MyBatis

목차

KH정보교육원

MyBatis의 개념 및 흐름

MyBatis

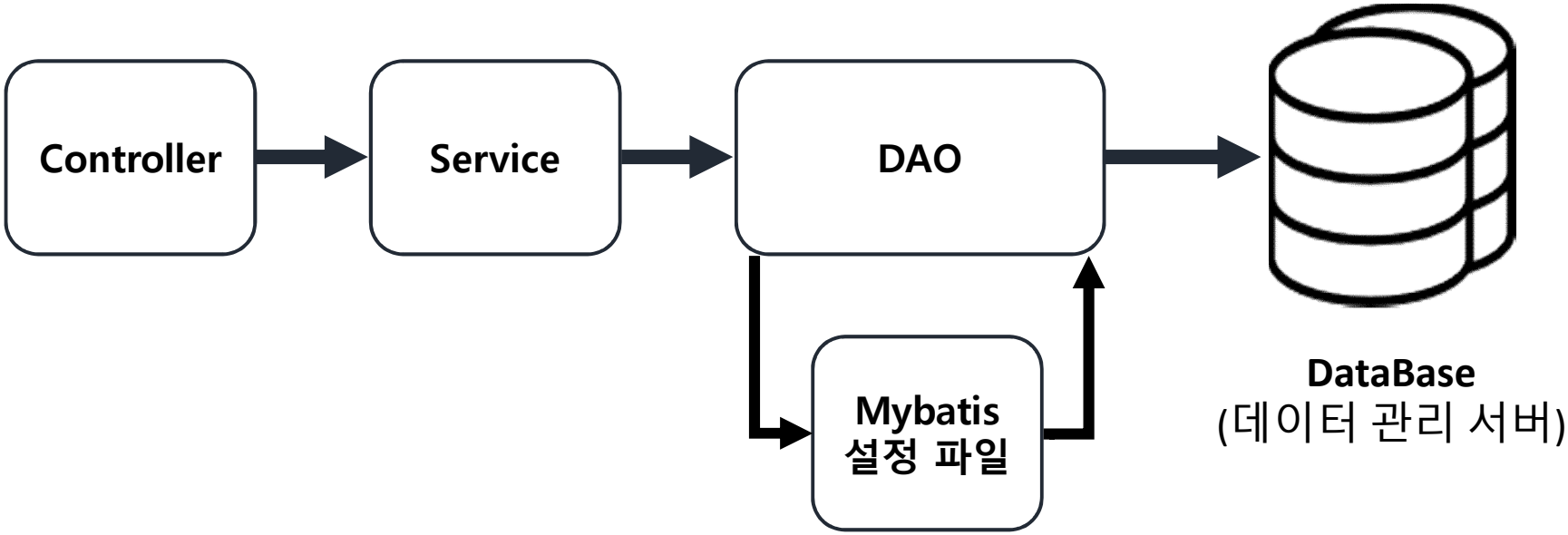
데이터의 입력, 조회, 수정, 삭제(CRUD)를 보다 편하게 하기 위해 xml로
구조화한 Mapper설정파일을 통해 JDBC를 구현한 영속성 프레임 워크
기존 JDBC를 통해 구현했던 상당 부분의 코드와 파라미터 설정 및
결과 매핑을 xml설정을 통해 쉽게 구현할 수 있게 함

