2024/10/24 8일차

강사님이 정리한 것

mybatis

- SQL Mapper Framework
- SQL 실행에 필요한 객체와 SQL 실행 후 조회된 데이터를 저장할 객체을 매핑하면 PreparedStatement처리와 ResuletSet처리에서 자동으로 값을 매핑시켜서 SQL를 실행한다.

mybatis의 내장 타입 별칭

java.lang.Integer → int,

java.lang.Long → long,

java.lang.Double → double, String → string,
java.util.Date → date,

java.util.Map → map,

java.util.HashMap - > hashmap,

java.util.ArrayList → arraylist
java.math.BigDecimal → bigdecimal

mapper 파일 작성 규칙

* select 쿼리

```
<select id="쿼리 식별자" parameterType="전체클래스명 혹은 별칭"
                resultType="전체클래스명 혹은 별칭">
    SELECT 구문
</select>
```

* insert 쿼리

```
<insert id="쿼리 식별자" parameterType="전체클래스명 혹은 별칭">
    INSERT 구문
</insert>
```

```
<insert id="쿼리 식별자" parameterType="전체클래스명 혹은 별칭">
    <selectKey keyProperty="조회된 값이 저장될 필드명"
        resultType="조회된 값의 타입"
        order="selectKey구문의 실행 순서(BEFORE, AFTER 중 하나)">
        PK로 사용할 값을 조회하는 구문
    </selectKey>
    INSERT 구문
</insert>
* Primary Key를 조회해서 먼저 필드에 저장하고 insert 작업을 실행
* insert작업이 종료되면 객체의 필드에 시퀀스값이 저장되어 있다.

작성예)
Order order = new Order();
order.setUserNo(user.getNo());
order.setCreateDate(new Date());
-----------------------------------⟶ Order [no=0, userNo=20, creatDate=2020-01-20]

<insert id="insertOrder" parameterType="Order">
    <selectKey keyProperty="no"
        resultType="int"
        order="BEFORE">
        select bookstore_orders_seq.nextval from dual
    </selectKey>
    insert into (order_no, user_no, create_date)
    values (#{no}, #{userNo}, #{createDate})
</insert>
orderDao.insertOrder(order);
-----------------------------------⟶ Order [no=100, userNo=20, creatDate=2020-01-20]
OrderItem item = new OrderItem();
item.setOrderNo(order.getNo());  // 시퀀스로 획득한 값이 Order에 저장되어 있기 때문에
item.setBookNo(100000)           // 현재 저장된 주문번호가 필요하면 Order객체에서 조회하면 된다.
item.setAmount(2);
```

* ${} 표현식을 사용한 문자열 대체하기
- #{}표현식은 SQL구문의 ?에 값을 셋팅할 때 사용한다.
- #{}표현식은 SQL의 키워드, 테이블명, 컬럼명 등의 자리에 사용할 수 없다.
- ${}표현식은 SQL의 키워드, 컬럼명의 자리에 사용해서 그 문자열을 대체할 수 있다.
- ${}표현식은 SQL 구문에서 조회 결과에 영향을 미치지 않는
order by 절에서 정렬대상 컬럼명이나 정렬방식을 지정할 때만 사용하자.
- 작성예

```
<select id="getAllBooks" parameterType="map" resultType="Book">
    select *
    from bookstore_books
    order by ${columnName} ${sortType}
</select>
```

다이나믹 SQL

* mybatis는 ibatis나 다른 SQL Mapper 프레임워크보다 훨씬 강력한 동적 SQL 작성기능을 제공한다.
* mybatis의 동적 SQL 관련 태그는 JSTL의 태그와 사용법이 유사하다.
* 태그 종류

if 태그

```
<if test="조건식">
    조건식의 결과가 참인 경우 실행될 SQL 구문
</if>
```

* if 태그 사용시 주의점
문자열 비교에서 한 글자 비교는
<if test='grade == "A"'>와 같은 형식으로 조건식을 작성한다.
* 두 개 이상의 조건식을 사용될 때는 논리연산자가 필요하다.
논리연산자는 and나 or로 표시하면 된다.
- choose, when, otherwise 태그

```
<choose>
    <when test="조건식1">
        조건식1의 결과가 참인 경우 실행될 SQL 구문
    </when>
    <when test="조건식2">
```

