### 3. 동적 SQL

하나의 SQL문을 다양한 형태로 실행하는 기능이 동적 SQL이다.

마이바티스는 if, choose(when, otherwise), trim(where, set), foreach 요소의 동적 SQL문의 작성 방법이다.

예를 들어 student 테이블에서 검색하는 SELECT문이 ① 전체 행을 검색하고 ② "김" 성을 검색하고 ③ 2학년의 "박"씨 성을 검색하는 3개의 SELECT문은 동적 SQL을 사용하여 한 문장으로 매퍼 XML 구문으로 작성할 수 있다.

① SELECT * FROM student

② SELECT * FROM student WHERE name LIKE '김%'

③ SELECT * FROM student WHERE name LIKE '박%' AND year=2

(1) if

<if>요소는 test의 조건문이 참일 때 연결문자열을 SQL문에 연결하고, 그렇지 않으면 무시된다. 동적 SQL에서 가장 공통적으로 사용되며 SQL문의 WHERE절의 포함 여부를 작성할 수 있다.

```
표기법 : <if test="조건문"> 연결문자열 </if>
```

클라이언트에서 전송된 name이 존재하지 않을 경우 "컴퓨터공학과" 학과의 학생들 전부 출력하고, name이 존재하면 "컴퓨터공학과" 학과 학생 중에서 name으로 검색된 학생들만 출력한다

```
<select id="selectSearch" resultType="studentVO">
    select * from student where dept_name="컴퓨터공학과"
    <if test="name !=null"> and name like #{name} || '%' </if>
</select>
```

(2) choose, when, otherwise

choose ~ when ~ otherwise는 if와 함께 가장 공통적으로 제공되는 분기 처리 방식이다. choose 하위의 when 엘리먼트에서 만족하는 조건이 하나라도 나오면 해당 조건의 결과를 동적 SQL에 적용하고 하나라도 없으면 otherwise 엘리먼트의 결과를 동적 SQL에 사용한다.

```
<choose>
    <when test="조건문1"> 연결문자열1</when>
    ...
    <otherwise>연결문자열n</otherwise>
</choose>
```

student 테이블에서 학과코드나 성별로 검색하고, 그렇지 않으면 1학년을 검색하는 경우는 다음과 같다.

```
<select id="selectFind" resultType="studentVO">
    select * from student
      <choose>
          <when test="dept_name != null"> where dept_name=#{deptName} </when>
          <when test="dept_sex != null"> where dept_sex=#{deptSex} </when>
          <otherwise> where year=1 </otherwise>
      </choose>
```

```
</select>
```

(3) trim, where, set

if, choose 등을 사용하여 조건의 결과가 참이 되지 않을 경우 완전하지 못한 SQL문의 요소를 제거하는 trim, where, set이 있다.

· <trim>은 WHERE절에 기술한 AND나 OR의 연산자를 제거한다.

```
표기법 : <trim prefix="WHERE" prefixOverrides="AND | OR"> ... </trim>
```

trim 엘리먼트 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.

· <where>은 SQL문내에 기술한 <where> ~ </where>내의 조건의 모두 참이 되지 않을 경우 <where> ~ </where>의 모든 문자열을 제거한다.

```
표기법 : <where>
            <if test="조건문1"> 연결문자열1</if>
            ...
            <if test="조건문n"> 연결문자열n</if>
         </where>
```

· <set>은 동적인 UPDATE문의 <if>문이 모두 참이 되지 않을 경우 <set> ~ </set>를 문자열을 제거한다.

```
표기법 : <set>
            <if test="조건문1"> 연결문자열1</if>
            ...
            <if test="조건문n"> 연결문자열n</if>
         </set>
```

다음과 같은 동적인 SELECT문은 <if>의 조건에 따라 부적합한 SELECT문이 생성되어 오류가 발생될 수 있다.

```
<select id="selectFind" resultType="studentVO">
    select * from student
    where   <if test="dept_name!=null"> dept_name=#{deptName} </if>
            <if test="dept_sex!=null"> dept_sex=#{deptSex} </if>
            <if test="year!=null"> year=#{year} </if>
</select>
```

