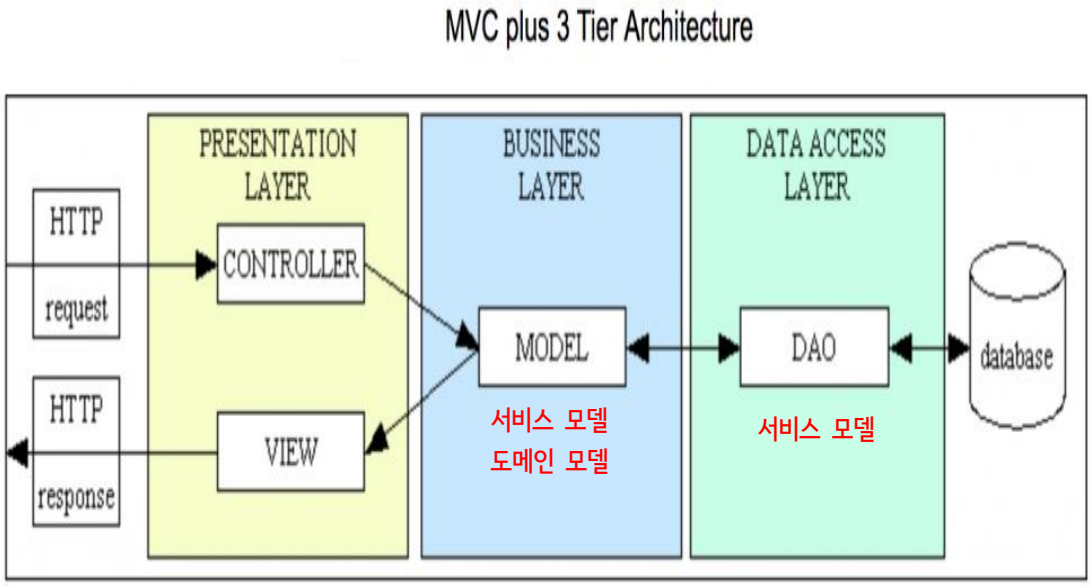
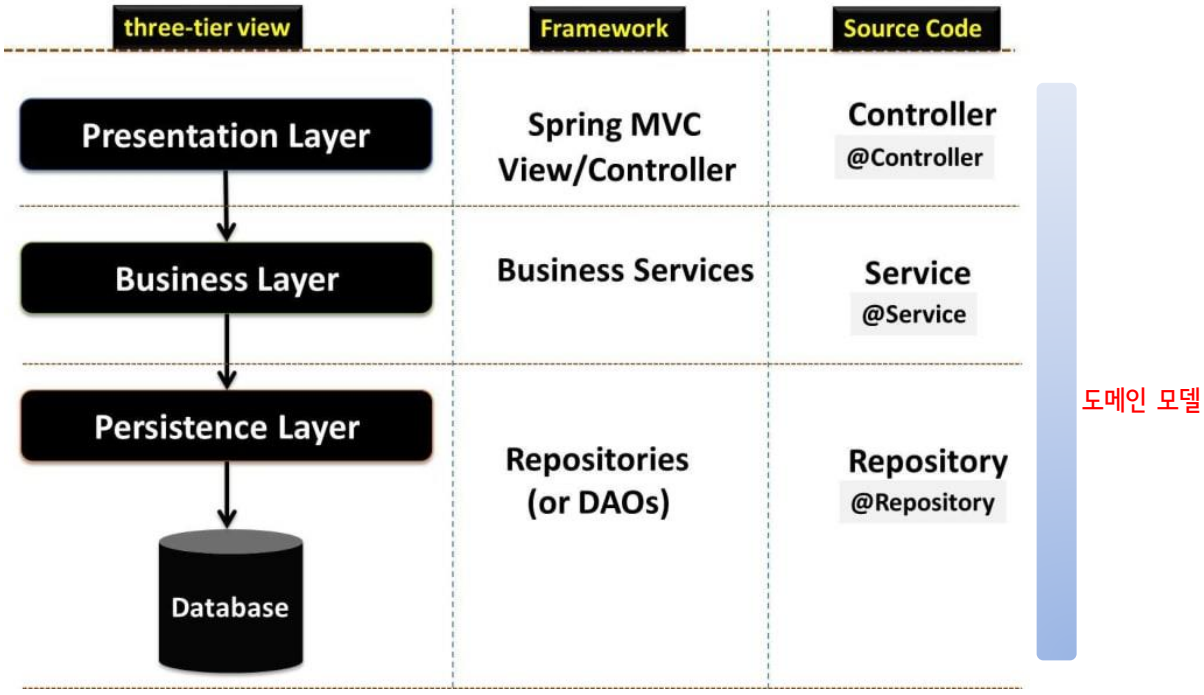