조건식2의 결과가 참인 경우 실행될 SQL 구문
</when>
<otherwise>
*** else에 해당하는부분**
제시된 조건을 만족하지 못할 경우 실행될 SQL 구문
</otherwise>
</choose>

foreach 태그

*** 반복할때 사용하는 태그 (검색or태그 복수선택) 항목들 다 적어야함**

<foreach index="인덱스" 몇번째 근데 index는 잘 지정 안 함 무조건 적어야함
item="변수명"
collection="프로퍼티명"
separator="구분문자"
open="반복시작 전 표시할 내용"
close="반복종료 후 표시할 내용">
#{변수명}

</foreach>

* collection에는 배열, List, Set과 같이 반복할 대상이 위치한다.
* index는 현재 조회된 값의 index값이 전달된다.
* item에 지정된 변수에 값이 순서대로 하나씩 전달된다.

- where 태그

*** 1개라도 참이면 where 생성 and의 탈락여부를 결정 많이 사용됨**

<where>

항상 사용되는 조건이 있으면 안 적어도됨

if태그, choose,when,otherwise 등의 태그를 포함하고 있다.
</where>
* <where>태그 내부에서 동적 SQL의 실행결과로 SQL이 반환되면 where태그를 추가한다.
<where> 태그를 사용하면 아래와 같은 경우가 방지할 수 있다.
select *
from tables
where

* <where>태그 내부에서 동적 SQL의 실행결과로 반환된 SQL에 'and'나

'or'가 있으면 지워버린다.

아래의 동적쿼리에서 startDate가 null이고 endDate가 null이 아닌경우 아래와 같은 경우일 때 and를 제거한다.

```
select *
from tables
where
    and event_date <= '2019-12-31'
<where>
    <if test="startDate != null">
        event_date >= #{startDate}
    </if>
    <if test="endDate != null">
        and event_date <= #{endDate}
    </if>
    <if test="eventType != null">
        and event_type = #{eventType}
    </if>
</where>
```

set 태그

* update 구문에서 사용되는 태그다.
* update 항목들이 동적 SQL로 작성되어 있는 경우에만 사용하자.

*** update 항목의 맨 마지막 항목에 ,가 있으면 자동으로 제거한다.**

```
update bookstore_user
<set>
    <if test="point > 0">
        user_point = #{point},
    </if>
    <if test="grade != null">
        user_grade = #{grade},
    </if>
    <if test="filename != null">
        user_image_filename = #{filename},
    </if>
    <if test="enabled != null">
        user_enabled = #{enabled}
    </if>
```

```
        </set>
        where
            user_no = #{no}
```

- 두 개 이상의 테이블을 조인하는 경우
    * <resultMap/> 태그를 사용해서 조인결과를 VO객체와 매핑할 수 있다.
    * 작성예

```
    <resultMap id="식별자" type="클래스의 별칭 혹은 클래스의 전체이름">
        <id property="객체의 변수명" column="컬럼명 혹은 컬럼의 별칭" />
        <result property="객체의 변수명" column="컬럼명 혹은 컬럼의 별칭"
/>
        <result property="객체의 변수명" column="컬럼명 혹은 컬럼의 별칭"
/>
        <result property="객체의 변수명" column="컬럼명 혹은 컬럼의 별칭"
/>
        <result property="객체의 변수명" column="컬럼명 혹은 컬럼의 별칭"
/>
        <association property="객체의 변수명" javaType="클래스의 별칭 혹
은 클래스의 전체이름">
            <id property="객체의 변수명" column="컬럼명 혹은 컬럼의 별칭"
/>
            <result property="객체의 변수명" column="컬럼명 혹은 컬럼의
별칭" />
            <result property="객체의 변수명" column="컬럼명 혹은 컬럼의
별칭" />
        </assocication>
        <collection property="객체의 변수명" ofType="클래스의 별칭 혹은 클
래스의 전체이름">
            <id property="객체의 변수명" column="컬럼명 혹은 컬럼의 별칭"
/>
            <result property="객체의 변수명" column="컬럼명 혹은 컬럼의
별칭" />
            <result property="객체의 변수명" column="컬럼명 혹은 컬럼의
별칭" />
        </collection>
    </result>
```

    * <resultMap/> 태그에서 <id />는 Primary Key 값에 해당하는 컬럼을 매핑할 때 사용한다.

