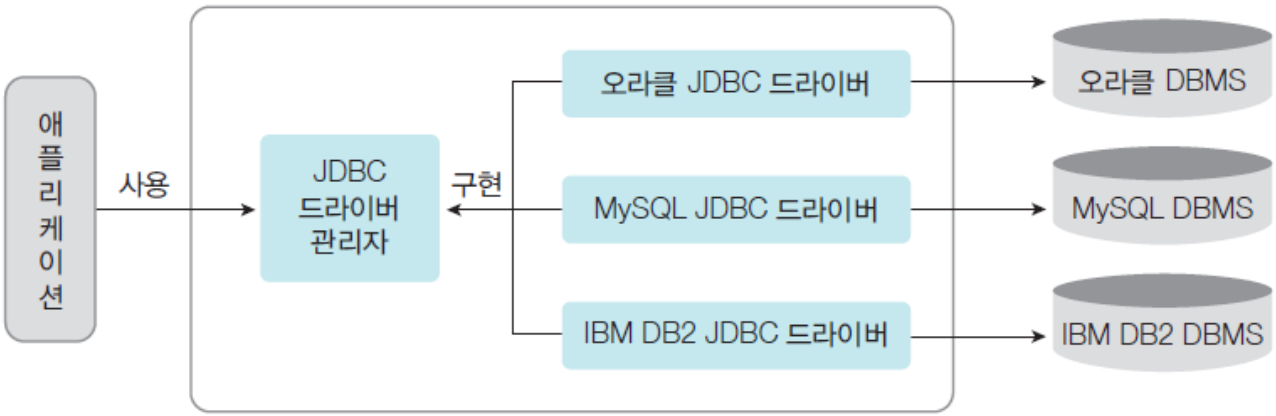
JDBC 정리
MyBatis Framework
iBatis vs MyBatis
myBatis & eGov

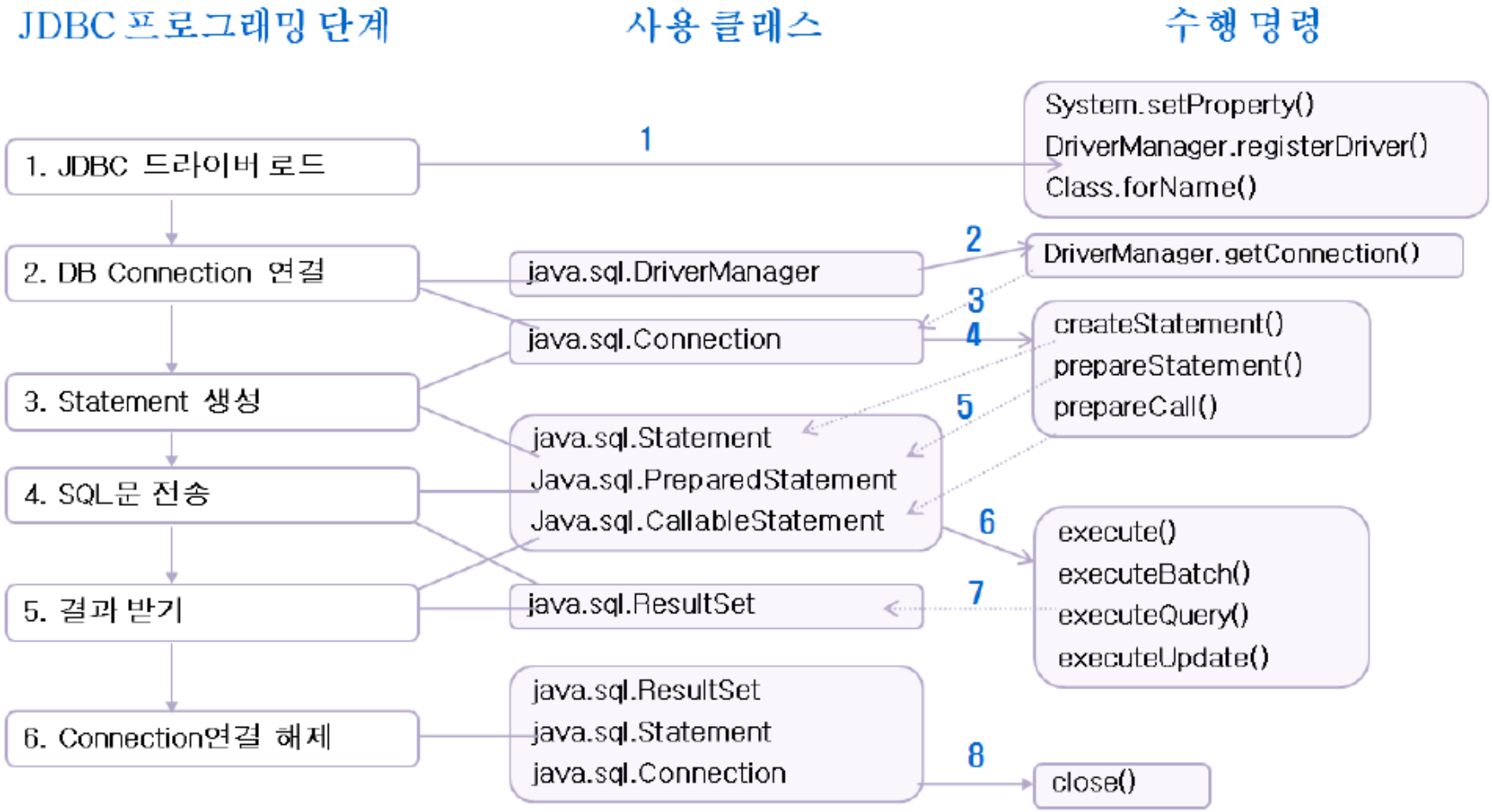
JDBC 정리

- JDBC는 Java database Connectivity의 약자이다.
- JDBC는 Java SE 기술 중의 일부이다.
- JDBC는 자바 프로그래밍에서 데이터베이스를 접속하는 데에 사용하는 API를 의미한다.

그림 10-22 JDBC의 동작 구조



JDBC 프로그래밍 절차



MySQL 설치 및 연결하기

- MySQL & Workbench를 설치한다.
- JDBC로 연결해본다.
  - Oracle과 연동하는 부분과 다른 점이 거의 없음을 느껴본다.

try ~ catch ~ finally로 작성하기

- JDBC로 데이터베이스 접속할 경우 커넥션의 누수 현상을 없애기 위해 finally 절로 반드시 close()를 해야 한다.