컨트롤러 : @Controller (프레젠테이션 레이어, 웹 요청과 응답을 템플릿을 통해 또는 직접 처리함)
로직 처리 : @Service (서비스 레이어, 필요시 내부에서 비즈니스 로직을 처리함)
외부I/O 처리 : @Repository (퍼시스턴스 레이어, DB나 파일같은 외부 I/O 작업을 처리함)

1. Mybatis 란?

자바에서는 DB 연동 프로그램을 구현하기 위해 JDBC 기술을 사용한다. JDBC는 관계형 데이터 베이스를 연동하기 위한 다양한 API를 제공한다. 다양한 관계형 데이터베이스를 지원하기 위해 많은 부분 인터페이스로 설계된 API 와 DB 서버에 알맞은 JDBC 드라이버를 사용한다.

원하는 CRUD 작업을 처리하기 관련 SQL 명령을 API 에서 제공되는 메서드에 아규먼트로 전달하여 수행시키고 실행된 결과를 읽어 도메인 객체 및 리스트 객체로 만들어 사용한다. 그런데 이러한 작업들이 매우 반복적이며 자바소스안에 자바코드와 SQL 코드가 혼재하다 보니 구현 소스가 복잡해 보이는 단점을 가지고 있다.

애플리케이션의 규모가 커지면서 SQL이 수백개가 넘는 경우가 많아졌다. SQL 자체의 체계적인 관리 방법이나 SQL의 입출력 데이터와 자바 객체의 효율적인 변환 방법 등 스프링의 기능만으로는 해결할 수 없는 과제가 발생했다.

Mybatis는 DB 연동 구현시 사용되는 자바 퍼시스턴스 프레임워크로서 SQL Mapper 기능을 지원한다. SQL 파일을 별도로 분리하여 관리할 수 있고, 객체-SQL 사이의 파라미터를 자동으로 매핑해주기 때문에 편리하다. JDBC로 처리하는 상당부분의 코드와 파라미터 설정및 결과 매핑을 Mybatis 프레임워크가 대신해준다.

