

*

← SQL + JDBC

spring + My Batis

JDBC. Template
공통 코드 x
머신

Spring Framework

- MyBatis 연동

필수!

■ 개요

- **JDBC 코드의 패턴**

- Connection -> Statement -> 쿼리전송-> 연결 close
- 모든 JDBC 코드는 위의 패턴을 가진다.
- 이 패턴을 캡슐화 하여 JDBC 코드를 간편하게 사용할 수 있도록 Framework화 가능

- **iBatis(MyBatis) 란**

- SQL실행 결과를 자바 빈즈 혹은 Map 객체에 매핑 해주는 Persistence 솔루션으로 SQL을 소스코드가 아닌 XML로 따로 분리해 관리하도록 지원

- **장점**

- SQL 문장과 프로그래밍 코드의 분리
- JDBC 라이브러리를 통해 매개변수를 전달하고 결과를 추출하는 일을 간단히 처리가능
- 자주 쓰이는 데이터를 변경되지 않는 동안에 임시 보관(Cache) 가능
- 트랜잭션처리 제공

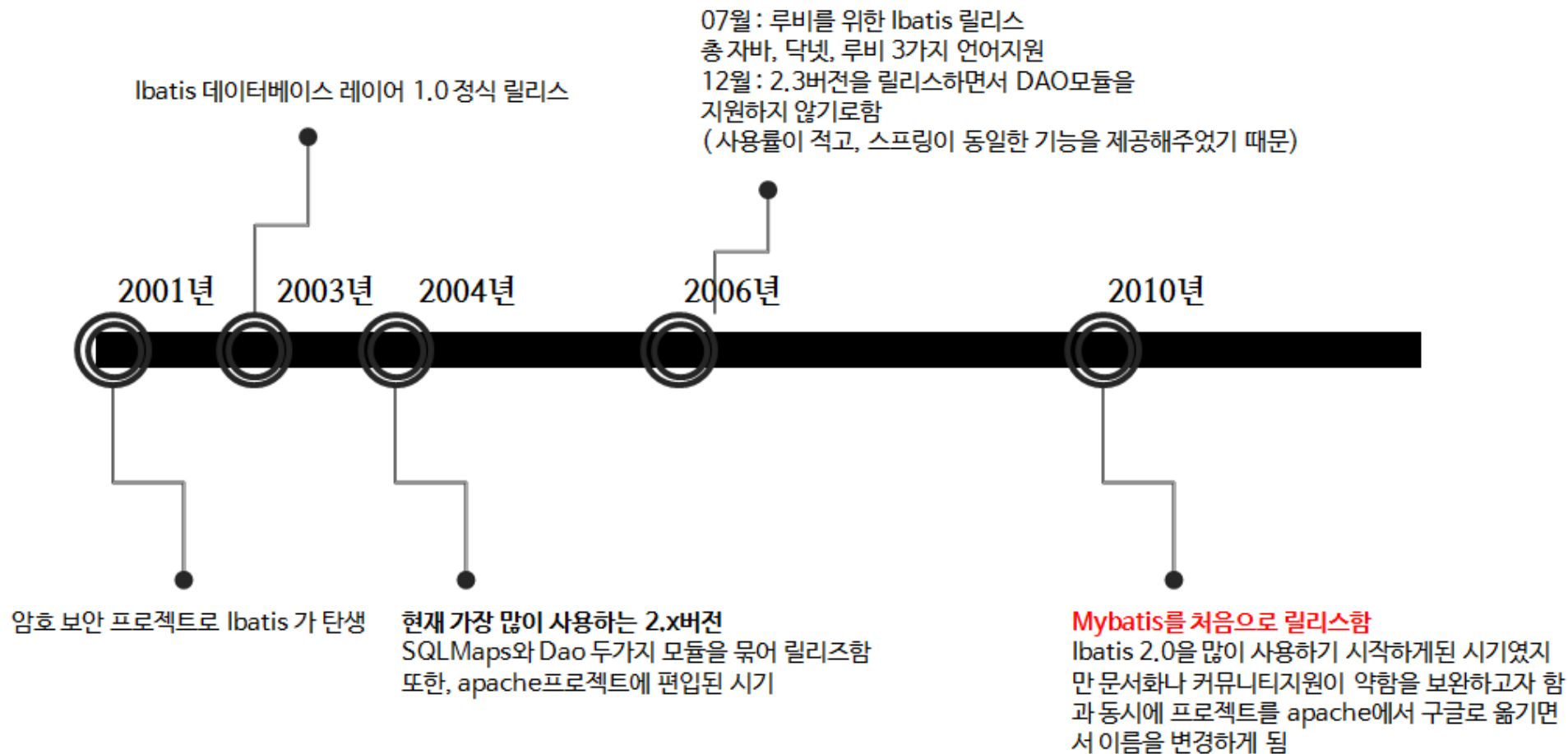
■ MyBatis 새로운 기능

- 전반적으로 크게 달라진 점은 없어 기존에 ibatis를 사용하셨던 개발자들은 어려움 없이 사용가능.



MyBatis

■ MyBatis 역사



- 현재 Mybatis에서 제공하는 최신버전은 3.3.0버전
- <http://blog.mybatis.org/p/products.html> 에서 다운받을 수 있도록 제공

■ Mybatis와 ibatis의 차이점

구분	ibatis(아이바티스)	Mybatis(마이바티스)
네임스페이스	선택사항	필수사항
매핑구문정의	xml만 사용	xml과 에노테이션을 사용
동적 SQL	XML 엘리먼트만 사용 동적 SQL을 위한 XML 엘리먼트는 16개 내외	XML 엘리먼트 및 구문 빌더 사용 동적SQL을 위한 XML 엘리먼트는 4개 내외 (if, choose, trim, foreach, set)
스프링 연동	스프링 자체 구현체 사용	마이바티스 별도 모듈 사용
지원계획	향후 아이바티스에 대한 공식적인 릴리스는 없음	향후 계속 릴리스 될 예정

■ XML과 에노테이션

- XML

- 사용방법은 기존과 동일하나 용어들이 변경

이전용어	변경된 용어
SqlMapConfig	Configuration
sqlMap	mapper

- XML 엘리먼트가 축소

if	조건문
choose(when otherwise)	반복문
trim(where)	공백제거
foreach	반복문 (IN절 사용시 유용)
set	update에서 마지막으로 명시된 칼럼 표기에서 쉼표제거