MyBatis 개요 1st

- MyBatis는 개발자가 지정한 SQL, 저장프로시저 그리고 여러 고급 매핑을 지원하는 퍼시스턴스 프레임워크이다.
- iBATIS라는 이름의 ASF 프로젝트였으나, 분리되어 Google Code 프로젝트가 되었다.
- 자바가 닷넷보다 느리지 않다는 증명하기 위해 petStore를 효율적으로 개발하는 과정에서 ORM 프레임워크가 탄생하게 되었고, iBATIS로 발표되었다.

MyBatis 개요 2nd

- iBATIS라는 단어는 abatis(가시 울타리) 를 연상하게 하며, MyBatis는 batis(작은 새)라는 이름을 연상하게 한다. iBATIS는 2001년에 시작한 ASF 프로젝트이다. 당시에는 암호화 솔루션을 개발하는 프로젝트였으나, 2002년에 프로젝트 목표를 변경했는데, 그 계기는 Microsoft에서 닷넷이 자바보다 10배가량 빠르다고 했기 때문이다.

MyBatis 개요 3rd

- World's most popular SQL based data mapping Solution 이라 할 수 있다.
- XML을 이용하여 SQL과 OR 매핑을 다채롭고 단순하기 기술할 수 있는 특징이 있다.
- Mapper라는 개념이 있다.
- 애플리케이션의 소스 코드를 타입에 안전하게 기술할 수 있다.
- 소스코드에서 SQL Query 부분을 유연하게 관리할 수 있다는 특징이 있으며, 가장 널리 사용되는 Java ORM Framework이다.

개발 환경 구축

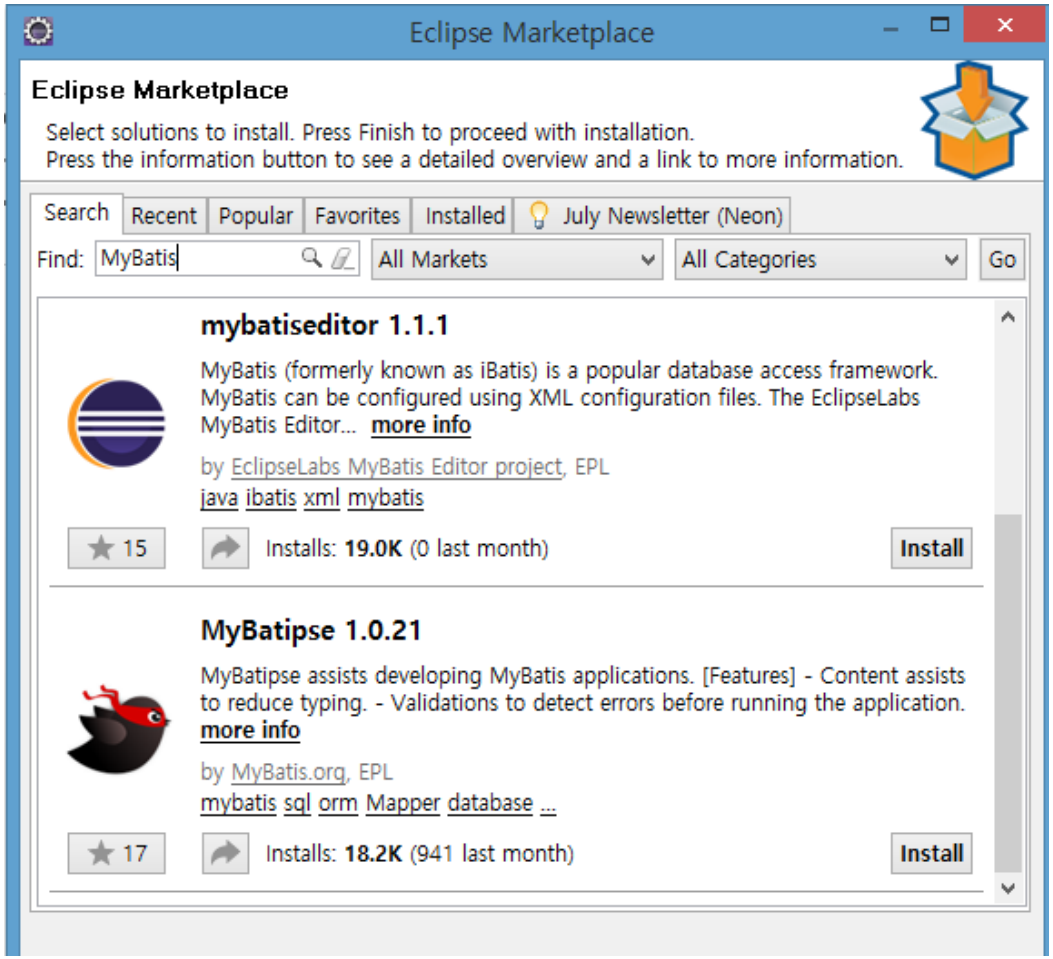
- MyBatis API를 다운로드한다.
- Eclipse 플러그인을 설치한다.
  - MyBaptise

개발 환경 구축 - MyBatis API Download

- blog.mybatis.org에서 MyBatis API를 다운로드한다.

개발 환경 구축 – Eclipse 플러그인 설치

- Eclipse 마켓플레이스에서 설치한다.



Connection Pooling

- 데이터베이스와의 커넥션이 필요할 때마다 새로 생성하는 것이 아니라 이미 생성되어 있는 커넥션을 재활용하는 개념이다.
- 이미 생성된 커넥션을 사용하면 커넥션을 새로 맺는 과정이 생략되므로 성능이 향상된다.
- 이러한 커넥션 풀은 검증된 오픈소스 등을 사용하는 경향이 강하다.

DataSource

- DataSource 서비스
  - 데이터베이스에 대한 연결을 제공하는 서비스이다. 다양한 방식의 데이터베이스 연결을 제공하고,이에 대한 추상화계층을 제공함으로써, 업무로직과 데이터베이스 연결방식 간의 종속성을 배제한다.
  - DataSource 객체 획득 방법
    - DriverManagerDataSource 빈 등록
    - BasicDataSource 빈 등록
    - ComboPooledDataSource 빈 등록
    - JNDI DataSource
    - JNDI Lookup을 이용하여 WAS에서 제공되는 JNDI tree로부터 DataSource를 가져옴
  - Connection 객체 획득 방법: dataSource.getConnection();

Open Source Connection Pools

- Commons DBCP
  - http://commons.apache.org/dbcp/
- C3P0
  - http://www.mchange.com/projects/c3p0/index.html

주요 파일들

- 설정파일
  - 데이터베이스 커넥션풀(Connection Pool) 등을 xml로 기술한다.
- SqlSessionFactory
  - SqlSession을 생성한다.
- SqlSession
  - SQL을 실행하고, 트랜잭션을 관리한다.
- Mapping XML
  - SQL 문장을 XML파일로 기술한다.