위 SQL문에서 test 조건이 모두 참이 아닐 경우 즉 null이면 where만 남게 된다.

```
select * from student where
```

이 동적인 SQL문은 <where> ~ </where>내에 기술하면 문제를 해결할 수 있다.

```
<select id="selectFind" resultType="studentVO">
    select * from student
    <where> <if test="dept_name!=null"> dept_name=#{deptName} </if>
            <if test="dept_sex!=null"> dept_sex=#{deptSex} </if>
            <if test="year!=null"> year=#{year} </if>
   </where>
</select>
```

<where>의 조건이 참이 되면 "WHERE"를 추가하고, 그렇지 않으면 제거한다.

<set>은 동적인 update문에서 <if>의 조건이 참이 될 때 set을 추가하고 불필요한 컴마(,)를 제거한다.

department 테이블을 수정하기 위한 update문에서 동적인 SQL의 예이다.

```
<update id="updateDepartment">
    update department
    <set> <if test="dept_name!=null"> dept_name=#{deptName} ,</if>
          <if test="dept_tel!=null"> dept_tel=#{deptTel} </if>
    </set>
    where dept_id=#{deptId}
</update>
```

<trim>요소로 제거할 prefix와 suffixOverrides를 기술한다.

```
<update id="updateDepartment">
    update department
        <trim prefix="SET" suffixOverrides=",">
            <if test="dept_name!=null"> dept_name=#{deptName} ,</if>
            <if test="dept_tel!=null"> dept_tel=#{deptTel} </if>
        </trim>
    where dept_id=#{deptId}
</update>
```

(4) foreach

SELECT문의 WHERE절에 IN 연산자를 사용할 경우, <foreach>는 컬렉션 명시를 허용하고, item과 index를 사용하여 복수 개의 값을 사용할 수 있다. 복수 개의 값은 open("("), sepaerator(","), close(")")로 구분자를 둘 수도 있다. 파라메타 객체가 마이바티스에 list나 배열로 전달될 때 list와 배열은 "list"와 "array" 키로 사용하고, Map은 "맵명"을 키로 한다.

```
<foreach collection="키" item="아이템" index="인덱스" open="(" close=")" separator=",">
            #{item}
 </foreach>
```

foreach 엘리먼트는 다음과 같은 속성을 가진다.

① collection : 값 목록을 가진 객체를 설정하면 된다. 파라미터의 타입은 배열이나 List 모두 처리가 가능

하다.

② item : 목록에서 각각의 값을 사용하고자 할 때 사용하는 속성이다.

③ index : 몇 번째 값인지 나타내는 인덱스 값이다. 0부터 시작한다.

④ open : 목록에서 값을 가져와서 설정할 때 가장 앞에 붙여 주는 문자를 지정한다. IN절에서는 대부분 (로 시작한다.

⑤ close : 목록에서 값을 가져와서 설정할 때 가장 뒤에 붙여 주는 문자를 지정한다. IN절에서는 대부분 ) 로 끝난다.

⑥ separator : 목록에서 값을 가져와서 설정할 때 값들 사이에 붙여주는 문자를 지정한다. IN절에서는 값 사이에 쉼표(,)를 붙여준다.

```
<select id="selectPostIn" resultType="domain.blog.Post">
      select *
      from post
      where id in
            <foreach item="item" index="index" collection="list" open="(" separator="," close=")">
               #{item}
            </foreach>
</select>
```

department 테이블에서 전체 행이나 검색 조건을 적용하는 매퍼 XML 구문 작성

```
<select id="listDepartment" paramType="vo.DeptVO" resultType="vo.DeptVO">
     select * from department
     <where>
           <if test="department != null and deptname != ' ' ">
                and dept_name like '%' || #{deptname} || '%'
            </if>
     </where>
</select>
```

