>
    <a href="<c:url value='/pd/pdWrite.do' />" >
    상품 등록</a>
    <br><br>
    <form action="<c:url value='/pd/pdList.do' />"      method="post" name="frm1">
        검색어:<input type="text" name="pdName">
        <input type="submit" value="검색">
    </form>
</body>
</html>
```

PdListController

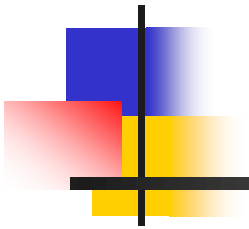
```
@RequestMapping("/pd/pdList.do")
    public ModelAndView listPd(PdDTO pdDto){
        //전체 상품 조회
        //1. 파라미터
        System.out.println("PdListController:listPd(), "+ "pdName="+pdDto);

        //2. db작업 - select
        List<PdDTO> alist = pdService.selectAll(pdDto);
        System.out.println("상품전체조회 결과:alist.size()="+alist.size());

        //3. 결과 뷰페이지 저장,리턴
        ModelAndView mav = new ModelAndView();
        mav.addObject("pdList", alist);
        mav.setViewName("pd/pdList");

        return mav;
    }
```

PdDAO

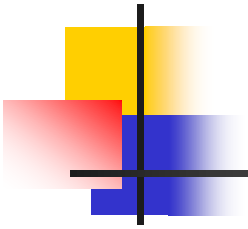
```
public List<PdDTO> selectAll(PdDTO pdDto){
        //db에서 pd테이블의 전체 레코드를 조회해서 화면에 출력하기
        SqlSession sqlSession
                = getSqlSessionFactory().openSession();
        try{
                List<PdDTO> alist
                        = sqlSession.selectList(namespace
                                + ".selectAll", pdDto);
                return alist;
        }finally{
                sqlSession.close();
        }
}
```

- PdService

```
public List<PdDTO> selectAll(PdDTO pdDto){
        return pdDao.selectAll(pdDto);
}
```

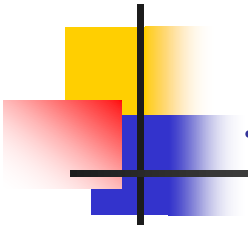


마이바티스와 스프링 웹 애플리케이션 연동



마이바티스 라이브러리

- 마이바티스와 스프링을 연동하는 모듈을 마이바티스에서 별도로 제공하기 때문에 추가로 연동 모듈을 다운로드 해야 함
- http://code.google.com/p/mybatis/
- https://github.com/mybatis/spring/releases
  - mybatis-spring-1.3.1.zip
  - mybatis-spring-1.3.1.jar
    - 마이바티스 스프링 연동 모듈 API를 가진 jar 파일
    - 스프링과 연동해서 스프링 빈을 생성하기 위해 필요함



스프링의 데이터베이스 관련 설정

- 마이바티스와 스프링을 함께 사용할 때는 마이바티스의 데이터베이스 설정 일부는 생략하고 스프링에 그 설정을 추가한다.
- 스프링을 사용하면 스프링 설정이 마이바티스 설정의 일부를 대체하는 셈이다

dispatcher-servlet.xml

```
<?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"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.1.xsd
        http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-
    tx-3.1.xsd
        http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.1.xsd">
    <!-- 데이터소스 -->
    <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
        <property name="driverClassName" value="com.mysql.jdbc.Driver" />
        <property name="url" value="jdbc:mysql://localhost:3306/mybatis_example" />
        <property name="username" value="mybatis" />
        <property name="password" value="mybatis" />
        <property name="maxActive" value="20" />
        <property name="maxWait" value="6000" />
        <property name="poolPreparedStatements" value="true" />
        <property name="defaultAutoCommit" value="true" />
        <property name="initialSize" value="10" />
        <property name="maxIdle" value="20" />
        <property name="validationQuery" value="select 1" />
        <property name="testWhileIdle" value="true" />
        <property name="timeBetweenEvictionRunsMillis" value="7200000" />
    </bean>
```

```
<!-- 트랜잭션 관리자 -->
<bean id="transactionManager"
        class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
        <property name="dataSource" ref="dataSource" />
</bean>

<!-- Annotation 을 사용한 트랜잭션 사용시 활성화 -->
<tx:annotation-driven transaction-manager="transactionManager" />