■ Mybatis

- Mybatis는 직접 스프링과 연동하기 위한 모듈을 제공하고 있어 이 모듈을 이용해서 스프링이 제공하는 DataSource 및 트랜잭션 관리 기능을 MyBatis에 적용이 가능하다.
- **Spring 과 MyBatis와 연동 방법**
 - MyBatis-Spring 모듈 추가
 - SqlSessionFactoryBean을 이용해서 SqlSessionFactory 설정
 - 트랜잭션 설정
 - MyBatis 를 이용해서 DAO 구현
 - SqlSession 을 이용해서 구현
 - 매퍼 동적 생성을 이용해서 구현

■ Mybatis Spring 모듈 추가

```
<!-- https://mvnrepository.com/artifact/org.springframework/spring-jdbc -->  
<dependency>  
    <groupId>org.springframework</groupId>  
    <artifactId>spring-jdbc</artifactId>  
    <version>${org.springframework-version}</version>  
</dependency>
```

```
<!-- https://mvnrepository.com/artifact/org.springframework/spring-tx -->  
<dependency>  
    <groupId>org.springframework</groupId>  
    <artifactId>spring-tx</artifactId>  
    <version>${org.springframework-version}</version>  
</dependency>
```

```
<!-- https://mvnrepository.com/artifact/org.mybatis/mybatis -->  
<dependency>  
    <groupId>org.mybatis</groupId>  
    <artifactId>mybatis</artifactId>  
    <version>3.4.1</version>  
</dependency>
```

```
<!-- https://mvnrepository.com/artifact/org.mybatis/mybatis-spring -->  
<dependency>  
    <groupId>org.mybatis</groupId>  
    <artifactId>mybatis-spring</artifactId>  
    <version>1.3.0</version>  
</dependency>
```