MyBatis API Site : http://www.mybatis.org/mybatis-3/ko

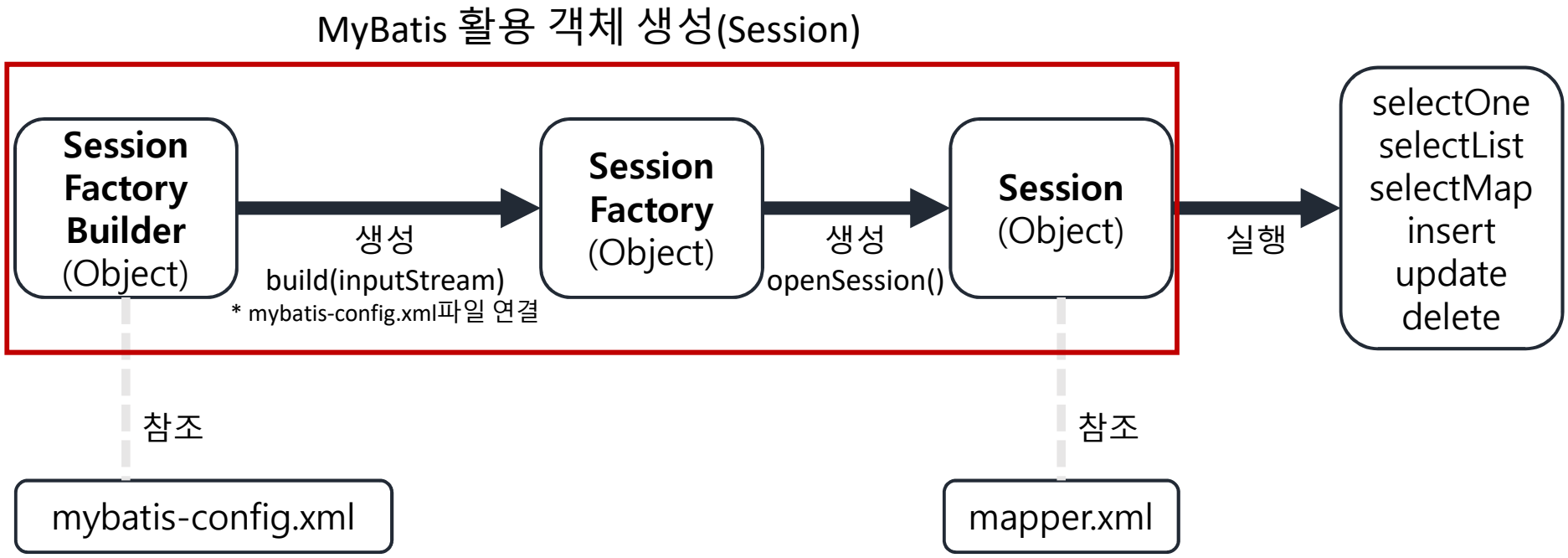
▶ MyBatis 흐름

이전 JDBC Template을 통해 SQL을 실행하였다면

MyBatis는 해당 흐름을 전용 라이브러리를 통해 대체하여 동작



▶ MyBatis 동작 구조



- Class의 Alias(별칭) 설정
- DB 연결 설정
- sql 구문 경로 설정

- sql 쿼리문 설정
  (인자 값, 결과 값, 데이터 타입 등 설정)

* 각 패키지마다 존재 (게시판, 멤버 패키지 등)

* 위의 사용 객체들은 mybatis-x.x.x.jar 파일에 존재

MyBatis 라이브러리

✓ 다운로드

https://blog.mybatis.org/ 접속

Products 탭 선택 하여 MyBatis3의 download 클릭

연결되는 Github에서 아래 Assets의 mybatis-x.x.x.zip 다운로드

▶ MyBatis 라이브러리

✓ 연동

압축 해제 후 mybatis-x.x.x.jar 라이브러리를 프로젝트 내
'WEB-INF/lib/'경로에 추가

▶ MyBatis 참고

Apache project에서 ibatis를 운영하던 팀이 2010년 5월 9일에 Google팀으로 이동하면서 MyBatis로 이름을 바꿈

MyBatis는 기존 ibatis의 한계점이었던 동적 쿼리와 어노테이션 처리를 보강하여 더 나은 기능 제공

ibatis는 현재 비활성화 상태이며 기존에 ibatis로 만들어진 애플리케이션의 지원을 위해 라이브러리만 제공하고 있음

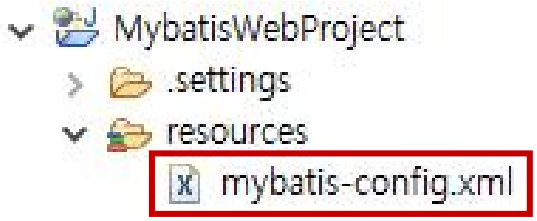
✓ 차이점

	ibatis	MyBatis
Java 요구 버전	JDK 1.4 이상	JDK 1.5이상
패키지 구조	com.ibatis.*	org.apache.ibatis.*
사용 용어	SqlMapConfig sqlMap resultClass	Configuration Mapper resultType
동적쿼리/어노테이션	X	O