<!-- @Service, @Repository 애노테이션이  붙은 클래스를 빈으로 등록 -->
<context:component-scan base-package="ldg.mybatis">
        <context:exclude-filter type="annotation"
                expression="org.springframework.stereotype.Controller" />
</context:component-scan>
```

```
    <!-- 마이바티스 설정 // -->
    <bean id="sqlSessionFactoryBean" class="org.mybatis.spring.SqlSessionFactoryBean">
        <property name="typeAliasesPackage" value="ldg.mybatis.model" />
        <property name="dataSource" ref="dataSource" />
        <property name="configLocation" value="classpath:/mybatis-config.xml" />
        <property name="mapperLocations">
            <array>
                <value>classpath*:/ldg/mybatis/repository/mapper/**/*.xml</value>
            </array>
        </property>
    </bean>

    <bean id="sqlSessionTemplate" class="org.mybatis.spring.SqlSessionTemplate">
        <constructor-arg name="sqlSessionFactory" ref="sqlSessionFactoryBean" />
    </bean>

    <bean id="mapperScannerConfigurer" class="org.mybatis.spring.mapper.MapperScannerConfigurer">
        <property name="basePackage" value="ldg.mybatis.repository" />
    </bean>
    <!-- // mybatis 설정 -->

</beans>
```

스프링의 데이터베이스 관련 설정

```
<bean id="dataSource"
        class="org.apache.commons.dbcp.BasicDataSource">
```

- dataSource
  - dataSource id를 가진 bean은 데이터베이스 연결 정보를 가진 객체
  - 마이바티스와 스프링을 연동하면 **데이터베이스 설정과 트랜잭션 처리**는 스프링에서 관리한다.
  - 스프링과 연동하면 마이바티스 설정에는 더 이상 데이터베이스 연결 정보에 대한 설정이 필요 없다.
  - 데이터베이스 Connection Pool 을 위해 아파치 DBCP 프로젝트 구현체를 사용하도록 설정했다
  - 데이터 소스를 담당하는 dataSource 빈은 자바 코드에서 직접 사용하기 보다는 트랜잭션을 관리하는 빈에서 사용한다

스프링의 데이터베이스 관련 설정

- transactionManager
  - transactionManager id를 가진 bean은 트랜잭션을 관리하는 객체
  - 마이바티스는 다른 ORM 제품과 달리 JDBC를 그대로 사용하기 때문에 DataSourceTransactionManager 타입의 빈을 사용한다.
  - 스프링에는 이외에도 하이버네이트나 JDO, JMS, JPA 등을 위한 다양한 트랜잭션 관리자를 제공한다.
  - property 엘리먼트를 사용해서 먼저 정의했던 데이터 소스 빈을 사용한다.

```
<bean id="transactionManager"
   class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
            <property name="dataSource" ref="dataSource" />
</bean>
```

스프링의 데이터베이스 관련 설정

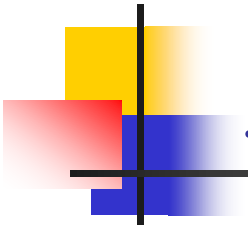
- tx:annotation-driven
  - 스프링이 제공하는 트랜잭션 관리 방법이 몇 가지 있다
    - 직접 코드로 처리할 수도 있고
    - AOP 를 통해 처리할 수도 있으며
    - 애노테이션을 사용해서도 처리가 가능함
  - 여기서는 애노테이션으로 처리하는 방식을 선택
    - 트랜잭션을 적용하고자 하는 메서드나 클래스에 @Transactional 애노테이션을 적어주기만 하면 됨

```
<tx:annotation-driven transaction-manager="transactionManager" />
```

스프링의 데이터베이스 관련 설정

- context:component-scan
  - 스프링 빈을 매번 XML에 설정하지 않고 자동으로 검색해서 스프링 빈으로 등록할 수 있다
  - 스프링 빈으로 등록되는 대상은 클래스명 위에 @Component, @Repository, @Service, @Controller 애노테이션을 선언한 클래스가 대상이 됨
  - base-package 속성에 자동 검색할 클래스들이 모여 있는 패키지를 지정해주면 됨