■ SqlSessionFactoryBean 과 트랜잭션 관리자 설정

- mybatis-spring 모듈이 제공하는 SqlSessionFactoryBean을 이용해서 mybatis의 SqlSessionFactory를 생성

MyBatis 설정

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

■ SqlSessionFactoryBean 과 트랜잭션 관리자 설정

- Mapper 작성 (MemberMapper.xml)

```
<?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="com.bitcamp.memberboard.mapper.mybatis.MemberMapper">

  <select id="selectById"
    resultType="com.bitcamp.openproject.member.model.Member">
    select * from member where member_id = #{id}
  </select>

  <insert id="insertMember"
    parameterType="com.bitcamp.openproject.member.model.Member">

    insert into member
    values( #{member_id}, #{member_name}, #{password}, #{regdate}, #{photo})

  </insert>
</mapper>
```

■ SqlSessionFactoryBean 과 트랜잭션 관리자 설정

- MyBatis 트랜잭션 관리자는 DataSourceTransactionManager를 사용
servlet-context.xml 아래 코드 적용

```
<!-- 트랜잭션 처리 시작 -->
```

```
<beans:bean id="transactionManager"  
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">  
    <beans:property name="dataSource">  
        <beans:ref bean="dataSource" />  
    </beans:property>  
</beans:bean>
```

```
<tx:annotation-driven transaction-manager="transactionManager" />
```

```
<!-- 트랜잭션 처리 끝 -->
```

■ MyBatis를 이용한 DAO 구현

- SqlSessionFactoryBean을 이용해서 mybatis의 SqlSessionFactory를 생성하면 MyBatis를 이용해서 DAO 구성 가능
- DAO 구현 방법은 두 가지로 구분
 - **SqlSessionTemplate을 이용한 DAO 구현**
 - **자동 매퍼 생성 기능을 이용한 DAO 구현**

■ SqlSessionFactory를 이용한 DAO 구현

- SqlSessionFactory 클래스는 SqlSession 을 위한 스프링 연동 부분을 구현하고 있고, DAO 는 SqlSessionFactory를 이용해서 구현

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

<beans:bean id="sqlSession" class="org.mybatis.spring.SqlSessionTemplate">
    <beans:constructor-arg index="0" ref="sqlSessionFactory" />
</beans:bean>

<beans:bean id="mybatisDao"
    class="com.bitcamp.memberboard.member.dao.MemberMyBatisDao" />
```

■ SqlSessionTemplate을 이용한 DAO 구현

```
com.bitcamp.openproject.member.dao
```

```
public class MemberMyBatisDao {
```

```
    @Autowired
```

```
    private SqlSessionTemplate sqlSessionTemplate;
```

```
    public void insert(Member member) {
```

```
        String str =
```

```
        "com.bitcamp.openproject.mapper.mybatis.MemberMapper.insertMember";
```

```
        int num = sqlSessionTemplate.update(str, member);
```

```
        System.out.println(num);
```

```
    }
```

```
    public Member selectById(String userid) {
```

```
        String str =
```

```
        "com.bitcamp.openproject.mapper.mybatis.MemberMapper.selectById";
```

```
        return (Member) sqlSessionTemplate.selectOne(str, userid);
```

```
    }
```

```
}
```

■ 자동 매퍼 생성 기능을 이용한 DAO 구현

- MyBatis를 이용해서 DAO를 구현 할 때 대부분의 코드는 단순히 SqlSession의 메서드를 호출하는 것으로 끝난다.

이러한 단순 코드 작업을 줄여주기 위해서 MyBatis는 인터페이스를 이용해서 런타임에 매퍼 객체를 생성하는 기능을 제공.

■ 자동 매퍼 생성 기능을 이용한 DAO 구현