MyBatis의 특징

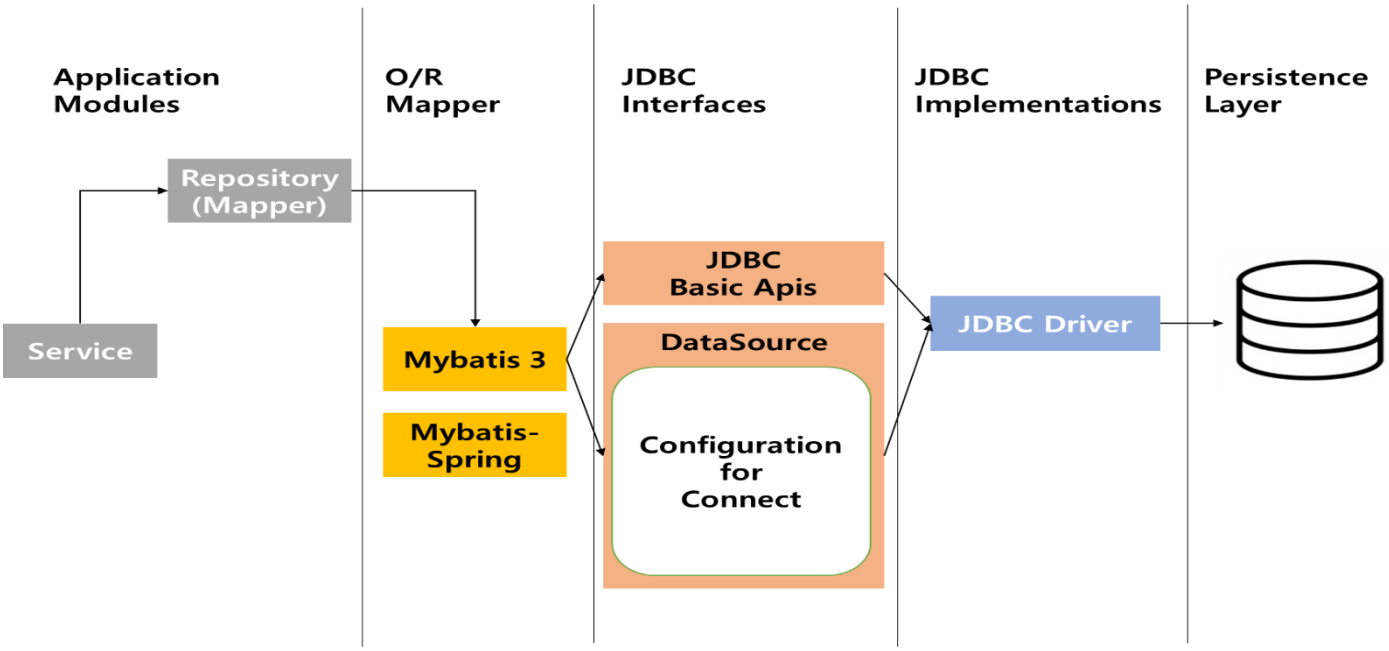
- 간단하다 : 간단한 퍼시스턴스 프레임워크
- 생산성 : 62%정도 줄어드는 코드 , 간단한 설정, 예외 처리도 선택적
- SQL문이 애플리케이션 소스 코드로부터 완전 분리
  SQL쿼리 변경 시마다 자바코드를 수정하거나 따로 컴파일 할 필요가 없다.
  SQL의 체계적인 관리, 선언적 정의(설정 파일, 애노테이션)
- 자바 객체와 SQL 입출력 값의 투명한 바인딩
- 동적 SQL 조합
- 이식성 : 어떤 프로그래밍 언어로도 구현가능 (자바,C#,.NET,RUBY)
- 오픈소스이며 무료이다.

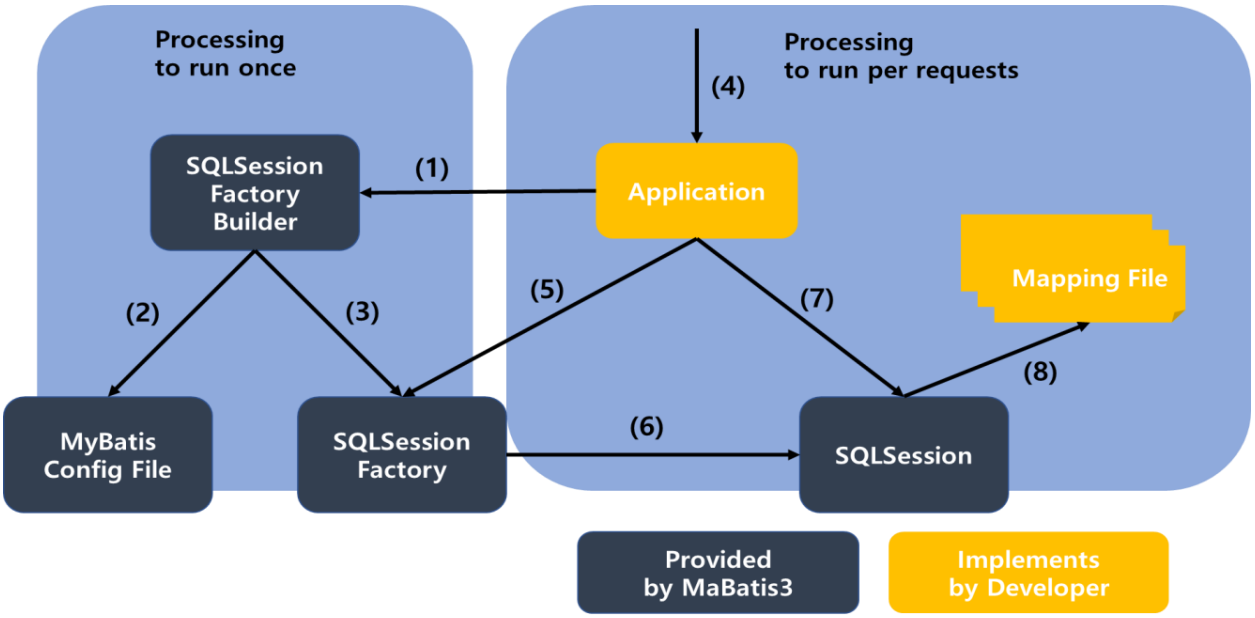
Mybatis는 SQL과 객체를 매핑하기 위한 'SQL Mapper'라고 부르는 것이 정확한 표현이다. Mybatis는 'MyBatis SQL Mapper Framework for Java'라는 이름을 가진다.