나머지 <result/>는 일반 컬럼들을 매핑할 때 사용한다.

* <resultMap/> 태그에서 <association>은 복잡한 타입(객체의 하나)의 연관관계를 매핑한다.

* <resultMap/> 태그에서 <collection>은 복잡한 타입의 콜렉션에 대한 연관관계를 매핑한다.

* 작성예

```
class Dept {
    int id;
    String name;
    Manager mgr;
    List<Emp> employees;
}

class Manager {
    int id;
    String name;
    }
}

class Emp {
    int id;
    String firstName;
```

* 부서정보 + 관리자정보 + 소속사원들

```
<resultMap id="DeptDetail" type="Dept">
    <id property="id"   column="dept_id" />
    <result property="name" column="dept_name"/>
    <association property="mgr" javaType="Manager">
        <id property="id"   column="mgr_id" />
        <result property="name"     column="mgr_name"/>
    </association>
    <collection property="employees" ofType="Emp">
        <id property="id"         column="emp_id"/>
        <result property="firstName"      column="emp_name"/>
    </collection>
</resultMap>
<select id="complexSql" parameterType="int"
resultMap="DeptDetail">
    select
        A.department_id             dept_id,
        A.department_name           dept_name,
        B.employee_id               mgr_id,
        B.first_name            mgr_name,
        C.employee_id               emp_id,
        C.first_name            emp_name
    from departments A, employees B, employees C
    where A.manager_id = B.employee_id
    and A.department_id = C.department_id
```

```
        and A.department_id = #{value}
    </select>
```

spring-mybatis 연동하기
1. pom.xml에 의존성 추가하기
mybatis, mybatis-spring 의존성 추가
2. /META-INF/mybatis 폴더 생성하기
/META-INF/mybatis/mybatis-config.xml 추가
* NULL값 추가 설정
* TypeAlias 설정
/META-INF/mybatis/mappers폴더 추가
3. Dao 인터페이스 추가
kr.co.jhta.xxx.dao 패키지에 xxxDao 인터페이스 추가
* xxxDao 인터페이스는 각각의 테이블에 대한 CRUD작업을 정의한다.
4. /META-INF/mybatis/mappers 폴더에 매퍼파일 추가
* xxxDao 하나당 매퍼파일 하나를 생성한다.
* 매퍼파일의 <mapper namespace="xxxDao의 전체이름">으로 작성한다.
* 매퍼파일의 sql 구문 식별자(id)는 xxxDao의 메소드명과 반드시 일치해야 한다.
* 매퍼파일의 sql 구문에서 parameterType과 resultType은
xxxDao의 메소드의 메소드 인자타입, 반환값과 일치해야 한다.
(단, 반환값이 List<클래스타입> 인 경우는 resultType에 클래스타입만 적는다.)
5. bean configuration 파일 설정하기
* DataSource 빈 등록
* SqlSessionFactory 빈 등록 <--- SqlSessionFactoryBean을 이용해서 등록

```
<bean id="sqlSessionFactory"
class="org.mybatis.spring.SqlSessionFactoryBean">
<property name="dataSource" ref="dataSource"></property>
<property name="configLocation" value="classpath:/META-
INF/mybatis/mybatis-config.xml" />
<property name="mapperLocations" value="classpath:/META-
INF/mybatis/mappers/*.xml" />
</bean>
```

* ConnectionPool, mybatis 환경설정 파일, mybatis 매퍼파일 경로 주입

* MapperScannerConfigurer 빈 등록

```
<mybatis-spring:scan base-package="kr.co.jhta.xxx.dao"
                     factory-ref="sqlSessionFactory"/>
```

* base-package는 xxxDao 인터페이스가 정의되어 있는 패키지를 지정한다.
* factory-ref에는 mybatis의 핵심객체 SqlSessionFactory를 주입한다.

* <mybatis-spring:scan> 태그는 MapperScannerConfigurer을 스프링의 빈으로 등록한다.
* MapperScannerConfigurer은 base-package로 지정된 패키지에서
xxxDao 인터페이스를 전부 스캔해서 xxxDao 인터페이스에 대한
구현객체를 자동으로 생성하고, 생성된 구현객체를 스프링의 빈으로 등록한다.
* 스프링의 빈으로 등록된 구현객체들은 xxxDao 인터페이스 타입으로 주입받을 수 있다.