MyBatis를 통해 DataSource 얻는 과정

- mybatis-config.xml 로딩
- SqlSessionFactory 생성
- SqlSession 생성
- Connection 생성

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>
	<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:@localhost:1521:xe" />
			<property name="username" value="hr" />
			<property name="password" value="hr" />
		</dataSource>
		</environment>
	</environments>
</configuration>
```

SqlSession으로부터 Connection 얻기

```
public class MBTest1 {
    public static void main(String[] args) throws Exception {
        String resource = "./mybatis-config.xml";
        InputStream inputStream = Resources.getResourceAsStream(resource);
        SqlSessionFactory sqlSessionFactory =
        new SqlSessionFactoryBuilder().build(inputStream);
        SqlSession session= sqlSessionFactory.openSession();
        Connection connection= session.getConnection();
        Statement stmt= connection.createStatement();
        String sql= "select * from employees";
        ResultSet rs= stmt.executeQuery(sql);
        System.out.println(rs.next());
        connection.close();
    }
}
```

Mapper 사용하기 1st

- config.xml에 매퍼 설정을 추가한다.

```
<mappers>
        <mapper resource="mapper-emp.xml"/>
</mappers>
```

Mapper 사용하기 2nd

- 매퍼 파일을 생성한다.

```
<?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="emp">

        <select id="selectEmpListAll" resultType="emp.EmpTO">
                select employee_id empId,
                last_name lastName
                from employees
        </select>
</mapper>
```

Mapper 사용하기 3rd

- 리턴 타입에 사용할 TO를 생성한다.

```
package emp;

public class EmpTO {
		private int empId;
		private String lastName;
		public int getEmpId() {
			return empId;
		}
		public void setEmpId(int empId) {
			this.empId = empId;
		}
		public String getLastName() {
			return lastName;
		}
		public void setLastName(String lastName) {
			this.lastName = lastName;
		}
}
```

Mapper 사용하기 4th

- SqlSession을 다음과 같이 사용한다.

```
public class MBTest2 {
        public static void main(String[] args) throws Exception {
                String resource = "./mybatis-config.xml";
                InputStream inputStream =
Resources.getResourceAsStream(resource);
                SqlSessionFactory sqlSessionFactory =
                                        new
SqlSessionFactoryBuilder().build(inputStream);
                SqlSession session= sqlSessionFactory.openSession();
                List<EmpTO> list= session.selectList("emp.selectEmpListAll");
                System.out.println(list.size());
                session.close();
        }
}
```

Configuration File

- MyBatis Configuration 파일에는 다음과 같은 요소가 있다.

  - properties
  - settings
  - typeAliases
  - typeHandlers
  - objectFactory
  - plugins
  - environments
    - environment
      - transactionManager
      - dataSource
  - databaseIdProvider
  - mappers

Configuration File - Properties

- Properties는 다음처럼 설정한다.

```
<properties resource="org/mybatis/example/config.properties">
        <property name="username" value="hr"/>
        <property name="password" value="hr"/>
</properties>
```

- 사용방법은 다음과 같다.

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

Configuration File - settings

```
<settings>
	<setting name="cacheEnabled" value="true"/>
	<setting name="lazyLoadingEnabled" value="true"/>
	<setting name="multipleResultSetsEnabled" value="true"/>
	<setting name="useColumnLabel" value="true"/>
	<setting name="useGeneratedKeys" value="false"/>
	<setting name="autoMappingBehavior" value="PARTIAL"/>
	<setting name="autoMappingUnknownColumnBehavior" value="WARNING"/>
	<setting name="defaultExecutorType" value="SIMPLE"/>
	<setting name="defaultStatementTimeout" value="25"/>
	<setting name="defaultFetchSize" value="100"/>
	<setting name="safeRowBoundsEnabled" value="false"/>
	<setting name="mapUnderscoreToCamelCase" value="false"/>
	<setting name="localCacheScope" value="SESSION"/>
	<setting name="jdbcTypeForNull" value="OTHER"/>
	<setting name="lazyLoadTriggerMethods" value="equals,clone,hashCode,toString"/>
</settings>
```

Configuration File - typeAlias

- 자바타입의 간략화된 이름이다.

```
<typeAliases>
        <typeAlias alias="Author" type="domain.blog.Author"/>
        <typeAlias alias="Blog" type="domain.blog.Blog"/>
        <typeAlias alias="Comment" type="domain.blog.Comment"/>
        <typeAlias alias="Post" type="domain.blog.Post"/>
        <typeAlias alias="Section" type="domain.blog.Section"/>
        <typeAlias alias="Tag" type="domain.blog.Tag"/>
</typeAliases>
```

Configuration File - plugins

- 호출을 가로챌 수 있게 한다.

Executor (update, query, flushStatements, commit, rollback, getTransaction, close, isClosed)
- ParameterHandler (getParameterObject, setParameters)
- ResultSetHandler (handleResultSets, handleOutputParameters)
- StatementHandler (prepare, parameterize, batch, update, query)

Configuration File – plugins – Interceptor class

- 호출을 가로채어 실행될 수 있다.

```
// ExamplePlugin.java
@Intercepts({@Signature(
type= Executor.class,
method = "update",
args = {MappedStatement.class,Object.class})})
public class ExamplePlugin implements Interceptor {
        public Object intercept(Invocation invocation) throws Throwable {
                return invocation.proceed();
        }
        public Object plugin(Object target) {
                return Plugin.wrap(target, this);
        }
        public void setProperties(Properties properties) {
        }
}
```

Configuration File – plugins – config.xml

- config.xml 내부에 다음을 적용한다.

```
<!-- mybatis-config.xml -->
<plugins>
        <plugin interceptor="org.mybatis.example.ExamplePlugin">
                <property name="someProperty" value="100"/>
        </plugin>
</plugins>
```

Configuration File – environments

- DBMS와의 설정을 지정한다.

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

Configuration File – mappers

- DBMS와의 설정을 지정한다.