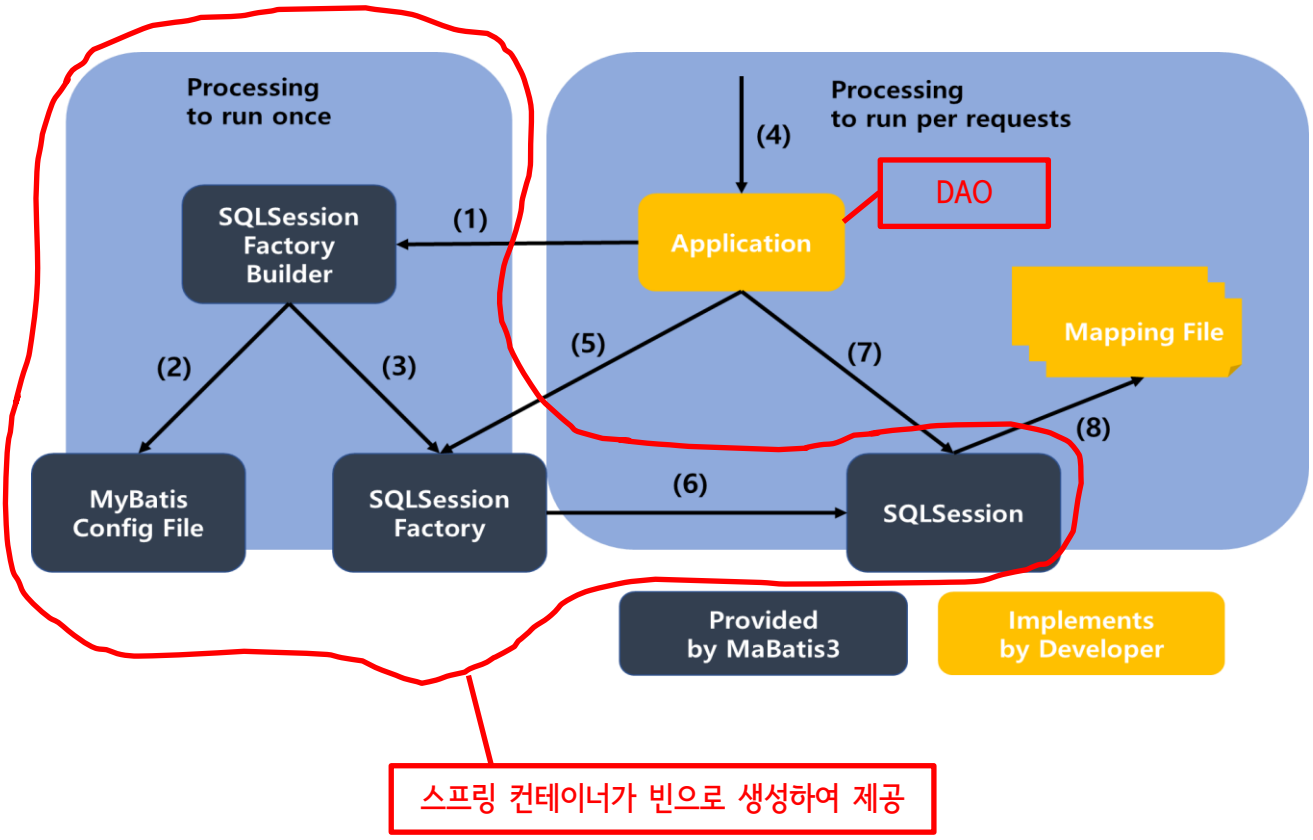
[SQL 정의 방법]

SQL 지정 방법	설명
매핑 파일	ibatis 시절부터 지원된 전통적인 지정 방법으로, 마이바티스 기능을 완벽하게 이용할 수 있다.
애노테이션	Mybatis3부터 지원되는 방법으로, 개발의 용이성을 우선시할 때 효과적이다. SQL 지정은 간단하지만 애노테이션의 표현력과 유연성의 제약 탓에 복잡한 SQL이나 매핑을 지정할 때는 적합하지 않다. 표준 기능은 지원하지만 매핑 파일에서 표현할 수 있는 모든 것이 지원되는 것은 아니다.