6. xxxServiceImpl에서 xxxDao 주입받기

```
@Service
public class xxxServiceImpl implements xxxService {
    @AutoWired
    xxxDao dao;   // xxxDao인터페이스 타입의 필드를 정의하고
                  // @AutoWired 어노테이션을 부착해두면
                  // 스프링 컨테이너는 보유하고 있는 빈들 중에서
                  // xxxDao 타입의 객체를 주입한다.
    // xxxDao 타입의 객체는 MapperScannerConfigurer이 구현해서 등록한 객체다.

}
```

강사님이 정리한 것

Map<String, Object>

```
=====================================================
Map<String, Object> m= new hashMap<>();
m.put("title","자바");
=====================================================


=====================================================
boardMapper.searchBoard(m);
=====================================================


=====================================================
List<Board) searchBoard(Map<String, Object> condition;
```

```
<select id="searchBoard" parameterType="java.utill.Map">
    select *
    from tb_boards
    where board_title like #{title}
</select>
=====================================================
<select id="searchBoard" parameterType="map">
    select *
    from tb_boards
    where board_title like #{title}
</select>
=====================================================


List<Board> searchBoard(@Param("condition")Map<String,Objec
t>cond);

<select id="searchBoard">
    select *
    from tb_baords
    where board_title like #{condtion.title}
</select>
```

```
<insert id="insertBoard">
        insert into tb_boards
        (board_no
        , board_title
        , board_content
        , user_no)
        values
        (tb_boards_seq.nextval
        , #{board.title}
        , #{board.content , jdbcType=CLOB}  대용량 데이터일때
CLOB이라고 표기 대용량 데이터를 넣을때 필요함
        , #{board.user.no})
    </insert>
```

💡 **오후수업**

```
<insert id="insertUser">
      <selectKey keyProperty="user.no" resultType="int" ord
er="BEFORE"> 오라클 한정 order="BEFORE"  Mysql은 orde="AFTER"
애 먼저 실행됨
         select tb_users_seq.nextval
         from dual
      </selectKey>

      insert into tb_users
      (user_no, user_id
       , user_password
       , user_name
       , user_email)
      values
      (#{user.no}
       , #{user.id}
       , #{user.password}
       , #{user.name}
       , #{user.email})
   </insert>
```

<selectKey>

💡 **중요하게 자주 사용함**

<selectKey>를 이용해서 자동생성된 값 이용하기

- 자동 생성되는 값
  - 테이블에 새로운 행을 추가할 때 자동으로 생성되어 컬럼에 저장된 값
  - * 오라클 : 시퀀스
  - * MySQL : 새로운 행 추가시 컬럼의 값이 자동으로 증가된다.

<selectKey>의 활용

- 데이터베이스 신규 데이터를 저장할 때 2개 이상의 테이블에 데이터가 저장되고,
  해당 테이블이 부모 테이블과 자식 테이블의 관계에 있을 때 사용할 수 있다.
- <selectKey>를 사용하면 부모테이블에 데이터를 추가할 때 자동 생성된 값(기본키 값)을
  부모 테이블을 표현한 VO객체의 멤버변수에 저장할 수 있다.
- 자식테이블에 데이터를 추가할 때 부모테이블의 기본키 값을 부모 VO객체의 멤버변수에 저장된 값을 가져오면 된다.