```
<!-- Using classpath relative resources -->
<mappers>
	<mapper resource="org/mybatis/builder/AuthorMapper.xml"/>
	<mapper resource="org/mybatis/builder/BlogMapper.xml"/>
	<mapper resource="org/mybatis/builder/PostMapper.xml"/>
</mappers>
```

```
<!-- Using url fully qualified paths -->
<mappers>
	<mapper url="file:///var/mappers/AuthorMapper.xml"/>
	<mapper url="file:///var/mappers/BlogMapper.xml"/>
	<mapper url="file:///var/mappers/PostMapper.xml"/>
</mappers>
```

```
<!-- Using mapper interface classes -->
<mappers>
	<mapper class="org.mybatis.builder.AuthorMapper"/>
	<mapper class="org.mybatis.builder.BlogMapper"/>
	<mapper class="org.mybatis.builder.PostMapper"/>
</mappers>
```

매퍼 파일에 대해

- SQL이나 OR 매핑을 XML파일로 기술한 파일이다.

- 하나의 애플리케이션에는 복수 개의 매퍼 파일이 만들어 진다.
- NameSpace
  - 매퍼 파일의 Root Element의 속성으로 지정한다.
  - 매퍼 파일에 기술된 SQL문장들을 그룹으로 묶는 역할을 한다.

- SQL 문장의 기술
  - select, update, insert, delete라는 이름의 태그를 사용한다.
  - <select id="sql문장의ID" parameterType="" resultType="">
  - id : SQL 문장의 식별자 역할을 수행한다.
  - parameterTpye : 전달될 매개변수를 지정한다.
  - resultType : 리턴타입을 지정한다.

매퍼 파일

- 매퍼 파일 내부엔 다음과 같은 요소를 넣는다.

• cache – Configuration of the cache for a given namespace.
• cache-ref – Reference to a cache configuration from another namespace.
• resultMap – The most complicated and powerful element that describes how to load your objects from the database result sets.
• parameterMap – Deprecated! Old-school way to map parameters. Inline parameters are preferred and this element may be removed in the future. Not documented here.
• sql – A reusable chunk of SQL that can be referenced by other statements.
• insert – A mapped INSERT statement.
• update – A mapped UPDATE statement.
• delete – A mapped DELETE statement.
• select – A mapped SELECT statement.

매퍼 파일 – SELECT

- 가장 일반적인 쿼리문장이다.

```
<select id="selectPerson" parameterType="int" resultType="hashmap">
    SELECT * FROM PERSON WHERE ID = #{id}
</select>
```

- jdbc로 코딩한 소스이다.

```
// Similar JDBC code, NOT MyBatis…
String selectPerson = "SELECT * FROM PERSON WHERE ID=?";
PreparedStatement ps = conn.prepareStatement(selectPerson);
ps.setInt(1,id);
```

매퍼 파일 – SELECT

```
<select
id="selectPerson"
parameterType="int"
parameterMap="deprecated"
resultType="hashmap"
resultMap="personResultMap"
flushCache="false"
useCache="true"
timeout="10000"
fetchSize="256"
statementType="PREPARED"
resultSetType="FORWARD_ONLY">
```

파라미터 바인딩

- parameterType에는 파라미터타입을 지정한다.

- 자바클래스를 지정할 수 있고, Map을 지정할 수 있다.
  - 자바로 지정하기
    - <insert id="insertPet" parameterType="PetBean"
      INSERT INTO T_PET(id, name)
      VALUES(#{pedId}, #{petName})
      </insert>
  - Map으로 지정하기
    - <insert id="insertPet" parameterType="hashMap"
      INSERT INTO T_PET(id, name)
      VALUES(#{pedId}, #{petName})
      </insert>

동적SQL

- where절을 다음과 같이 작성할 수 있다.

```
<select id="" parameterType="PetBean" resultType="PetBean">
    SELECT id, name
    FROM T_PET
    <where>
            <if test="name !=null">
                    PET_NAME= #{name}
            </if>
    </where>
</select>
```

SELECT문의 리턴 타입 매핑

- MyBatis3.x에서는 자바클래스로 매핑을 할 수 있다.
  - <select id="selectPetById" parameterType="int"
    resultType="PetBean">
    SELECT PET_ID as id, PET_NAME, name
    from T_PET
    WHERE PET_ID= #{id}
    </select>

- 호출 결과가 PetBean타입의 객체로 매핑된다.
  - PenBean bean= (PetBean)session.selectOne(
    "pet.selectPetById", id);
  - List<PetBean> list= session.selectList("pet.selectLPetList");

매퍼 파일 – insert

- insert는 다음처럼 사용한다.

```
<insert id="insertAuthor">
        insert into Author (id,username,password,email,bio)
        values (#{id},#{username},#{password},#{email},#{bio})
</insert>
```

매퍼 파일 – update

- update는 다음처럼 사용한다.

```
<update id="updateAuthor">
        update Author set
        username = #{username},
        password = #{password},
        email = #{email},
        bio = #{bio}
        where id = #{id}
</update>
```

매퍼 파일 – delete

- delete는 다음처럼 사용한다.

```
<delete id="deleteAuthor">
    delete from Author where id = #{id}
</delete>
```

매퍼 파일 – sql

- sql 문의 일부를 설정해두고 필요한 곳에 재사용가능하다.

```
<sql id="userColumns"> ${alias}.id,${alias}.username,${alias}.password </sql>
```

```
<select id="selectUsers" resultType="map">
        select
        <include refid="userColumns"><property name="alias" value="t1"/></include>,
        <include refid="userColumns"><property name="alias" value="t2"/></include>
        from some_table t1
        cross join some_table t2
</select>
```

매퍼 파일 – 문자열 대체 하기

- 기본적으로 #{name}으로 사용한다. PreparedStatement에서의 세팅과 기본적으로 동일하다.

- 그런데, 때로는 문자열 그대로를 사용해야할 경우가 있다.

```
ORDER BY ${columnName}
```

  - 그럴 때엔 ${}를 사용한다.

매퍼 파일 – ResultMap

- 다음처럼 사용할 수 있다.

```
<select id="selectUsers" resultType="map">
        select id, username, hashedPassword
        from some_table
        where id = #{id}
</select>
```

매퍼 파일 – ResultMap

- 다음처럼 사용할 수 있다.