## 4. 매퍼

매퍼(mapper)는 매핑된 구문을 주입하기 위한 인터페이스이며, 반드시 인터페이스로 선언해야 한다. 마이바티스의 기본적인 자바 인터페이스는 SqlSession이며, SqlSession인터페이스는 SQL 매퍼 구문을 실행하고 커밋이나 롤백의 트랜잭션을 관리할 수 있다. SqlSessionFactroy 객체가 마이바티스의 전반적인 정보를 가지고 제어한다. SqlSessionFactroy 객체를 생성하기 위해 SqlSessionFactroyBuilder 객체를 먼저 선언한다. SqlSessionFactroy 객체는 SqlSessionFactroyBuilder 의 build 메소드를 사용해서 생성한다.
마이바티스-스프링에서 마이바티스 API를 직접 사용하는 방법이 있다.
스프링에서 SqlSessionFactroyBean을 이용해서 SqlSessionFactroy를 생성하고, 코드상에서 SqlSessionFactroy를 사용한다.

매퍼 XML 구문 실행의 매퍼 인터페이스

```
public interface DeptMapper {
    public List<DeptVO> listDepartment(DeptVO param);
    public DeptVO selectDepartment(DeptVO param);
    public int insertDepartment(DeptVO param);
    public int updateDepartment(DeptVO param);
    public int deleteDepartment(DeptVO param);
}
```

(1) 매퍼 XML 구문의 실행 메서드
실행 메서드들은 매퍼 XML 파일에 정의된 select, insert, update, delete 요소를 실행한다. 메서드명 자체가 그 역할을 설명하도록 명명되었고, id와 파라메터 객채(원시타입, 자바빈, Map)를 가진다.
select 요소를 실행하는 메서드는 selectOne(), selectList(), selectMap() 세가지가 있다.
· selectOne 메서드는 오직 하나의 객체만을 반환하는 메서드이다. 한 개 이상 또는 null이 반환되면 예외가 발생한다.
· selectList 메서드는 복수 개의 객체를 반환하는 메서드이다.
· selectMap 메서드는 복수 개의 객체를 반환(Map)하는 메서드이다.

insert, update, delete 요소는 insert(), update(), delete()의 실행 메서드가 있으며, 실행한 후 트랜잭션의 수가 반환된다.

## 5. 마이바티스와 스프링 설정

마이바티스와 스프링을 연동하기 위해서는 다음과 같은 설정이 필요하다.

- 데이터베이스 서버(JDBC 드라이버, url, 계정이름, 암호)의 dataSource
- SqlSessionFactroy
- 매퍼 인터페이스
- 트랜잭션
- SqlSession
- Mapper 주입

(1) dataSource

dataSource는 데이터베이스 서버에 대한 정보로 JDBC 드라이버, url, 계정이름, 암호 정보가 필수 요소이며, 스프링 설정 파일에 구체적인 정보를 설정하거나 별도의 프로퍼티 파일을 작성하여 호출할 수도 있다.

① 데이터베이스 서버의 프로퍼티 파일

프로퍼티 파일에는 기본적인 드라이버, url, 계정이름, 암호를 추가한다. 일반적으로 파일명은 jdbc와 properties 확장자를 붙인다. 프로퍼티 파일이 저장되는 위치는 src의 classpath 또는 "WEB-INF" 하위폴더에 저장하면 된다. 다음은 *.properties 파일의 일반적인 설정 내용이다.

jdbc.properties

```
jdbc.driverClass=oracle.jdbc.driver.OracleDriver
jdbc.url=jdbc:oracle:thin:@127.0.0.1:1521:XE
jdbc.username=hr
jdbc.password=hr
```

② 스프링 설정 파일에서 dataSource 설정

스프링 설정 파일에 PropertyPlaceholderConfigurer의 빈을 설정하고, <value> 속성으로 프로퍼티 파일이 저장된 **"경로:프로퍼티명"**을 설정한다.