```
<context:component-scan base-package="ldg.mybatis">
    <context:exclude-filter type="annotation" expression="org.springframework.stereotype.Controller" />
</context:component-scan>
```



스프링 연동 설정

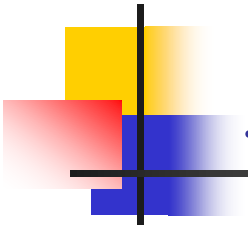
- 마이바티스와 스프링을 함께 사용할 때 사용하는 클래스
  - org.mybatis.spring.SqlSessionFactoryBean
  - org.mybatis.spring.SqlSessionTemplate
    - 스프링 연동 모듈에 포함돼 있다
    - 마이바티스와 스프링을 연동하기 위해서는 반드시 사용해야 함

```
<!-- 마이바티스 설정 // -->
<bean id="sqlSessionFactoryBean" class="org.mybatis.spring.SqlSessionFactoryBean">
        <property name="typeAliasesPackage" value="ldg.mybatis.model" />
        <property name="dataSource" ref="dataSource" />
        <property name="configLocation" value="classpath:/mybatis-config.xml" />
        <property name="mapperLocations">
                <array>
                        <value>classpath*:/ldg/mybatis/repository/mapper/**/*.xml</value>
                </array>
        </property>
</bean>

<bean id="sqlSessionTemplate" class="org.mybatis.spring.SqlSessionTemplate">
        <constructor-arg name="sqlSessionFactory" ref="sqlSessionFactoryBean" />
</bean>

<bean id="mapperScannerConfigurer" class="org.mybatis.spring.mapper.MapperScannerConfigurer">
        <property name="basePackage" value="ldg.mybatis.repository" />
</bean>
<!-- // mybatis 설정 -->
```

마이바티스 설정 파일의 위치 지정

스프링 연동 설정

- sqlSessionFactoryBean
  - SqlSessionFactory를 생성하기 위한 FactoryBean 설정이다
  - 이 빈을 사용해서 스프링은 SqlSessionFactory객체를 한 번만 생성한다
  - 마이바티스를 사용할 때마다 SqlSessionFactory를 이용해서 마이바티스 객체를 매번 생성한다
- sqlSessionTemplate
  - sqlSessionTemplate은 마이바티스의 SqlSession과 같은 역할을 담당하지만 트랜잭션을 처리하는 방법에서 약간의 차이점이 있다.
  - 마이바티스의 SqlSession은 트랜잭션 처리를 위해 commit/rollback 메서드를 명시적으로 호출해야 하지만,
  - SqlSessionTemplate은 스프링이 트랜잭션을 대신 처리하게 구조화돼 있기 때문에 commit/rollback 메서드를 명시적으로 호출할 수 없다.
  - SqlSession 의 메서드에서 던지는 예외타입이 PersistenceException 이지만, SqlSessionTemplate은 스프링의 DataAccessException을 던진다



스프링 연동 설정

- mapperScannerConfigurer
  - 매퍼 인터페이스를 자동 검색해서 등록한다
  - 매퍼 인터페이스의 패키지 중 가장 상위 패키지를 지정해주면 그 하위에 있는 매퍼 인터페이스를 모두 등록함
  - 세부적으로 하위 패키지별로 지정하고자 할 때는 구분자를 사용해서 여러 개의 패키지를 지정
    - 구분자 => ,(쉼표) ;(세미콜론) 둘 중 하나를 사용

레이어별 예제

Service 클래스가 주로 담당하는 역할은 비즈니스 로직을 처리하거나 트랜잭션을 처리하는 것

```
@Service
public class CommentService {
    @Autowired
    private CommentSessionRepository commentRepository;

    public List<Comment> selectComment(Long commentNo) {
        Map<String, Object> condition = new HashMap<String, Object>();
        condition.put("commentNo", commentNo);

        return commentRepository.selectCommentByCondition(condition);
    }

    @Transactional
    public Integer insertComment(Comment comment) {
        return commentRepository.insertComment(comment);
    }
```

Service

트랜젝션은 반드시 서비스에서

```
    @Transactional
    public Integer deleteComment(Long commentNo) {
            return commentRepository.deleteComment(commentNo);
    }