```
package com.someapp.model;
public class User {
	private int id;
	private String username;
	private String hashedPassword;
	public int getId() {
		return id;
	}
	public void setId(int id) {
		this.id = id;
	}
	public String getUsername() {
		return username;
	}
	public void setUsername(String username) {
		this.username = username;
	}
	public String getHashedPassword() {
		return hashedPassword;
	}
	public void setHashedPassword(String hashedPassword) {
		this.hashedPassword = hashedPassword;
	}
}
```

매퍼 파일 – ResultMap

- 다음처럼 사용할 수 있다.

```
<select id="selectUsers" resultType="com.someapp.model.User">
        select id, username, hashedPassword
        from some_table
        where id = #{id}
</select>
```

```
!-- In Config XML file -->
<typeAlias type="com.someapp.model.User" alias="User"/>
<!-- In SQL Mapping XML file -->
<select id="selectUsers" resultType="User">
        select id, username, hashedPassword
        from some_table
        where id = #{id}
</select>
```

매퍼 파일 – ResultMap

- 다음처럼 사용할 수 있다.

```
<select id="selectUsers" resultType="User">
select
user_id as "id",
user_name as "userName",
hashed_password as "hashedPassword"
from some_table
where id = #{id}
</select>
```

```
<resultMap id="userResultMap" type="User">
<id property="id" column="user_id" />
<result property="username" column="user_name"/>
<result property="password" column="hashed_password"/>
</resultMap>
```

```
<select id="selectUsers" resultMap="userResultMap">
select user_id, user_name, hashed_password
from some_table
where id = #{id}
</select>
```

동적 SQL

- 동적 SQL에는 다음과 같은 요소가 있다.

  - if
  - choose (when, otherwise)
  - trim (where, set)
  - foreach

동적 SQL - if

- 동적 SQL에는 다음과 같은 요소가 있다.

```
<select id="findActiveBlogWithTitleLike“ resultType="Blog">
        SELECT * FROM BLOG
        WHERE state = ‘ACTIVE’
        <if test="title != null">
                AND title like #{title}
        </if>
</select>
```

```
<select id="findActiveBlogLike“ resultType="Blog">
        SELECT * FROM BLOG WHERE state = ‘ACTIVE’
        <if test="title != null">
        AND title like #{title}
        </if>
        <if test="author != null and author.name != null">
        AND author_name like #{author.name}
        </if>
</select>
```

동적 SQL - choose, when, otherwise

- 동적 SQL에는 다음과 같은 요소가 있다.

```
<select id="findActiveBlogLike“ resultType="Blog">
        SELECT * FROM BLOG WHERE state = ‘ACTIVE’
        <choose>
        <when test="title != null">
        AND title like #{title}
        </when>
        <when test="author != null and author.name != null">
        AND author_name like #{author.name}
        </when>
        <otherwise>
        AND featured = 1
        </otherwise>
        </choose>
</select>
```

동적 SQL - trim, where, set

- 동적 SQL에는 다음과 같은 요소가 있다.

```
<select id="findActiveBlogLike“ resultType="Blog">
        SELECT * FROM BLOG
        WHERE
        <if test="state != null">
        state = #{state}
        </if>
        <if test="title != null">
        AND title like #{title}
        </if>
        <if test="author != null and author.name != null">
        AND author_name like #{author.name}
        </if>
</select>
```

```
SELECT * FROM BLOG
WHERE
```

```
SELECT * FROM BLOG
WHERE
AND title like ‘someTitle’
```

동적 SQL - trim, where, set

- 동적 SQL에는 다음과 같은 요소가 있다.

```
<select id="findActiveBlogLike“ resultType="Blog">
        SELECT * FROM BLOG
        <where>
        <if test="state != null">
        state = #{state}
        </if>
        <if test="title != null">
        AND title like #{title}
        </if>
        <if test="author != null and author.name != null">
        AND author_name like #{author.name}
        </if>
        </where>
</select>
```

동적 SQL - trim, where, set

- 동적 SQL에는 다음과 같은 요소가 있다.

```
<trim prefix="WHERE" prefixOverrides="AND |OR ">
…
</trim>
```

```
<update id="updateAuthorIfNecessary">
update Author
<set>
<if test="username != null">username=#{username},</if>
<if test="password != null">password=#{password},</if>
<if test="email != null">email=#{email},</if>
<if test="bio != null">bio=#{bio}</if>
</set>
where id=#{id}
</update>
```

```
<trim prefix="SET" suffixOverrides=",">
…
</trim>
```

동적 SQL - foreach

- 동적 SQL에는 다음과 같은 요소가 있다.

```
<select id="selectPostIn" resultType="domain.blog.Post">
        SELECT *
        FROM POST P
        WHERE ID in
        <foreach item="item" index="index" collection="list"
                open="(" separator="," close=")">
                #{item}
        </foreach>
</select>
```

스프링과의 연계 1st

- MyBatis3.x의 MyBatis-Spring이라는 라이브러리를 이용하여 Spring과 연계한다.
- SqlSessionFactory와 SqlSession 및 DAO를 Sping Component로 생성하여 와이어링(Wiring: DI Injection)하여 사용한다.

- SqlSessionFactory를 빈(Bean)으로 정의하기
  - <bean id="sqlSessionFactory"
    class="org.mybatis.spring.SqlSessionFactory>
    <property name="dataSource" ref="dataSource"/>
    <property name="mapperLocation">
    <list>
    <value>파일위치/이름</value>
    </list>
    </property>
    </bean>

스프링과의 연계 2nd

- SqlSession 인젝션

- SqlSessionTemplate으로 인젝션한다.
  - <bean id="sqlSessionTemplate"
    class="org.mtbatis.spring.SqlSessionTemplate">
    <constructor-arg ref="sqlSessionFactory">
    </bean>

스프링과의 연계 3rd

- SqlSession을 인젝션하여 사용한다.

- PetDAO 클래스 내부에서