[Spring-Mybatis의 처리 흐름]







mybatis-spring-boot-autoconfig

mybatis.org/spring-boot-starter/mybatis-spring-boot-autoconfigure/

mybatis-spring-boot-autoconfigure

MyBatis

Last Published: 17 December 2022 | Version: 3.0.1

REFERENCE DOCUMENTATION
- Introduction

PROJECT DOCUMENTATION
- Project Information
  - CI Management
  - Dependencies
  - Dependency Information
  - Dependency Management
  - Distribution Management
  - About
  - Issue Management
  - Licenses
  - Mailing Lists
  - Plugin Management

Introduction

Translations

Users can read about MyBatis-Spring-Boot-Starter in the following translations:

English
简体中文

What is MyBatis-Spring-Boot-Starter?

The MyBatis-Spring-Boot-Starter help you build quickly MyBatis applications on top of the Spring Boot.

By using this module you will achieve:

- Build standalone applications
- Reduce the boilerplate to almost zero
- Less XML configuration

Requirements

The MyBatis-Spring-Boot-Starter requires following versions:

MyBatis-Spring-Boot-Starter	MyBatis-Spring	Spring Boot	Java
3.0	3.0	3.0	17 or higher
2.3	2.1	2.5 - 2.7	8 or higher
2.2	2.0 (need 2.0.6+ for enable all features)	2.5 - 2.7	8 or higher

www.mybatis.org

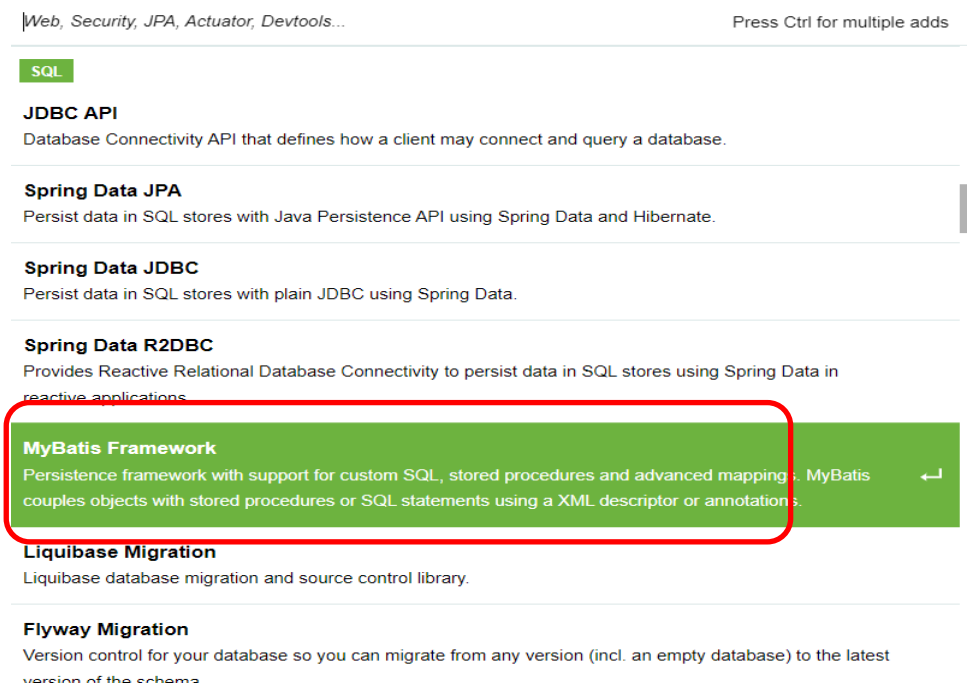
2. Mybatis 설치

Spring Boot 에서 Mybatis 를 사용하기 위해서는 MyBatis-Spring API 를 따로 설치해야 한다.

build.gradle 의 dependencies 항목에 다음에 제시된 API 의존성 정보를 추가한다.

```
implementation 'org.mybatis.spring.boot:mybatis-spring-boot-starter:2.3.0'
```