}
```

Repository

```
@Repository          //abstractRepository를 대체함
public class CommentSessionRepository {
    @Autowired    <<자동으로 서비스가 들어감, 이전에 bean 등록한걸 자동으로 해줌
    private SqlSessionTemplate sqlSession;

    private final String namespace = "ldg.mybatis.repository.mapper.CommentMapper";

    public Comment selectCommentByPrimaryKey(Long commentNo) {
        return (Comment)sqlSession.selectOne(String.format("%s.selectCommentByPrimaryKey", namespace),
    commentNo);
    }

    public List<Comment> selectCommentByCondition(Map<String, Object> condition) {
        return sqlSession.selectList(String.format("%s.selectCommentByCondition", namespace), condition);
    }

    public Integer insertComment(Comment comment) {
        int result = sqlSession.insert(String.format("%s.insertComment", namespace), comment);
        return result;
    }

    public Integer deleteComment(Long commentNo) {
        int result = sqlSession.delete(String.format("%s.deleteComment", namespace), commentNo);
        return result;
    }
}
```

Repository 클래스- DAO의 역할을 담당

@Service, @Repository, @Controller 애노테이션을 선언한 클래스는 자동 빈 검색을 통해 빈으로 자동 등록한다

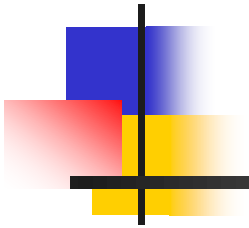
Repository

```
@Autowired
private SqlSessionTemplate   sqlSession;
```

- SqlSessionTemplate
  - SqlSessionTemplate 빈은 스프링과 연동한 마이바티스 객체
  - 마이바티스 기능을 사용하기 위해서는 Repository 클래스에서 SqlSessionTemplate 빈을 반드시 선언하고 사용해야 함
  - SqlSessionTemplate 빈은 마이바티스가 제공하는 메서드를 그대로 제공하기 때문에 마이바티스만 사용할 때와 동일하게 처리

데이터를 출력하는 jsp

```
<%@page
    import="java.io.*,javax.servlet.*,java.util.*,ldg.mybatis.service.*,ldg.mybatis.model.*,org.springf
    ramework.context.*,org.springframework.context.support.*" contentType="text/html;
    charset=utf8"%>
<%
ApplicationContext applicationContext = new
    ClassPathXmlApplicationContext("applicationContext.xml");

Long commentNo = Long.parseLong(request.getParameter("commentNo"));
CommentService commentService =
    (CommentService)applicationContext.getBean("commentService");
List<Comment> result = commentService.selectComment(commentNo);
%>
<% for( Comment comment : result ) { %>
댓글번호 : <%= comment.getCommentNo() %><br>
작성자아이디 : <%= comment.getUserId() %><br>
작성일시 : <%= comment.getRegDate() %><br>
댓글내용 : <%= comment.getCommentContent() %><br>
<% } %>
```

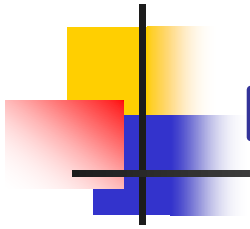


스프링 연동 예제

dispatcher-servlet.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:p="http://www.springframework.org/schema/p"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
     http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
     http://www.springframework.org/schema/tx
     http://www.springframework.org/schema/tx/spring-tx-3.1.xsd
     http://www.springframework.org/schema/context
     http://www.springframework.org/schema/context/spring-context-3.0.xsd">

    <!-- 데이터소스 -->
    <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
            <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver" />
            <property name="url" value="jdbc:oracle:thin:@yang-hp:1521:orcl" />
            <property name="username" value="scott" />
            <property name="password" value="tiger" />
    </bean>

    <!-- 트랜잭션 관리자 -->
    <bean id="transactionManager"
            class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
            <property name="dataSource" ref="dataSource" />
    </bean>
```

```
<!-- 마이바티스 설정 // -->
<bean id="sqlSessionFactoryBean" class="org.mybatis.spring.SqlSessionFactoryBean">
        <property name="typeAliasesPackage" value="com.pd.model" />
        <property name="dataSource" ref="dataSource" />
        <property name="configLocation" value="classpath:/mybatis-config.xml" />
        <property name="mapperLocations">
                <array>
                        <!-- <value>classpath:/com/mybatis/mapper/pd.xml</value> -->
                        <value>classpath*:/com/mybatis/mapper/**/*.xml</value>
                </array>
        </property>
</bean>

<bean id="sqlSessionTemplate" class="org.mybatis.spring.SqlSessionTemplate">
        <constructor-arg name="sqlSessionFactory" ref="sqlSessionFactoryBean" />
</bean>

<bean id="mapperScannerConfigurer" class="org.mybatis.spring.mapper.MapperScannerConfigurer">
        <property name="basePackage" value="com.mybatis.mapper" />
</bean>

<!-- Annotation 을 사용한 트랜잭션 사용시 활성화 -->
<tx:annotation-driven transaction-manager="transactionManager" />

<!-- @Controller, @Service, @Repository애노테이션이 붙은 클래스를 빈으로 등록 -->
<context:component-scan base-package="com.pd“>
</context:component-scan>  ... </beans>
```