```
com.bitcamp.openproject.member.dao.LoginDao.java
```

```
package com.bitcamp.openproject.member.dao;
```

```
import com.bitcamp.openproject.member.model.Member;
```

```
public interface LoginDao {
```

```
    public Member selectById(String userId);
```

```
    public void insertMember(Member member);
```

```
}
```


■ 자동 매퍼 생성 기능을 이용한 DAO 구현

```
com.bitcamp.openproject.member.dao.LoginService.java
```

```
public class LoginService {
```

```
    // @Autowired
```

```
    // private MemberMyBatisDao dao;
```

```
    @Autowired
```

```
    private SqlSessionTemplate sqlSessionTemplate;
```

```
    private LoginDao dao;
```

```
    @Transactional
```

```
    public User login(
```

```
        String userid,
```

```
        String password,
```

```
        HttpServletRequest req) throws SQLException {
```

```
        dao = sqlSessionTemplate.getMapper(LoginDao.class);
```

■ 자동 매퍼 생성 기능을 이용한 DAO 구현

com.bitcamp.openproject.member.dao.LoginService.java

```
Member member = dao.selectById(userid);  
System.out.println(member);
```

```
if (member == null) {  
    throw new Exception();  
}  
if (!member.matchPassword(password)) {  
    throw new Exception();  
}
```

```
User user = new User();  
user.setName(member.getName());  
user.setUserId(member.getMemberid());  
user.setPhoto(member.getPhoto());
```

```
return user;
```

```
}
```

```
}
```

■ XML과 에노테이션


- 참고사이트

- <http://mybatis.github.io/mybatis-3/ko/index.html>

Spring_02_DL_1_의존설정.prSpring_08_MyBatis연동_08CMyBatis - 마이바티스 3

https://mybatis.org/mybatis-3/ko/index.html

mybatis



Last Published: 20 10월 2019 | Version: 3.5.3

CORE

소개

시작하기

매퍼 설정

Mapper XML 파일

동적 SQL

자바 API

SQL Builder 클래스

로깅

프로젝트 문서화

프로젝트 정보

프로젝트 보고서



소개

대신해준다

JDBC, API

마이바티스는 무엇인가?

마이바티스는 개발자가 지정한 SQL, 저장프로시저 그리고 몇가지 고급 매핑을 지원하는 퍼시스턴스 프레임워크이다. 마이바티스는 JDBC로 처리하는 상당부분의 코드와 파라미터 설정 및 결과 매핑을 대신해준다. 마이바티스는 데이터베이스 레코드에 원시타입과 Map 인터페이스 그리고 자바 POJO를 설정해서 매핑하기 위해 XML과 애노테이션을 사용할 수 있다.

bean

이 문서가 더 좋아지도록 도와주세요...

이 문서가 일부 내용을 자세히 다루지 않거나 특정 기능에 대해 설명하지 않을 경우 마이바티스를 공부하는 가장 좋은 방법은 사용자 스스로 그 부분에 대한 문서를 작성하는 것이다!


이 문서는 xdoc포맷으로 되어 있으며 프로젝트 깃허브에서 볼수 있다. 저장소를 포크하고 수정한 뒤 Pull request요청을 보내면 됩니다.


그러면 이 문서의 가장 멋진 저자가 되고 다른 많은 사람들이 이 문서를 읽을 것이다!

번역에 대하여

번역자 : 이동국(fromm0@gmail.com, http://ldg.pe.kr, https://www.facebook.com/dongguk.lee.3)

다음의 몇가지 언어로 번역된 마이바티스 문서를 읽을수 있다.

영어

스페인어

바탕 화면

오전 10:13 2020-01-14

Spring_02_DI_1_의존설정.pr Spring_08_MyBatis연동_08C MyBatis - 마이바티스 3 X

https://mybatis.org/mybatis-3/ko/getting-started.html

XML파일에서 SqlSessionFactory인스턴스를 빌드하는 것은 매우 간단하다. 빌드를 위해 클래스패스 자원을 사용하는 것을 추천한다. 파일 경로나 메소드 URL로부터 읽어들이는 InputStream인스턴스를 사용할 수도 있다. 마이바티스는 클래스패스와 다른 위치에서 자원을 로드하는 것으로 좀더 쉽게 해주는 Resources 라는 유틸성 클래스를 가지고 있다.