```
@Repository
public class PetDAO {
    @Autowired
    SqlSession session;

    public PetBean selectPetById(int id) {
            return session.selectOne("pet.selectPetById", id);
    }
}
```

참고 – 전자정부 프레임워크 데이터 레이어

- 데이터처리 레이어는 Spring, iBatis, MyBatis, Hibernate 등 총 4종의 오픈소스 SW를 사용하고 있다.

서비스	오픈소스 SW	버전
DataSource	Spring	3.2.9
Data Access	iBatis SQL Maps	2.3.4
	MyBatis	3.2.7
ORM	Hibernate	4.3.5
Transaction	Spring	3.2.9

참고 – 전자정부 프레임워크 Data Access 개요

- JDBC 를 사용한 Data Access를 추상화하여 간편하고 쉽게 사용할 수 있는 Data Mapper framework 인 iBATIS 를 Data Access 기능의 기반 오픈 소스로 채택
- iBATIS 를 사용하면 관계형 데이터베이스에 엑세스하기 위해 필요한 일련의 자바 코드 사용을 현저히 줄일 수 있으며 간단한 XML 기술을 사용하여 SQL 문을 JavaBeans (또는 Map) 에 간편하게 매핑할 수 있음
- **Data Access 서비스는 다양한 데이터베이스 솔루션 및 데이터베이스 접근 기술에 일관된 방식으로 대응하기 위한 서비스**
  - 데이터를 조회하거나 입력, 수정, 삭제하는 기능을 수행하는 메커니즘을 단순화함
  - 데이터베이스 솔루션이나 접근 기술이 변경될 경우에도 데이터를 다루는 시스템 영역의 변경을 최소화할 수 있도록 데이터베이스와의 접점을 추상화함
  - 추상화된 데이터 접근 방식을 템플릿(Template)으로 제공함으로써, 개발자들의 업무 효율을 향상시킴.

참고 – 전자정부 프레임워크 iBatis 개요

- iBatis는 단순성이라는 사상을 강조한 퍼시스턴스 프레임워크로, SQL 맵을 이용하여 반복적이고 복잡한 DB 작업 코드를 최소화 한다.
- 단순성이라는 사상을 강조하여, XML을 이용하여 Stored Procedure 혹은 SQL 문과 자바 객체간의 매핑을 지원한다.
- 2001년 Clinton Begin (Apache 소프트웨어 재단)에 의해 개발된 퍼시스턴스 프레임워크이다.

항목	iBatis	Hibernate	비고
응답 지연 시간 (Round Trip Delay Time)	짧다	길다	Hibernate의 경우, 쿼리 자동생성 등의 기능으로 인해 다소 시간 소요 길다.
유연성 (Flexibility)	우수	미흡	
학습 곡선 (Learning Curve)	작다	크다	iBatis가 보다 JDBC와 유사하기 때문
SQL 지식	높아야 함	별로 필요치 않음	

* “Performance Comparison of Persistence Frameworks,” Sabu M. Thampi, Ashwin a K. (2007)

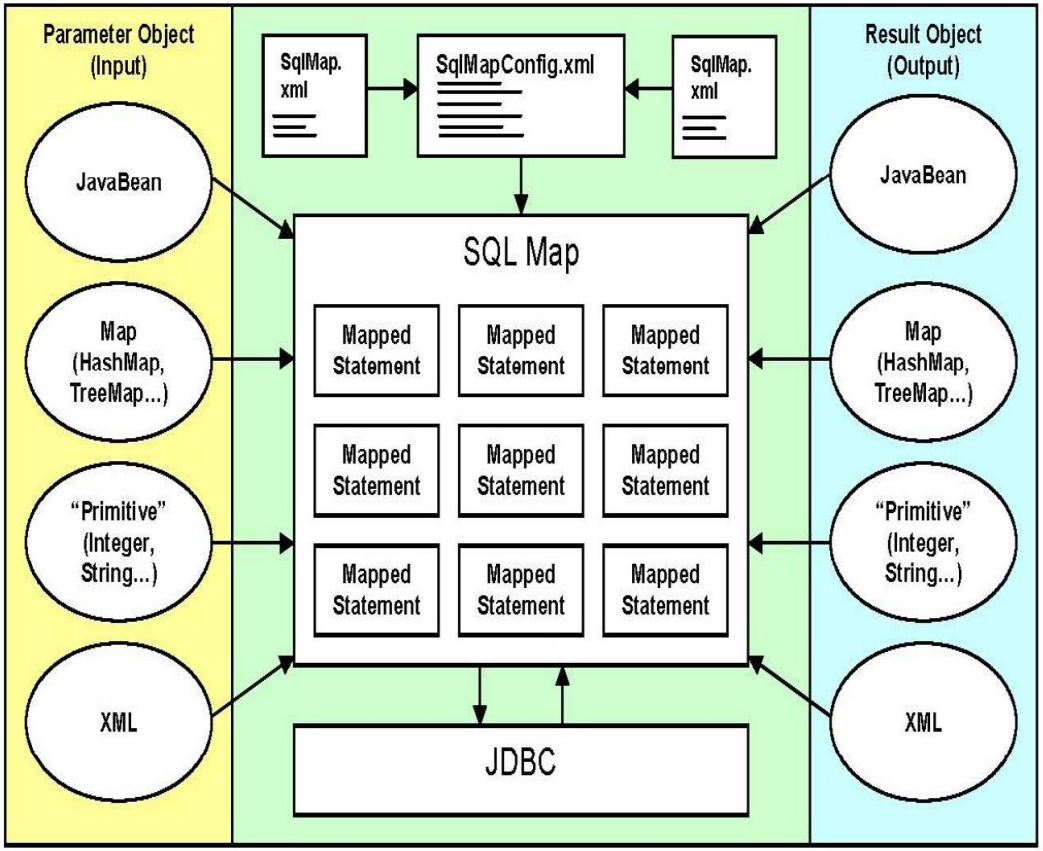
참고 – 전자정부 프레임워크 iBatis 개요 (2/4)

- iBatis는 소스코드 외부에 정의된 SQL문과 설정 정보를 바탕으로, 객체와 테이블 간의 매핑 기능 제공한다.
- iBATIS Data Mapper API 는 XML을 사용하여 SQL문과 객체 매핑 정보를 간편하게 기술할 수 있도록 지원한다.
- 자바 빈즈 객체와 Map 구현체, 다양한 원시 래퍼 타입(String, Integer..) 등을 PreparedStatement 의 파라미터나 ResultSet에 대한 결과로 쉽게 매핑하게 한다.

구성요소	설명
SqlMapConfig XML File	- iBatis 동작을 위한 DataSource, Data Mppaer 및 Thread Management 등과 같은 공통 설정
SqlMap XML File	- XML 방식으로 실행할 SQL문과 매핑 정보를 설정
SQL Map	- iBatis는 PreparedStatement 인스턴스를 생성하고, 제공된 파라미터 객체를 사용해서 파라미터를 셋팅한 후, Statement를 실행하고 ResultSet으로부터 결과 객체를 생성
Mapped Statement	- Parameter 객체와 Result 객체를 이용하여 SQL Statement로 치환
Parameter Object	- 파마리터 객체는 JavaBean, Map, Primitive 객체로서, 입력값을 셋팅하기 위해 사용되는 객체
Result Object	- 결과 객체는 JavaBean, Map, Primitive 객체로서, 쿼리문의 결과값을 담는 객체

참고 – 전자정부 프레임워크 iBatis 개요 (3/4)

- iBatis는 소스코드 외부에 정의된 SQL문과 설정 정보를 바탕으로, 객체와 테이블 간의 매핑 기능 제공한다.

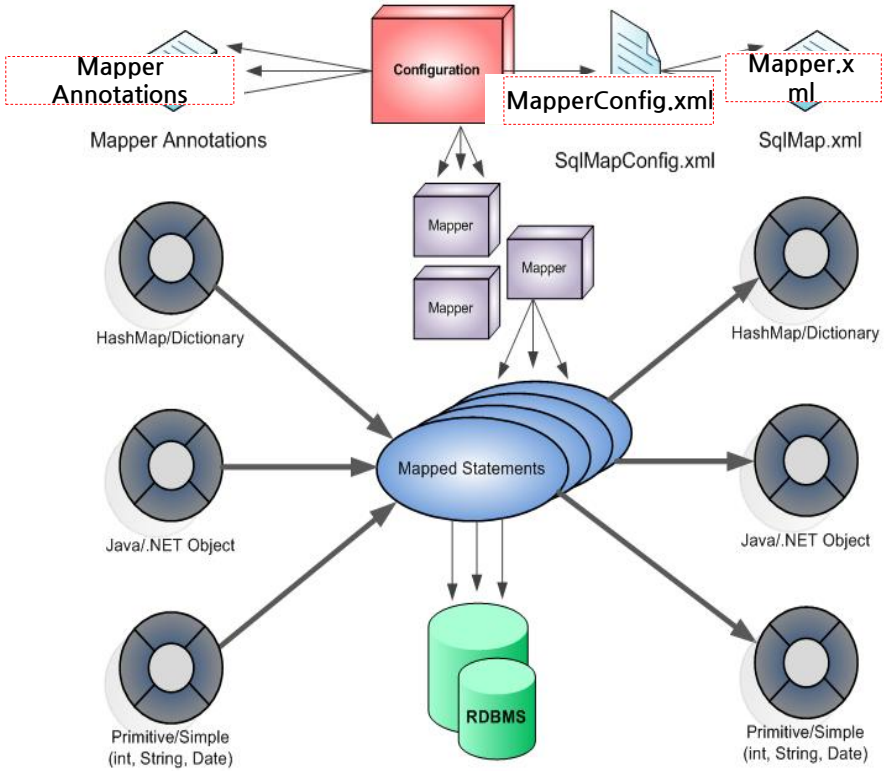


참고 – 전자정부 프레임워크 iBatis 개요 (4/4)

- 추상화된 접근 방식 제공
  - JDBC 데이터 엑세스에 대한 추상화된 접근 방식으로, 간편하고 쉬운 API, 자원 연결/해제, 공통 에러 처리 등을 통합 지원함
- 자바 코드로부터 SQL문 분리
  - 소스코드로부터 SQL문을 분리하여 별도의 repository(의미있는 문법의 XML)에 유지하고 이에 대한 빠른 참조구조를 내부적으로 구현하여 관리/유지보수/튜닝의 용이성을 보장함.