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>
      <!-- 마이바티스의 작동 규칙정의 -->
      <settings>
        <setting name="cacheEnabled" value="false"/>
        <setting name="useGeneratedKeys" value="false"/>
        <setting name="mapUnderscoreToCamelCase" value="true"/>
      </settings>

</configuration>
```

pd.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.mybatis.mapper.pd">
	<insert id="pdInsert" parameterType="pdBean">
		<selectKey keyProperty="no" resultType="int" order="BEFORE">
			select pd_seq.nextval  as no from dual
		</selectKey>
		insert into pd(no, pdname, price)
		values(#{no}, #{pdName}, #{price})
	</insert>

	<select id="pdList" resultType="pdBean">
		select * from pd order by no desc
	</select>

	<select id="pdDetail" parameterType="int"
		resultType="pdBean">
		select * from pd where no=#{no}
	</select>

	<update id="pdUpdate" parameterType="pdBean">
		update pd set pdname=#{pdName}, price=#{price}
		 where no=#{no}
	</update>

	<delete id="pdDelete" parameterType="int">
		delete from pd where no=#{no}
	</delete>
</mapper>
```

PdDAO

```
package com.pd.model;
import java.util.List;
import org.mybatis.spring.SqlSessionTemplate;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;

@Repository
public class PdDAO{
	@Autowired
	private SqlSessionTemplate sqlSession;
	private final String namespace = "com.mybatis.mapper.pd";

	public int insertPd(PdBean pdBean){
		//pd 테이블에 insert하는 메서드
		int n = (int) sqlSession.insert(namespace + ".pdInsert", pdBean);
		return n;
	}
	public List<PdBean> selectAll(){
		//전체 글을 조회하는 메서드
		List<PdBean> alist
			= sqlSession.selectList(namespace + ".pdList");

		return alist;
	}
```

PdDAO

```
    public PdBean selectByNo(int no){
            //no에 해당하는 글 조회
            PdBean bean
                = (PdBean) sqlSession.selectOne(namespace + ".pdDetail", no);

            return bean;
    }

    public int updatePd(PdBean bean){
            int n= sqlSession.update(namespace + ".pdUpdate", bean);
            return n;
    }


    public int deletePd(int no){
            //no에 해당하는 상품 삭제
            int n=sqlSession.delete(namespace + ".pdDelete", no);
            return n;
    }
}//class
```