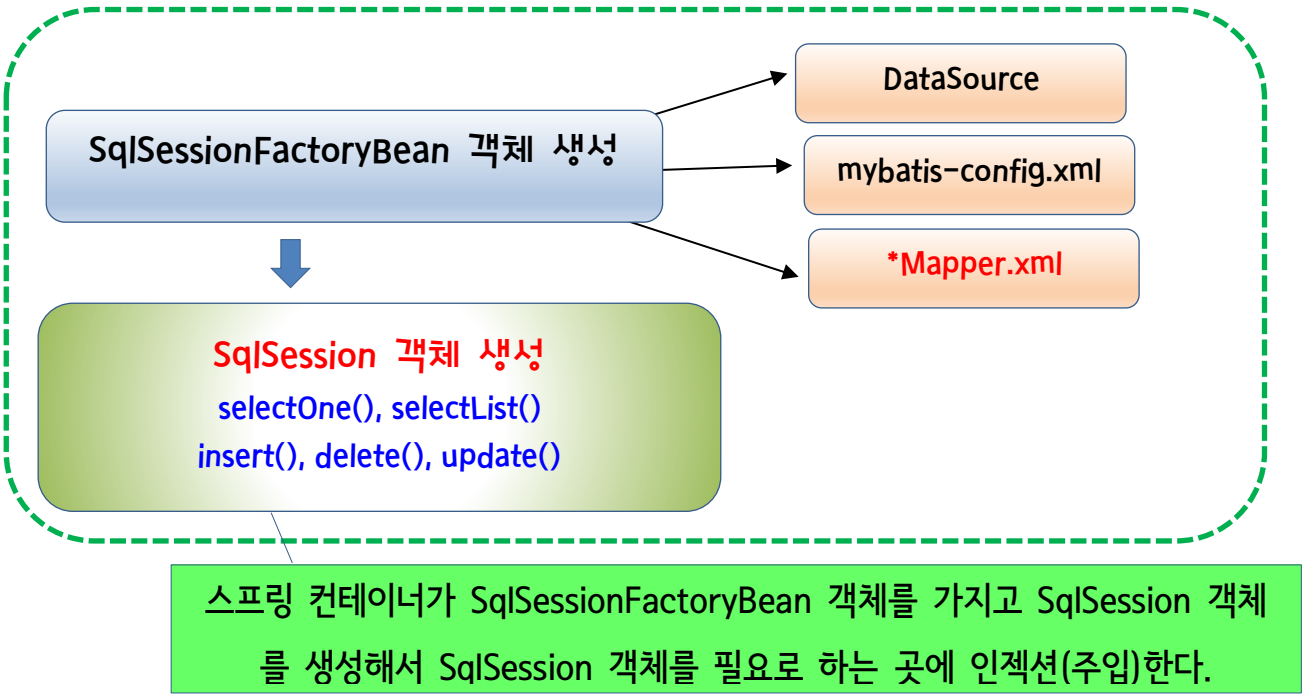
물론 spring initializr 에서 스프링 부트 프로젝트를 만들 때 다음과 같이 dependencies 설정시 Mybatis 정보를 선택해도 된다.



또한 application.properties 파일에 다음 정보를 추가한다.

```
# database
spring.datasource.url: jdbc:mysql://localhost:3306/edudb?characterEncoding=UTF-8&serverTimezone=UTC
spring.datasource.username: jdbctest
spring.datasource.password: jdbctest
spring.datasource.driver-class-name: com.mysql.cj.jdbc.Driver
# mybatis
mybatis.mapper-locations: mybatis/mapper/*.xml
mybatis.type-aliases-package=com.example.springedu.domain
```

- Mybatis 의 주요 객체



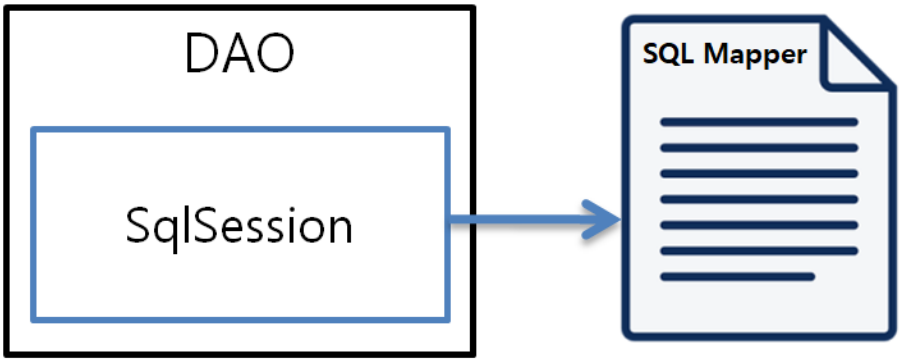
스프링 컨테이너가 SqlSessionFactoryBean 객체를 가지고 SqlSession 객체를 생성해서 SqlSession 객체를 필요로 하는 곳에 인젝션(주입)한다.