mybatis-config 설정하기

▶ mybatis-config.xml 생성 위치

'resources'라는 Source Folder를 생성하고 mybatis-config.xml 파일 등록




```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="${driver}"/>
        <property name="url" value="${url}"/>
        <property name="username" value="${username}"/>
        <property name="password" value="${password}"/>
      </dataSource>
    </environment>
  </environments>
  <mappers>
    <mapper resource="org/mybatis/example/BlogMapper.xml"/>
  </mappers>
</configuration>
```

▶ mybatis-config.xml 작성

KH정보교육원

mybatis-config.xml 최상단에 아래와 같이 xml형식을 지정하여
이하의 설정 내용이 mybatis config 설정임을 선언

```
<?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>최상위 태그를 작성하고 내부에 필요한 설정 작성

```
<configuration>

  <!-- 내부에 필요한 설정들 작성 -->

</configuration>
```

▶ mybatis-config.xml 작성

<properties>태그 : 외부 properties파일의 내용을 불러 올 때 사용

예시 :

```
<properties resource="경로+ 파일명.properties">
    <!-- properties 파일에 값 설정 가능 -->
    <property name="key명" value="설정 값"/>
</properties>
```

<properties> 설정 값 활용

```
<dataSource type="POOLED">
    <property name="명칭" value="${ properties에 설정된 key명 }"/>
    <property name="명칭" value="${ properties에 설정된 key명 }"/>
</dataSource>
```

▶ mybatis-config.xml 작성

<settings>태그 : MyBatis구동 시 선언할 설정 작성

예시 :

```
<settings>
    <!-- Null 값이 발생할 경우 빈칸이 아닌 null로 인식해라 -->
    <setting name="jdbcTypeForNull" value="NULL"/>
</settings>
```

* 속성값 참조 : http://www.mybatis.org/mybatis-3/ko/configuration.html

<typeAliases> 태그 : MyBatis에서 사용할 자료형의 별칭 선언

예시 :

```
<typeAliases>
    <!-- type에는 패키지 명까지 전부 기술해줘야 된다. -->
    <typeAlias type="com.kh.member.model.vo.Member" alias="Member"/>
</typeAliases>
```

▶ mybatis-config.xml 작성

<environments>태그 : MyBatis에서 연동할 DataBase 정보 등록

예시 :

```
<environments default="development">
    <!-- environment id를 구분하여 연결할 DB를 여러 개 구성할 수도 있다. -->
    <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="mybatis"/>
            <property name="password" value="mybatis"/>
        </dataSource>
    </environment>
</environments>
```

* 여러 개의 DB를 등록하여 사용할 수 있음

  build()메소드 구현 시 매개변수에 environment의 id를 설정하면 됨

▶ mybatis-config.xml 작성

<mappers>태그 : 사용하고자 하는 쿼리문이 정의된 mapper파일 등록

예시 :