```
String resource = "org/mybatis/example/mybatis-config.xml";
InputStream inputStream = Resources.getResourceAsStream(resource);
SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);
```

XML설정파일에서 지정하는 마이바티스의 핵심이 되는 설정은 트랜잭션을 제어하기 위한 TransactionManager과 함께 데이터베이스 Connection인스턴스를 가져오기 위한 DataSource 를 포함한다. 세부적인 설정은 조금 뒤에 보고 간단한 예제를 먼저보자.

설정파일

```
<?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>
```

CONNECTION

SQL

MAPPER.XML

좀 더 많은 XML 설정방법이 있지만 위 예제에서는 가장 핵심적인 부분만을 보여주고 있다. XML 가장 위부분에서는 XML 문서의 유효성체크를 위해 필요하다. environment엘리먼트는 트랜잭션 관리와 커넥션 풀링을 위한 환경적인 설정을 나타낸다. mappers엘리먼트는 SQL 코드와 매핑 정의를 가지는 XML 파일인 mapper 의 목록을 지정한다.

XML 을 사용하지 않고 SqlSessionFactory 빌드하기

XML 보다 자바를 사용해서 직접 설정하길 원한다면 XML 파일과 같은 모든 설정을 제공하는 Configuration 클래스를 사용하면 된다.

```
DataSource dataSource = BlogDataSourceFactory.getBlogDataSource();
TransactionFactory transactionFactory = new JdbcTransactionFactory();
```

그럼 자세히 살펴보자.

매핑된 SQL 구문 살펴보기

이 시점에 `SqlSession`이나 `Mapper` 클래스가 정확히 어떻게 실행되는지 궁금할 것이다. 매핑된 SQL 구문에 대한 내용이 가장 중요하다. 그래서 이 문서 전반에서 가장 자주 다루어진다. 하지만 다음의 두가지 예제를 통해 정확히 어떻게 작동하는지에 대해서는 충분히 이해하게 될 것이다.

위 예제처럼 구문은 XML이나 애노테이션을 사용해서 정의할 수 있다. 그럼 먼저 XML 을 보자. 마이바티스가 제공하는 대부분의 기능은 XML을 통해 매핑 기법을 사용한다. 이전에 마이바티스를 사용했었다면 쉽게 이해되겠지만 XML 매핑 문서에 이전보다 많은 기능이 추가되었다. `SqlSession`을 호출하는 XML 기반의 매핑 구문이다.

```
<?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="org.mybatis.example.BlogMapper">
  <select id="selectBlog" resultType="Blog">
    select * from Blog where id = #{id} full name
  </select>
</mapper>
```

간단한 예제치고는 조금 복잡해 보이지만 실제로는 굉장히 간단하다. 한 개의 매퍼 XML 파일에는 많은 수의 매핑 구문을 정의할 수 있다. XML 도입무의 헤더와 doctype 을 제외하면 나머지는 쉽게 이해되는 구문의 형태이다. 여기서 `org.mybatis.example.BlogMapper` 네임스페이스에서 `selectBlog`라는 매핑 구문을 정의했고 이는 결과적으로 `org.mybatis.example.BlogMapper.selectBlog`형태로 실제 명시되게 된다. 그래서 다음처럼 사용하게 되는 셈이다.