SqlSession 객체의 주요 메서드 - org.apache.ibatis.session.SqlSession

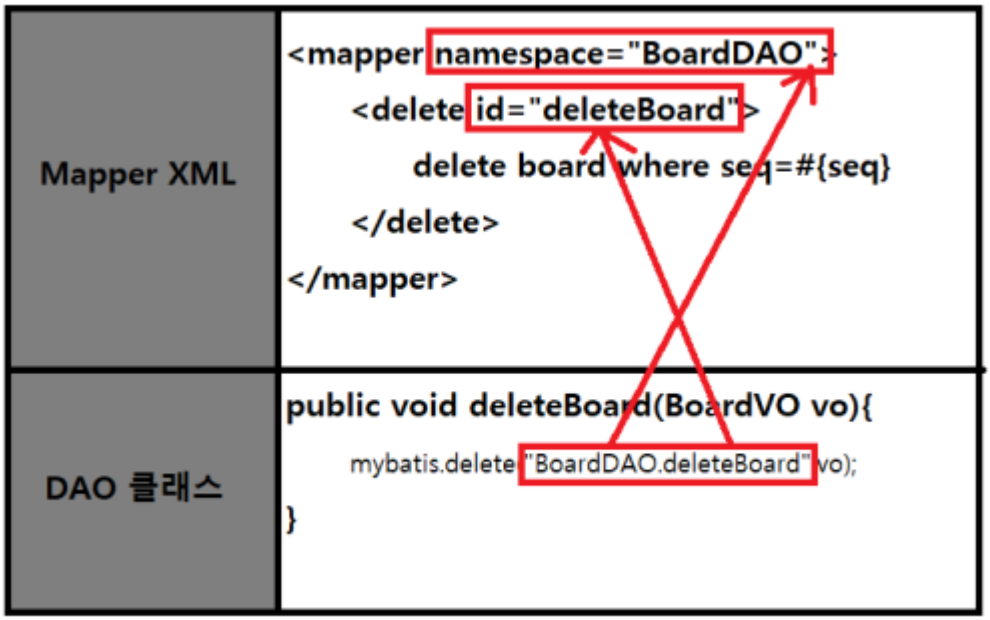
SQL 발행이나 트랜잭션 제어용 API 를 제공하는 컴포넌트다.
마이바티스를 이용해 데이터베이스에 접근할 때 가장 중요한 역할을 하는 컴포넌트다.
스프링 프레임워크에서 사용하는 경우에는 마이바티스 측의 트랜잭션 제어 API 는 사용하지 않는다.
SqlSession 이 Mapper 파일에서 SQL 을 수행하고 결과 데이터를 반환하는 역할을 한다.

```
T selectOne(String statement, Object parameter)
T selectOne(String statement)
List<E> selectList(String statement, Object parameter)
List<E> selectList(String statement)
<T> Cursor<T> selectCursor(String statement, Object parameter)
<T> Cursor<T> selectCursor(String statement)
Map<K,V> selectMap(String statement, Object parameter, String mapKey)
Map<K,V> selectMap(String statement, Object parameter)
int insert(String statement, Object parameter)
int insert(String statement)
int delete(String statement, Object parameter)
int delete(String statement)
int update(String statement, Object parameter)
int update(String statement)
```

[Mybatis 의 SQL 매퍼 파일 생성과 활용]



<mapper namespace="……">
<select id="…" resultType="…" parameterType="…">
<insert id="…" parameterType="…">
<delete id="…" parameterType="…">
<update id="…" parameterType="…">
<sql id="…">
</mapper>