```
<mappers>
    <mapper resource="resources/mappers/member-mapper.xml"/>
    <mapper resource="resources/mappers/board-mapper.xml"/>
</mappers>
```

▶ MyBatis 내장 별칭

KH정보교육원

Mybatis 타입	Java 자료형	Mybatis 타입	Java 자료형
_byte	byte	double	Double
_long	long	float	Float
_short	short	boolean	Boolean
_int / _integer	int	date	Date
_double	double	object	Object
_float	float	map	Map
_boolean	boolean	hashmap	HashMap
string	String	list	List
byte	Byte	arraylist	ArrayList
long	Long	collection	Collection
short	Short	iterator	Iterator
int / integer	Integer		

▶ POOLED와 UNPOOLED 차이점

KH정보교육원

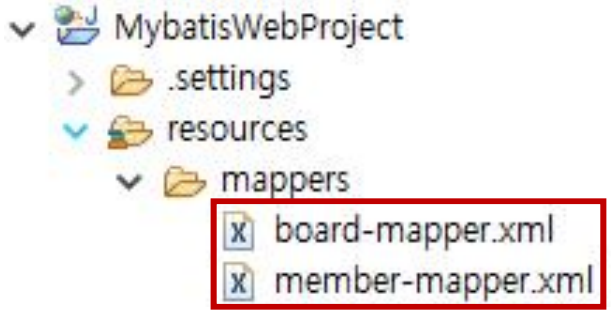
구분	POOLED	UNPOOLED
특징	최초 Connection객체를 생성할 때 해당 정보를 pool영역에 저장해두고 이후 Connection객체를 생성할 때 이를 재 사용함	Connection객체를 별도로 저장하지 않고 객체 호출 시 매번 생성하여 사용
장점	Connection객체를 생성하여 DataBase와 연결을 구축하는데 걸리는 시간이 단축됨	Connection연결이 많지 않은 코드를 작성할 때 간단하게 구현 가능
단점	단순한 로직을 수행하는 객체를 만들기에는 설정해야 할 정보가 많음	매번 새로운 Connection 객체를 생성하므로 속도가 상대적으로 느림

* 설정 가능한 type 중 JNDI도 있는데 이는 MyBatis에서 Connection객체를 생성하여 관리하지 않고 Web Application의 설정을 따르겠다는 의미

mapper 설정하기

▶ *-mapper.xml 생성 위치

'resources' 폴더 안에 'mappers'폴더 생성 후 그 안에 식별하기 쉬운 이름을 지어 파일 등록



▶ *-mapper.xml 작성

xml 최상단에 아래와 같이 xml형식을 지정하여 이하의 설정 내용이 MyBatis mapper 설정임을 선언

```
<?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" >
```

이어서 <mapper>태그를 작성하고 외부에서 접근할 수 있는 이름인 namespace속성 기입, 이후 작성될 태그들은 <mapper>태그 안에 기록

```
<mapper namespace="memberMapper">

    <!-- mapper 내부에 작성될 내용 -->

</mapper>
```

▶ *-mapper.xml 작성

<resultMap>태그 : 조회한 결과를 객체와 Row간의 1:1 매칭이 아닌
원하는 객체의 필드에 담아 반환하고자 할 때 사용

예시 :

```
<resultMap type="Member" id="memberResultSet">
    <!-- property = 자바의 필드변수 이름 / column = DB의 해당 컬럼 -->
    <!-- id는 primary key / result는 일반 컬럼 -->
    <id property="mid" column="MID"/>
    <result property="userId" column="USER_ID"/>
    <result property="userPwd" column="USER_PWD"/>
    <result property="userName" column="USER_NAME"/>
</resultMap>
```

* <resultMap>의 type속성은 실제로 구현해 놓은 자바 POJO객체를 사용해야 하며, mybatis-config.xml에서 typeAlias를 지정하지 않은 경우 패키지 명부터 클래스 명까지 모두 기술해야 함

▶ *-mapper.xml 작성

<select>태그 : SQL의 조회구문을 작성 할 때 사용되는 태그, 해당 쿼리를 외부에서 접근하고자 할 때 namespace.id명을 적어 접근가능

예시 :