WEB-INF/**/*-context.xml

```
<bean id="propertyConfigurer"
class= "org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
        <property name="locations">
              <list><value>classpath:jdbc.properties</value></list>
        </property>
        <property name="fileEncoding" value="utf-8"/>
</bean>
```

③ dataSource 빈 설정

jdbc.properties에서 읽은 값으로 dataSource를 설정한다. dataSource의 빈은 DriverManagerDataSource 클래스명을 사용하여 빈 설정하고, <property>와 <value> 속성으로 지정한다.

WEB-INF/**/*-context.xml

```
<!-- DataSource -->
<bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
```

```
        <property name="driverClassName" value="${jdbc.driverClass}"/>
        <property name="url"                value="${jdbc.url}"/>
        <property name="username"          value="${jdbc.username}"/>
        <property name="password"          value="${jdbc.password}"/>
</bean>
```

### 6. SqlSessionFactoryBean

마이바티스-스프링 연동 모듈의 SqlSessionFactoryBean은 스프링의 팩토리빈 인터페이스를 구현한다. 이 설정은 SqlSessionFactoryBean를 생성하는 것이 아니라 팩토리에서 getXXX() 메서드를 호출 결과를 반환하는 것을 의미한다. 스프링은 애플리케이션 시작 시점에 SqlSessionFactory를 생성하여 저장한다.
스프링 설정 파일에 팩토리 빈을 생성하는 XML 설정은 다음과 같다.

WEB-INF/**/*-context.xml

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

일반적으로 마이바티스-스프링에서는 SqlSessionFactoryBean이나 sqlSessionFactory를 직접 사용하기 보다는 **sqlSessionDaoSupport**나 **MapperFactoryBean**을 확장하여 다른 DAO에 주입된다.

(1) <property> 속성

sqlSessionFactory는 JDBC dataSource의 필수 프로퍼티로, dataSource의 연결을 설정해야만 한다. 마이바티스 XML 설정 파일의 위치를 지정하기위해 사용되는 configLocation을 프로퍼티로 지정 할 수도 있다. 마이바티스 XML 설정 파일이 매퍼 클래스와 동일한 클래스 패스에 있지 않으면 프로퍼티를 설정하여야 한다.

① 마이바티스 설정파일에 <mappers> 요소로 XML 파일의 클래스 패스 지정.

② 팩토리 빈의 mapperLocations 프로퍼티 사용한다. mapperLocations 프로퍼티는 매퍼의 자원 위치나 마이바티스 XML 매퍼 파일들의 위치를 명시할 때 사용된다.

## 7. 트랜잭션

마이바티스-스프링 연동 모듈을 사용하면 스프링 트랜잭션에 자연스럽게 연동 될 수 있다. 마이바티스에 종속되는 트랜잭션 관리보다 스프링의 DataSourceTransactionManager로 트랜잭션을 설정할 수 있으며, 3 가지 설정 방법이 있다.

(1) 표준 설정 방법

스프링 트랜잭션을 가능하게 하려면 스프링 XML 설정파일에 스프링의 DataSourceTransactionManager를 생성한다. dataSource는 필수 프로퍼티이며, SqlSessionFactoryBean을 생성한 것과 반드시 같아야 한다.

```
<!-- a PlatformTransactionManager is still required -->
<bean id="transactionManager"
  class= "org.springframework.jdbc.datasource.DataSourceTransactionManager">
        <property name="dataSource" ref="dataSource" />
</bean>
```

(2) 컨테이너 관리 트랜잭션

컨테이너 관리 트랜잭션은 컨테이너 서버의 설정으로 컨테이너 객체들의 트랜잭션을 관리하는 방법이다. 다중 자원 접근을 위한 JTA(Java Transaction API)로 트랜잭션을 관리할 경우 스프링 트랜잭션 네임스페이스를 사용하여 설정할 수 있다.

지원하지 않는다.

```
<tx:jta-transaction-manager />
```

(3) 프로그램의 트랜잭션 관리

마이바티스 sqlSession은 트랜잭션을 메서드로 제어하지만, 마이바티스-스프링 연동 모듈은 빈을 스프링이 관리하는 sqlSession이나 스프링이 관리하는 매퍼에 주입하여 트랜잭션을 관리한다. 스프링이 관리하는 sqlSession이나 주입된 매퍼 클래스에서는 sqlSession.commit(), sqlSession.rollback() 또는 sqlSession.close() 메서드를 호출할 수가 없다.