- 예제 1

```
<mapper namespace="testdb">
  <select id="countEmp" resultType="int">
        select count(*) from emp
  </select>
  <select id="selectEmp" resultType="EmpVO">
        select empno, ename, job, date_format(hiredate, '%Y년 %m월 %d일') hiredate,
                                    sal from emp
  </select>
  <select id="partEmp" resultType="EmpVO" parameterType="PageDTO" >
        select empno, ename, job, date_format(hiredate, '%Y년 %m월 %d일') hiredate,
              sal from emp order by sal limit #{startNum}, #{countNum}
  </select>
</mapper>
```

```
@Autowired
SqlSession session;
public int getAllDataNum() {
        String statement = "testdb.countEmp";
        int num = session.selectOne(statement);
        return num;
}
public List<EmpVO> listAll() {
        String statement = "testdb.selectEmp";
        List<EmpVO> list = session.selectList(statement);
        return list;
}
public List<EmpVO> listPart(PageDTO vo) {
        String statement = "testdb.partEmp";
        vo.setCountNum(vo.getEndNum()-vo.getStartNum()+1);
        List<EmpVO> list = session.selectList(statement, vo);
        return list;
}
```

T selectOne(String statement, Object parameter)

List<E> selectList(String statement, Object parameter)

- 예제 2

```
<mapper namespace="resource.VisitorMapper">
	<select id="selectVisitor" resultType="VisitorDTO">
	  select id, name, date_format(writedate, '%Y년 %m월 %d일') writedate, memo from visitor
	</select>
	<select id="selectVisitorOne" resultType="VisitorDTO" parameterType="_int" >
	  select id, name, date_format(writedate, '%Y년 %m월 %d일') writedate, memo from visitor
	   where id = #{id}
	</select>
	<insert id="insertVisitor"   parameterType="VisitorDTO">
	    insert into visitor (name, writedate, memo) values (#{name}, now(), #{memo})
	</insert>
	<select id="searchVisitor"   parameterType="java.lang.String" resultType="VisitorDTO">
	  select id, name, date_format(writedate, '%Y년 %m월 %d일') writedate, memo from visitor
	    where memo like concat('%',#{key},'%')
	</select>
	<delete id="deleteVisitor"   parameterType="_int"  >
	  delete from visitor where id = #{id}
	</delete>
	<update id="updateVisitor"   parameterType="VisitorDTO">
	  update visitor set name = #{name}, memo = #{memo} where id = #{id}
	</update>
</mapper>
```

```
	@Autowired
	SqlSession session;
	public List<VisitorDTO> list() {
		List<VisitorDTO> list = null;
		try {
			String statement = "resource.VisitorMapper.selectVisitor";
			list = session.selectList(statement);
		} catch (Exception e) {
			e.printStackTrace();
		}
		return list;
	}
	public VisitorDTO one(int id) {
		VisitorDTO vo = null;
		try {
			String statement = "resource.VisitorMapper.selectVisitorOne";
			vo = session.selectOne(statement, id);
		} catch (Exception e) {
			e.printStackTrace();
		}
		return vo;
	}
```

```
	public List<VisitorDTO> search(String keyword) {
		List<VisitorDTO> list = null;
		try {
			String statement = "resource.VisitorMapper.searchVisitor";
			list = session.selectList(statement, keyword);
		} catch (Exception e) {
			e.printStackTrace();
		}
		return list;
	}
	public boolean insert(VisitorDTO visitor) {
		boolean result = false;
		try {
			String statement = "resource.VisitorMapper.insertVisitor";
			session.insert(statement, visitor);
			result = true;
		} catch (Exception e) {
			e.printStackTrace();
		}
		return result;
	}
	public boolean delete(String id) {
		boolean result = false;
		try {
			String statement = "resource.VisitorMapper.deleteVisitor";
			session.delete(statement, Integer.parseInt(id));
			result = true;
		} catch (Exception e) {
			e.printStackTrace();
		}
		return result;
	}
	public boolean update(VisitorDTO visitor) {
		boolean result = false;
		try {
			String statement = "resource.VisitorMapper.updateVisitor";
			session.update(statement, visitor);
			result = true;
		} catch (Exception e) {
			e.printStackTrace();
		}
		return result;
	}
```

- typeAliases

타입 별칭은 자바 타입에 대한 짧은 이름, XML 설정에서만 사용되며, 타이핑을 줄이기 위해 활용한다.(mybatis-config.xml 사용)

```
<typeAliases>
  <typeAlias alias="Author" type="domain.blog.Author"/>
  <typeAlias alias="Comment" type="domain.blog.Comment"/>
</typeAliases>
```

다음은 Mybatis 에 내장된 별칭으로서 대소문자를 구분한다.

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