- SQL문 자동 실행, 입/출력 파라미터 자동 바인딩 지원
  - 쿼리문의 입력 파라메터에 대한 바인딩과 실행결과 resultset 의 가공(맵핑) 처리시 객체(VO, Map, List) 수준의 자동화를 지원함
- Dynamic SQL 지원
  - 코드 작성, API 직접 사용없이 입력 조건에 따른 동적인 쿼리문 변경을 지원함
- 다양한 DB 처리 지원
  - 기본 질의 외에 Batch SQL, Paging, Callable Statement, BLOB/CLOB 등 다양한 DB처리를 지원함

참고 – 전자정부 프레임워크 MyBatis 개요 (1/4)

- MyBatis 데이터 매퍼 서비스
  - 개발자가 작성한 SQL문 혹은 저장프로시저 결과값을 자바 오브젝트에 자동 매핑하는 서비스, 수동적인 JDBC 방식의 데이터 처리 작업 코드와는 달리 쿼리결과와 오브젝트 간 자동 매핑
  - SQL문과 저장프로시저는 XML 혹은 어노테이션 방식으로 작성 가능



참고 – 전자정부 프레임워크 MyBatis 개요 (1/4)

- 구성 요소와 설명

구성요소	설명
MapperConfig ML File	- MyBatis 동작을 위한 기본적인 설정을 공통으로 정의
Mapper XML File	- 실행할 SQL문 및 매핑 정보를 XML 방식 으로 정의
Mapper Annotations	- 자바 코드 내에서 실행할 SQL문 및 매핑 정보를 어노테이션을 이용하여 정의
Parameter Object	- SQL문의 조건절에서 값을 비교하거나 INSERT, UPDATE절 등에서 필요한 입력값을 받아오기 위한 오브젝트
Result Object	- 쿼리 결과를 담아 리턴하기 위한 오브젝트

참고 – 전자정부 프레임워크 MyBatis 개요 (2/4)

- 주요 변경 사항
  - iBatis의 SqlMapClient → SqlSession으로 변경
    - SqlSession 인터페이스
      - MyBatis를 사용하기 위한 기본적인 인터페이스로, SQL 문 처리를 위한 메서드를 제공
      - 구문 실행 메서드, 트랜잭션 제어 메서드 등 포함
      - - selectList(), selectOne(), insert(), update(), delete(), commit(), rollback(), …
      - SqlSessionFactory 클래스를 통해 MyBatis Configuration 정보에 해당 SqlSession 인스턴스를 생성

참고 – 전자정부 프레임워크 MyBatis 개요 (2/4)

- 주요 변경 사항
  - 어노테이션 방식 설정 도입
    - MyBatis는 본래 XML 기반의 프레임워크였으나, Mybatis 3.x 부터 어노테이션 방식의 설정을 지원
    - Mapper XML File 내 SQL문 및 매핑 정보를, 자바 코드 내에서 어노테이션으로 그대로 적용 가능
  - iBatis의 RowHandler → ResultHandler 변경
    - ResultHandler 인터페이스
      - Result Object에 담겨 리턴된 쿼리 결과를 핸들링할 수 있도록 메서드 제공
      - 사용 예시) 대량의 데이터 처리 시, 처리 결과를 File로 출력하고자 할 때 혹은 Result Object의 형태를 Map 형태로 가져올 때

참고 – 전자정부 프레임워크 MyBatis 개요 (3/4)

변경 또는 추가사항			iBatis 사용 시	MyBatis 사용 시
소스	패키지	변경	com.ibatis.*	org.apache.ibatis.*
	클래스	변경	SqlMapClient	SqlSession
		변경	SqlMapClientFactory	SqlSessionFactory
		변경	RowHandler	ResultHandler
Configuration XML File	DTD	변경	<!DOCTYPE sqlMapConfig PUBLIC "//iBATIS.com//DTD SQL Map Config 2.0//EN" "http://www.ibatis.com/dtd/sql-map-config-2.dtd">	<!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN" "http://mybatis.org/dtd/mybatis-3-config.dtd">
	요소와 속성	변경	<sqlMapConfig>	<configuration>
		변경	<settings ... />	<settings> <setting /> </settings>
		변경	<typeHandler callback="..." />	<typeHandlers> <typeHandler handler="..." /> </typeHandlers>
		추가	<transactionManager type="..."> <dataSource type="..." /> </transactionManager>	<environment id="..."> <transactionManager type="..." /> <dataSource type="..." /> </environment>
		추가		<typeAliases> <typeAlias alias="..." type="..." /> </typeAliases>
		변경	<sqlMap resource="..." />	<mappers> <mapper resource="..." /> </mappers>

참고 – 전자정부 프레임워크 MyBatis 개요 (4/4)

- Migrating from iBatis (2/2)

변경 또는 추가사항			iBatis 사용 시	MyBatis 사용 시