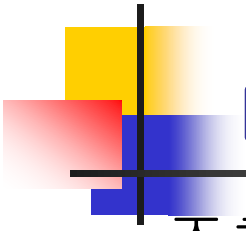
PdService

```
package com.pd.model;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

@Service
public class PdService {
	@Autowired
	private PdDAO pdDao;

	@Transactional
	public int insertPd(PdBean pdBean){
		return pdDao.insertPd(pdBean);
	}

	public int updatePd(PdBean bean){
		return pdDao.updatePd(bean);
	}

	public List<PdBean> selectAll(){
		return pdDao.selectAll();
	}

.....
}
```

```
package com.pd.controller;

@Controller
@RequestMapping("/pd/pdWrite.do")
public class PdWriteController {
	@Autowired
	private PdService pdService;

	@RequestMapping(method=RequestMethod.GET)
	public ModelAndView pdWriteGet(){
		ModelAndView mav = new ModelAndView();
		mav.setViewName("/pd/pdWrite");
		return mav;
	}
	@RequestMapping(method=RequestMethod.POST)
	public ModelAndView pdWritePost(@ModelAttribute("pdBean") PdBean pdBean){
		//1. 파라미터 읽어오기
		System.out.println("pdBean 파라미터 값 : " + pdBean);

		//2. db작업 - insert
		int n = pdService.insertPd(pdBean);
		System.out.println("상품 등록 여부, n="+n);

		//3. 결과, 뷰페이지 저장
		ModelAndView mav = new ModelAndView();

		//pdList.jsp 페이지로 redirect=> /pd/pdList.do로 redirect
		mav.setViewName("redirect:/pd/pdList.do");
		return mav;
	}
}//class
```

```
package com.pd.controller;

@Controller
public class PdListController {
	@Autowired
	private PdService pdService;

	@RequestMapping("/pd/pdList.do")
	public ModelAndView listPd(){
		//1. 파라미터
		System.out.println("PdListController : listPd()호출");

		//2. db작업  - select
		List<PdBean> alist=null;
		alist = pdService.selectAll();
		System.out.println("전체 목록 조회 결과, alist.size()="   + alist.size());

		//3. 결과, 뷰페이지 저장
		ModelAndView mav = new ModelAndView();
		mav.addObject("alist", alist);
		mav.setViewName("pd/pdList");

		return mav;
	}
}//class
```

```
package com.pd.controller;

@Controller
public class PdDetailController {
	@Autowired
	private PdService pdService;

	@RequestMapping("/pd/pdDetail.do")
	public ModelAndView detailPd(@RequestParam(value="no", defaultValue="0"
				/*,required=false*/)	int no){
		//1. 파라미터
		System.out.println("PdDetailController : detailPd()호출" + ", 파라미터 no="+no);

		//2. db작업 - select
		PdBean pdBean=null;
		pdBean = pdService.selectByNo(no);
		System.out.println("상품 상세보기 결과, pdBean="+pdBean);

		//3. 결과, 뷰 페이지 저장
		ModelAndView mav = new ModelAndView();
		mav.addObject("bean", pdBean);
		mav.setViewName("/pd/pdDetail");

		return mav;
	}
}//class
```



mybatis

- 조회 결과를 자바 객체에 설정(결과 매핑)
  - 컬럼명과 일치하지 않는다면 별도로 값을 설정하는 규칙을 정의해야만 정확히 값을 설정할 수 있다
  - 일치하지 않는 경우에 값을 설정해주기 위해 제공하는 XMl 엘리먼트
    - resultMap – 결과 매핑을 사용하기 위한 가장 상위 엘리먼트
      - 결과 매핑을 구분하기 위한 id 속성과 매핑하는 대상 클래스를 정의하는 type 속성을 사용
    - id – 기본 키에 해당되는 값을 설정
    - result – 기본 키가 아닌 나머지 칼럼에 대해 매핑
    - constructor, discriminator
    - association – 1:1 관계를 처리
    - collection – 1:N 관계를 처리
      - 1:N 관계를 나타내는 모델 클래스는 대개 List 타입의 객체 변수를 가짐