### 8. sqlSession

마이바티스-스프링 연동 모듈은 사용자 빈이 sqlSession에 주입되고 sqlSession은 스프링 트랜잭션 설정에 따라 세션을 자동으로 커밋/롤백한다.

(1) sqlSessionTemplate

sqlSessionTemplate은 마이바티스-스프링 연동 모듈의 핵심으로 sqlSession을 구현하고, 코드에서 sqlSession을 대체하는 역할을 한다. sqlSessionTemplate은 여러 개의 DAO나 매퍼에서 공유할 수 있다. SQL을 처리하는 마이바티스 메서드를 호출할 때 sqlSessionTemplate은 sqlSession이 현재의 스프링 트랜잭션에서 사용될 수 있도록 보장하고, 필요한 시점에 세션을 닫고, 커밋과 롤백, 그리고 세션의 생명 주기를 관리한다. sqlSessionTemplate은 <constructor-arg> 생성자로 sqlSessionFactory로 의존 주입할 수 있다.

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

sqlSessionTemplate로 설정된 sqlSession는 <property>로 설정하여 빈에 직접 주입한다.

```
<bean id="deptMapper" class="DeptMapperImpl">
    <property name="sqlSession" ref="sqlSession" />
</bean>
```

(2) sqlSessionDaoSupport

sqlSessionDaoSupport는 sqlSession을 제공하는 추상 클래스이다. **서비스 로직의 DAO 구현체에서 getSqlSession() 메서드**로 SQL문을 처리하는 마이바티스 메서드를 호출할 sqlSessionTemplate을 얻을 수 있다.

```
import org.mybatis.spring.support.SqlSessionDaoSupport;
public class DeptDAOImpl extends SqlSessionDaoSupport implements DeptDAO {
    public DeptVO selectDepatment(DeptVO param) {
        return DeptVO.getSqlSession().selectOne("dao.DeptDAO.selectDepartment");
    }
}
```

### 9. 매퍼 주입

스프링 설정 파일에 매퍼 주입하고, 검색할 수 있도록 설정한다.

(1) XML 설정을 사용하는 매퍼 주입

매퍼는 스프링 설정 파일에 MapperFactoryBean으로 주입한다.

```
<bean id="deptMap" class="org.mybatis.spring.mapper.MapperFactoryBean">
    <property name="mapperInterface" value="spring.DeptMapper" />
    <property name="sqlSessionFactory" value="sqlSessionFactory" />
</bean>
```

DeptMapper의 인터페이스와 동일한 클래스패스에 마이바티스 XML 매퍼 파일이 있으면 MapperFactoryBean이 자동으로 파싱한다. 매퍼 XML 파일이 동일한 클래스 경로에 있다면 설정 파일에 매퍼를 주입할 필요는 없다.

(2) 매퍼의 자동 검색

마이바티스-스프링 연동 모듈은 3가지 설정 방법의 매퍼 자동 검색이 있다. <mybatis:scan>과 @MapperScan은 마이바티스-스프링 연동 1.2.0에서 추가된 기능이며, @MapperScan은 스프링 버전이 3.1. 이상이어야 한다.

① <mybatis:scan> 사용

스프링에서 제공하는 <context:component-scan>과 매우 유사한 방법으로 매퍼를 검색한다.

다음과 같이 <mybatis:scan>을 설정한다.

```
<mybatis:scan base-package="spring.mapper" />
```

"base-package" 속성은 매퍼 인터페이스 파일이 저장된 최상위 패키지를 지정한다. 세미콜론이나 콤마 구분자로 여러 개의 패키지를 설정할 수 있다. 매퍼는 지정된 패키지에서 하위 패키지를 모두 검색한다.