```
<select id="memberInfo" parameterType="string" resultType="_int">
    <!-- #{field}는 pstmt의 '?'의 역할이며, 전달된 값을 뜻함
         또한 여러줄로 줄바꿈문자를 섞어 사용도 가능하다.
         단, 쿼리의 마지막을 알리는 세미콜론은 에러를 유발한다. -->
    SELECT *
    FROM MEMBER
    WHERE USER_ID = #{userId}
</select>
```

▶ <select>태그 주요 속성

KH정보교육원

속성 명	내용
id	구문을 찾기 위해 사용될 수 있는 네임스페이스 내 유일한 구분자
parameterType	구문에 전달될 파라미터의 클래스 명(패키지 경로 포함)이나 별칭
resultType	리턴되는 타입의 패키지 경로를 포함한 전체 클래스 명이나 별칭, collection인 경우 list, arraylist로 설정 가능
resultMap	리턴되는 타입의 필드 명이 다를 때 사용하며 직접 이름을 지정하여 매칭

* resultMap과 resultType은 둘 모두를 사용할 수 없으며 둘 중 하나만 선언해야 함

▶ <select>태그 주요 속성

KH정보교육원

속성 명	내용
flushCache	기본 값은 false이며 이 값을 true로 설정하면 구문이 호출 될 때마다 로컬, 2nd 레벨 캐시가 지워짐
useCache	기본 값은 true이며 이 값을 true로 설정하면 구문의 결과가 2nd 레벨 캐시에 저장
timeout	예외가 발생하기 전에 데이터베이스의 요청 결과를 기다리는 최대 시간 설정 드라이버에 따라 다소 지원되지 않을 수 있음
statementType	STATEMENT, PREPARED 또는 CALLABLE 중 하나 선택 가능(기본 값 : PREPARED) MyBatis에게 Statement, PreparedStatement, CallableStatement를 사용하게 함

FlushCache와 useCache

일반적으로 쿼리를 수행할 때 쿼리의 결과나 호출되는 내용이
변동이 없는 정적인 쿼리나 결과라면
이를 매 반복 시마다 굳이 새로운 쿼리로 생성하여 호출하거나
새로운 결과를 받아올 필요가 없을 것임
이러한 상황을 위해 MyBatis에서는 Cache라는 저장소를 내장하여
반복되는 정적인 쿼리의 호출이나 결과에 대한 내용을
한 번 이상 실행할 경우 이를 미리 저장해두어
재 호출에 소요되는 시간을 절약할 수 있게 도와줌

▶ *-mapper.xml 작성

<insert>, <update>, <delete> 태그 : 해당 태그들은 설정 동일

예시 :

```
<insert id="insertMember" parameterType="Member" flushCache="true"
        statementType="PREPARED", useGeneratedKeys="true" timeout="20">

    INSERT INTO MEMBER VALUES(
        #{userId}, #{userPwd}, #{userName}
    )

</insert>
```

파라미터로 객체를 받는 경우 해당 객체의 필드 값을
'변수명 = 값'의 Map 방식으로 조회하여 가져올 수 있다.

▶ *-mapper.xml 작성

<insert>, <update>, <delete> 태그 : 해당 태그들은 설정 동일

예시 :

```
<update id="updateMember" parameterType="Member" flushCache="true"
        statementType="PREPARED" timeout="20">
    UPDATE MEMBER
    SET USER_PWD = #{userPwd}, USER_NAME = #{userName}
    WHERE USER_ID = #{userId}
</update>
```

```
<delete id="deleteMember" parameterType="string" flushCache="true"
        statementType="PREPARED" timeout="20">
    DELETE FROM MEMBER
    WHERE USER_ID = #{userId}
</delete>
```

▶ <insert>,<update>,<delete>태그 주요 속성

속성 명	내용
id	구문을 찾기 위해 사용될 수 있는 네임스페이스 내 유일한 구분자
parameterType	구문에 전달될 파라미터의 클래스 명(패키지 경로 포함)이나 별칭
flushCache	기본 값은 false이며 이 값을 true로 설정하면 구문이 호출 될 때마다 캐시가 지워짐
timeout	예외가 발생하기 전에 데이터베이스의 요청 결과를 기다리는 최대 시간 설정 드라이버에 따라 다소 지원되지 않을 수 있음
userGeneratedKeys	(insert, update에만 적용) 데이터베이스에서 내부적으로 생성한 키(예를 들어 MySQL 또는 SQL Server의 자동 증가 필드)를 받는 JDBC getGeneratedKeys 메소드를 사용하도록 설정 (기본 값 : false)
keyProperty	(insert, update에만 적용) getGeneratedKeys 메소드나 insert 구문의 selectKey 태그의 설정 select 문의 결과를 저장할 프로퍼티를 지정, 디폴트는 세팅하지 않는 것 여러 개의 컬럼을 사용한다면 프로퍼티 명에 콤마를 구분자로 나열

KH정보교육원

MyBatis 활용하기

▶ SqlSession 생성하기

✓ 싱글톤을 적용한 Template 클래스 생성

mybatis-config.xml, *-mapper.xml파일 생성을 완료했다면
common패키지를 만들어 싱글톤을 적용한 Template클래스를 만들고
SqlSession을 반환해주는 static메소드 작성