Mapper XML File	DTD	변경	<!DOCTYPE sqlMap PUBLIC "-//iBATIS.com//DTD SQL Map 2.0//EN" "http://www.ibatis.com/dtd/sql-map-2.dtd">	<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
	요소와 속성	변경	<sqlMap namespace="…">	<mapper namespace="…">
		변경	<cacheModel />	<cache />
		변경	-- SQL Statement <select id="…" resultClass | resultMap="…"> SELECT * FROM EMP </select> <insert id="…" parameterClass | parameterMap="…" > INSERT INTO EMP VALUES (#empNo#, #empName#); </insert>	<select id="…" resultType | resultMap="…"> SELECT * FROM EMP </select> <insert id="…" parameterType | parameterMap="…"> INSERT INTO EMP VALUES (#{empNo}, #{empName}); </insert>
		변경/추가	<dynamic> <isEqual /> <isNotNull /> … </dynamic>	<if test="…" /> <choose><when />…<otherwise /> <trim prefixOverrides | suffixOverrides="…" /> <foreach collection="…" item="…" />
스프링연동	빈생성	변경	<bean id="sqlMapClient" class="org.springframework.orm.ibatis.SqlMapClientFactoryBean">	<bean id="sqlSession" class="org.mybatis.spring.SqlSessionFactoryBean">
쿼리 호출	파라미터	추가	-- Statement ID로 실행할 SQL문을 호출 List list = …selectList(queryId, parameterObject);	-- Statement ID로 실행할 SQL문을 호출 List list = selectList(queryId, parameterObject); -- 메서드명으로 실행할 쿼리호출 List list = …selectList(parameterObject);

참고 – 전자정부 프레임워크 MyBatis 개발 가이드(1/2)

- MyBatis를 활용한 Persistence Layer 개발
  - 1) [MyBatis 설정 1] SQL Mapper XML 파일 작성 설정
    - 실행할 SQL문과 관련 정보 설정
    - SELECT/INSERT/UPDATE/DELETE, Parameter/Result Object, Dynamic SQL 등
  - 2) [MyBatis 설정 2] MyBatis Configuration XML 파일 작성
    - MyBatis 동작에 필요한 옵션을 설정
    - <mapper>: SQL Mapper XML 파일의 위치
  - 3) [스프링연동 설정] SqlSessionFactoryBean 정의
    - Spring와 MyBatis 연동을 위한 설정
    - 역할) MyBatis 관련 메서드 실행을 위한 SqlSession 객체를 생성
    - dataSource, configLocation, mapperLocations 속성 설정

참고 – 전자정부 프레임워크 MyBatis 개발 가이드(2/2)

- MyBatis를 활용한 Persistence Layer 개발
  - 4) DAO 클래스 작성
    - 방법1) SqlSessionDaoSupport를 상속하는 EgovAbstractMapper 클래스를 상속받아 확장/구현
      - 실행할 SQL문을 호출하기 위한 메서드 구현: SQL Mapping XML 내에 정의한 각 Statement id를 매개변수로 전달
    - 방법2) DAO 클래스를 Interface로 작성하고, 각 Statement id와 메서드명을 동일하게 작성 (Mapper Interface 방식)
      - Annotation을 이용한 SQL문 작성 가능
      - 메서드명을 Statement id로 사용하기 때문에, 코드 최소화 가능

참고 – 전자정부 프레임워크 ORM - 개요

- 서비스 개요
  - 객체 모델과 관계형 데이터베이스 간의 매핑 기능인 ORM(Object-Relational Mapping) 기능을 제공함으로써, SQL이 아닌 객체를 이용한 업무 로직의 작성이 가능하도록 지원
- 주요 기능
  - 객체와 관계형 데이터베이스 테이블 간의 매핑
    - 프레임워크 설정정보에 저장된 ORM 매핑정보를 이용하여 객체와 관계형 데이터베이스 테이블간의 매핑 지원
  - 객체 로딩
    - 객체와 매핑되는 관계형 데이터베이스의 값을 읽어와 객체의 속성 값으로 설정함

참고 – 전자정부 프레임워크 ORM - 개요

- 주요 기능 cont.
  - 객체 저장
    - 저장하고자 하는 객체의 속성 값을 객체와 매핑되는 관계형 데이터베이스에 저장
  - 다양한 연관 관계 지원
    - 객체와 객체 간의 1:1, 1:n, n:n 등의 다양한 연관 관계를 지원
    - 객체의 로딩 및 저장 시, 연관 관계를 맺고 있는 객체도 로딩 및 저장 지원
  - Caching : 객체에 대한 Cache 기능을 지원하여 성능을 향상시킴