<selectKey>의 주요속성

- <selectKey order=″selectKey실행시점″
  keyProperty=″selectKey로 실행한 SQL구문의 결과값이 저장되는 멤버변수명″
  resultType=″selectKey로 실행한 SQL구문의 결과값 타입″
  자동생성되는 값을 조회하는 SQL 구문
  </selectKey>

```
[ Oracle ]

User 객체 int no = 0 ; String id = "hong" String pwd = "zxcv
1234" String name = "홍길동"
-> <selectKey>의 SQL구문실행
-> insert into users SQL 구문 실행
-> User 객체 int no = 31 ; String id = "hong" String pwd =
"zxcv1234" String name = "홍길동"

<insert id ="insertUser" parameterType="kr.co.jhta.vo.User"
    <selectKey keyProperty="no" resultType="int" order="BEF
ORE">  |
        select user_seq.nextval from dual
```

```
| <- 이게 먼저 시행됨  <selectKey>의 SQL구문실행 후
    </selectKey>
|

insert into user
|
(user_no,user_id, user_pwd, user_name)
|
values
|   insert into users SQL 구문 실행
(#{no},#{id}, #{pwd}, #{name})
|
</insert>
|

[ MySQL ]

User 객체 int no = 0 ; String id = "hong" String pwd = "zxcv
1234" String name = "홍길동"
-> insert into users SQL 구문 실행
         + users테이블에 새로운 행이 추가됨
         + 새로 추가된 행의 user_no에 자동증가된 사용자번호 값이 있
다.
-> <selectKey>의 SQL구문실행
         + 새로 추가된 행의 user_no컬럼값을 User객체의 no에 대입한
다.
-> -> User 객체 int no = 31 ; String id = "hong" String pwd
= "zxcv1234" String name = "홍길동"

<insert id="insertUser" parameterType="kr.co.jhta.vo.User">
    insert into users
(user_id, user_pwd, user_name)
    values
    (#{id, #{pwd},#{name}}
 <selectKey keyColumn="user_no"
                         keyProperty="no" resultType="int">
                         order="AFTER"
        select last_insert_id()
```

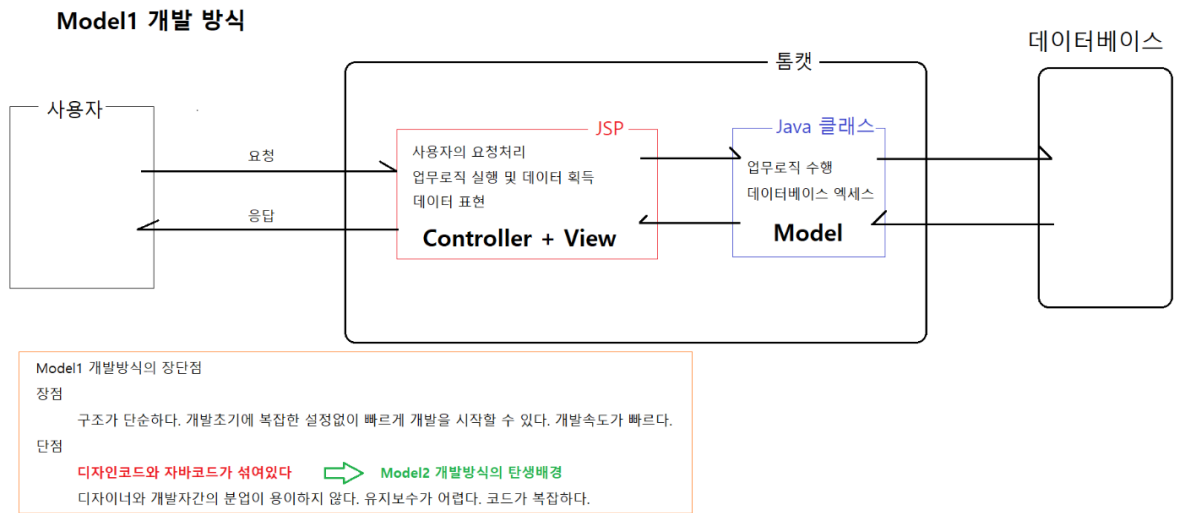
```
    </selectKey>
</insert

${column} $ : 컬럼명 자제치환, 키워드 * 보안에 위험함 사용자가 입력한 값이 치환되게하면 안됨  문자열 대체
#{value}  # : 값 치환
```

모델1 vs. 모델2

모델1

- **JSP로만 구현한 웹 애플리케이션이다.**
- **클라이언트의 요청을 JSP가 받아서 처리하는 구조다.**
- **JSP 페이지에 비즈니스로직을 처리하는 코드와 HTML 코드가 섞여있다.**
- **모델1 구조의 요청처리 흐름**



모델2

- **MVC패턴을 적용해서 구현한 웹 애플리케이션이다. 업무를 나눈것이다**
  - **M: Model V: View C: Controller**
    - **M: 업무로직, 데이터**

- **V: 데이터 표현 (JSP. JSP가 가벼워짐)**
- **C: 클라이언트와 상호작용**
  *** 프론트 엔드 기준 Model : HTML View: CSS Controller: JavaScript**

- **비즈니스로직과 디자인 코드를 분리하는 것이 목적이다.**
- **모델2 구조의 요청처리 흐름**