```
public class Template {

    public static SqlSession getSqlSession() {
        SqlSession session = null;

        String resource = "/mybatis-config.xml";

        try {
            InputStream stream = Resources.getResourceAsStream(resource);

            SqlSessionFactoryBuilder builder = new SqlSessionFactoryBuilder();
            SqlSessionFactory factory = builder.build(stream);

            session = factory.openSession(false);

        } catch (IOException e) {
            e.printStackTrace();
        }

        return session;
    }
}
```

1. mybatis-config.xml의 설정 정보를 InputStream 객체를 통해 읽어옴
2. SqlSessionFactoryBuilder 객체를 생성하고 build() 메소드를 통해 SqlSessionFactory 객체를 생성
3. SqlSessionFactory 객체의 openSession() 메소드를 통해 SqlSession 객체 생성

▶ SqlSessionFactoryBuilder 메소드

KH정보교육원

메소드	설명
build(InputStream)	config.xml 파일만 불러옴
build(InputStream, String)	config.xml 파일과 지정한 DB를 불러옴
build(InputStream, Properties)	config.xml 파일과 프로퍼티로 설정한 내용으로 불러옴 ("${ key명 }")
build(InputStream, String, Properties)	config.xml 파일과 지정한 DB, Properties 파일을 불러옴
build(configuration)	configuration 객체에 설정한 내용을 불러옴

* config.xml은 Resource 객체의 getResourceAsStream 메소드를 이용하여 InputStream으로 가져옴

▶ SqlSessionFactory 메소드

KH정보교육원

메소드	설명
openSession()	기본값을 통해 SqlSession 생성
openSession(Boolean)	SqlSession 생성 시 AutoCommit 여부를 true / false로 지정 (기본 값 : true)
openSession(Connection)	직접 생성한 Connection 객체를 이용해 SqlSession 생성 (기본 값 : X)
openSession(ExecutorType)	쿼리를 실행 할 때 PreparedStatement의 재사용 여부 설정 (기본 값 : ExecutorType.SIMPLE)

▶ SqlSession을 통한 쿼리 생성

KH정보교육원

1. Service클래스에서 getSqlSession메소드 호출을 통해 SqlSession 생성
2. DAO클래스의 메소드 호출 시 전달 인자로 SqlSession 객체 전달
3. DAO클래스의 메소드에서 SqlSession 객체를 통해 쿼리에 접근

```
public Member selectMember(SqlSession session, Member m) throws LoginFailException {
    // 리턴용 멤버 객체 선언
    Member member = null;

    member = session.selectOne("memberMapper.loginMember", m);

    if (member == null) {
        session.close();
        throw new LoginFailException("로그인 실패!!");
    }

    return member;
}
```

* mapper에 정의된 namespace명. query_id명을 통해 정의한 select문에 접근

▶ SqlSession을 통한 쿼리 실행

KH정보교육원

메소드	반환형	설명
selectOne(String mapper, Object param)	Object	하나의 객체만을 받고자 할 때 사용
selectList(String mapper, Object param)	List<E>	결과에 대한 값을 List로 받고자 할 때 사용
selectMap(String mapper, Object param, String mapKey)	Map<K, V>	결과에 대한 값을 Map으로 받고자 할 때 사용 (마지막 인자로 키로 사용될 컬럼 명시)
insert(String mapper, Object param)	int	DB에 데이터를 입력하고자 할 때 사용
update(String mapper, Object param)	int	DB에 데이터를 수정하고자 할 때 사용
delete(String mapper, Object param)	int	DB에 데이터를 삭제하고자 할 때 사용