예-동적 sql

```
<sql id="searchWhere">
    <where>
        <if test="searchKeyword !=null and searchKeyword !='' ">
                ${searchCondition} like '%'||#{searchKeyword}||'%'
        </if>
    </where>
</sql>

<select id="getBoardList" parameterType="searchBean"    resultType="boardBean">
      SELECT  no, name, pwd, title, email, content, regdate, readcount,
                (sysdate-regdate)*24 as newImgTerm
      FROM board
                <include refid="searchWhere"/>
      ORDER BY no DESC
</select>

<select id="getTotalRecord" parameterType="searchBean"     resultType="Integer">
      select count(*) from board
      <include refid="searchWhere"/>
</select>
```

```
<parameterMap type="map" id="reBoardDeleteParam">
        <parameter property="no" javaType="string"  jdbcType="VARCHAR"  mode="IN"/>
        <parameter property="step" javaType="string"  jdbcType="VARCHAR"  mode="IN"/>
        <parameter property="groupNo" javaType="string"  jdbcType="VARCHAR"  mode="IN"/>
</parameterMap>

<delete id="reBoardDeleteProc" parameterMap="reBoardDeleteParam">
        {call deleteReboard(?,?,?)}
</delete>

<select id="productList"  resultType="productBean"  parameterType="eventProductBean">
        SELECT  *
        FROM  (
        SELECT ROWNUM RNUM, ALL_LIST.*
          FROM  (
                    <choose>
                        <when test="eventName != null and eventName !='' ">
                                select * from productEventView  where eventname=#{eventName}
                        </when>
                        <otherwise>
                                select * from products
                                order by productNo desc
                        </otherwise>
                    </choose>
          ) ALL_LIST
    )
```

```
    <if test="recordCountPerPage != 0 ">
            <![CDATA[
            WHERE  RNUM  > #{firstRecordIndex}
            AND  RNUM <= #{firstRecordIndex} + #{recordCountPerPage} ]]>
    </if>
  </select>

  <select id="getTotalRecord" resultType="Integer"  parameterType="eventProductBean">
       <choose>
           <when test="eventName != null and eventName !='' ">
                    select count(*) from productEventView where eventname=#{eventName}
           </when>
           <otherwise>
                    select count(*) from products
           </otherwise>
      </choose>
  </select>

<!-- 다중 조건 상품 검색 -->
<!-- 예:브랜드 여러 개 선택, 사이즈 여러 개, 색상 여러 개 선택해서 검색하는 경우
select * from products  where company in ('허브나라', '향초사', '오일사')
and  sellprice in(7000, 8000) and mileage in(70, 80, 90); -->
<select id="selectPdByConditionForeach" parameterType="hashmap"  resultType="ProductBean">
         select * from products
         <!-- <trim prefix="WHERE" prefixOverrides="AND |OR "> -->
         <where>
```

```
            <if test="companys != null">
                company IN
                <foreach collection="companys" item="company"
                    index="index" open="(" close=")" separator=",">
                    #{company}
                </foreach>
            </if>
            <if test="sellPrices != null">
                and sellPrice IN
                <foreach collection="sellPrices" item="sellPrice"
                    index="index" open="(" close=")" separator=",">
                    #{sellPrice}
                </foreach>
            </if>
            <if test="mileages != null">
                and mileage IN
                <foreach collection="mileages" item="mileage"
                    index="index" open="(" close=")" separator=",">
                    #{mileage}
                </foreach>
            </if>
        </where>
        <!-- </trim> -->
</select>
```

```
<select id="orderListByOrderDate"  resultType="ordersAllViewBean" parameterType="dateSearchBean">
            SELECT  *
            FROM  (
               SELECT ROWNUM RNUM, ALL_LIST.*
                FROM  (
                          <![CDATA[
                                                                  select * from orders
                                                                  where orderDate>=#{startDay}
                                                                  and orderDate< to_date(#{endDay})+1
                                                                  ]]>
                                                                  <if test="customerId != null and customerId != '' ">
                                                                            and customerId=#{customerId}
                                                                  </if>
                                                                  order by orderno desc
           ) ALL_LIST
       )
       <![CDATA[
       WHERE  RNUM  > #{firstRecordIndex}
       AND  RNUM <= #{firstRecordIndex} + #{recordCountPerPage} ]]>
    </select>
   <select id="getTotalRecord" resultType="Integer"  parameterType="dateSearchBean">
          <![CDATA[
                                        select count(*) from orders
                                        where orderDate>=#{startDay}
                                        and orderDate< to_date(#{endDay})+1
                                        ]]>
                                        <if test="customerId != null and customerId != '' ">
                                                    and customerId=#{customerId}
                                        </if>
    </select>
```

```
<bean id="sqlSessionFactoryBean" class="org.mybatis.spring.SqlSessionFactoryBean">
        <property name="dataSource" ref="dataSource" />
        <property name="configLocation"
                value="classpath:/config/mybatis/${Globals.DbType}/mybatis-config.xml" />
        <property name="typeAliasesPackage" value="com.herb.app" />
        <property name="mapperLocations">
                <array>
                <!-- <value>classpath:/config/mybatis/mapper/oracle/category.xml</value> -->
                <value>classpath*:/config/mybatis/mapper/${Globals.DbType}/**/*.xml</value>
                </array>
        </property>
    </bean>

@Repository
public class BoardDAOMybatis extends SqlSessionDaoSupport     implements BoardDAO  {
    private static final Logger logger = LoggerFactory.getLogger(BoardDAOMybatis.class);
    private String namespace="config.mybatis.mapper.oracle.board";

    public void insertBoard(BoardBean board) {
        getSqlSession().insert(namespace + ".insertBoard", board);
    }

.....
```