② @MapperScan 어노테이션 사용

@Configuration은 스프링의 자바 설정에서 @MapperScan을 선호하는 방법이며, @MapperScan 어노테이션은 다음과 같다.

```
@Configuration
@MapperScan("spring.mapper")
public class AppConfig{ ... }
```

③ 스프링 XML 파일에 MapperScannerConfigurer 설정

MapperScannerConfigurer는 빈처럼 XML 애플리케이션 컨텍스트에 포함하는 방법이다.

MapperScannerConfigurer를 스프링 설정 파일에 추가한다.

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

MapperScannerConfigurer는 에러를 발생시키는 PropertyPlaceholderConfigurer보다 먼저 실행되기 때문에 이 프로퍼티보다 SqlSessionFactoryBeanName 과 SqlSessionTemplateName 프로퍼티 사용을 권장한다.

마이바티스-스프링 연동 모듈과 스프링을 연동하는 스프링 설정 파일을 작성한다.(root-context.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<beans ... >
<bean id="propertyConfigurer"
class= "org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
        <property name="locations">
              <list><value>classpath:jdbc.properties</value></list>
        </property>
        <property name="fileEncoding" value="utf-8"/>
</bean>
<!-- DataSource -->
<bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
       <property name="driverClassName" value="${jdbc.driverClass}"/>
       <property name="url"                value="${jdbc.url}"/>
       <property name="username"          value="${jdbc.username}"/>
       <property name="password"          value="${jdbc.password}"/>
</bean>
<!-- a PlatformTransactionManager is still required -->
<bean id="transactionManager"
class= "org.springframework.jdbc.datasource.DataSourceTransactionManager">
           <property name="dataSource" ref="dataSource" />
</bean>
<tx:annotation-driven transaction-manager="transactionManager" />
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
           <property name="dataSource" ref="dataSource"/>
           <property name="mapperLocations" value="classpath:mybatis/dept/*.xml"/>
</bean>
<bean id="sqlSession" class="org.mybatis.spring.SqlSessionTemplate" destroy-method="clearCache">
           <constructor-arg name="sqlSessionFactory" ref="sqlSessionFactory"></constructor-arg>
</bean>
<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
       <property name="basePackage"> <value>com.spring.dept.dao</value></property>
</bean>
</beans>
```

스프링 설정 파일을 작성한다.(servlet-context.xml)

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

```
http://www.springframework.org/schema/mvc/spring-mvc.xsd
                http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
                http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">
        <!-- DispatcherServlet Context: defines this servlet's request-processing infrastructure -->
        <!-- 어노테이션 설정 -->
        <context:annotation-config />
        <!-- Enables the Spring MVC @Controller programming model -->
        <annotation-driven />
        <!-- Handles HTTP GET requests for /resources/** by efficiently serving up static resources in the
${webappRoot}/resources directory -->
        <resources mapping="/resources/**" location="/resources/" />
        <!-- Resolves views selected for rendering by @Controllers to .jsp resources in the /WEB-
INF/views directory -->
        <beans:bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
                <beans:property name="prefix" value="/WEB-INF/views/" />
                <beans:property name="suffix" value=".jsp" />
        </beans:bean>
        <context:component-scan base-package="com.spring.controller" />
</beans:beans>
```

### 10. 서비스 로직 인터페이스와 구현

매퍼 또는 DAO로 생성된 인터페이스는 데이터 처리만을 담당하고, 데이터를 처리하는 역할은 서비스 로직이다. 컨트롤러에서 데이터를 처리하는 경우에도 서비스 로직이다.

서비스 로직은 데이터베이스 접근 객체인 매퍼를 호출하는 인터페이스를 정의하고, 인터페이스의 구현 클래스에서 처리하도록 권장하고 있다. 이 처리 방법은 관계형 데이터베이스가 변경되더라도 소스 코드에 영향을 주지 않고 적용이 가능하다. 그리고 매퍼로 호출을 전달하므로 마이바티스에 대한 의존성이 없어진다. 서비스 로직의 인터페이스는 매퍼의 인터페이스와 유사한 구조로 작성된다. 구현 클래스에서 메서드으 반환값은 "매퍼명.메서드명(파라메터)"이다.