```
Blog blog = (Blog) session.selectOne("org.mybatis.example.BlogMapper.selectBlog", 101);
```

이건 마치 패키지를 포함한 전체 경로의 클래스내 메소드를 호출하는 것과 비슷한 형태이다. 이 이름은 매핑된 `select`구문의 이름과 파라미터 그리고 리턴타입을 가진 네임스페이스와 같은 이름의 `Mapper` 클래스와 직접 매핑될 수 있다. 이걸 위에서 본것과 같은 `Mapper` 인터페이스의 메소드를 간단히 호출하도록 허용한다. 위 예제에 대응되는 형태는 아래와 같다.

```
BlogMapper mapper = session.getMapper(BlogMapper.class);
Blog blog = mapper.selectBlog(101);
```

두번째 방법은 많은 장점을 가진다. 먼저 문자열에 의존하지 않는다는 것이다. 이는 애플리케이션을 좀더 안전하게 만든다. 두번째는 개발자가 IDE 를 사용할 때 매핑된 SQL 구문을 사용할때의 수고를 덜어준다. 세번째는 리턴타입에 대해 타입 캐스팅을 하지 않아도 된다. 그래서 `BlogMapper` 인터페이스는 깔끔하고 리턴 타입에 대해 타입에 안전하며 이는 파라미터에도 그대로 적용된다.

참고 네임스페이스(Namespace)에 대한 설명

네임스페이스(Namespace)가 이전버전에서는 사실 선택사항이었다. 하지만 이제는 패키지경로를 포함한 전체 이름을 가진 구문을 구분하기 위해 필수로 사용해야 한다.

CORE

소개

시작하기

매퍼 설정

Mapper XML 파일

select

insert, update and delete

Parameters

Result Maps

Auto-mapping

cache

동적 SQL

자바 API

SQL Builder 클래스

로깅

프로젝트 문서화

프로젝트 정보

프로젝트 보고서

Built by:

maven

Mapper XML 파일

마이바티스의 가장 큰 장점은 매핑구문이다. 이걸 간혹 마법을 부리는 것처럼 보일 수 있다. SQL Map XML 파일은 상대적으로 간단하다. 더군다나 동일한 기능의 JDBC 코드와 비교하면 아마도 95% 이상 코드수가 감소하기도 한다. 마이바티스는 SQL을 작성하는데 집중하도록 만들어졌다.

SQL Map XML파일은 첫번째(first class)엘리먼트만을 가진다.

- cache - 해당 네임스페이스를 위한 캐시 설정
- cache-ref - 다른 네임스페이스의 캐시 설정에 대한 참조
- resultMap - 데이터베이스 결과데이터를 객체에 로드하는 방법을 정의하는 엘리먼트
- parameterMap - 비권장됨! 예전에 파라미터를 매핑하기 위해 사용되었으나 현재는 사용하지 않음
- sql - 다른 구문에서 재사용하기 위한 SQL 조각
- insert - 매핑된 INSERT 구문.
- update - 매핑된 UPDATE 구문.
- delete - 매핑된 DELETE 구문.
- select - 매핑된 SELECT 구문.

다음 섹션에서는 각각에 대해 세부적으로 살펴볼 것이다.

select

select구문은 마이바티스에서 가장 흔히 사용할 엘리먼트이다. 데이터베이스에서 데이터를 가져온다. 아마도 대부분의 애플리케이션은 데이터를 수정하기보다는 조회하는 기능이 많다. 그래서 마이바티스는 데이터를 조회하고 그 결과를 매핑하는데 집중하고 있다. 조회는 다음 예제처럼 단순한 경우에는 단순하게 설정된다.

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

이 구문의 이름은 selectPerson이고 int타입의 파라미터를 가진다. 그리고 결과 데이터는 HashMap 에 저장된다.

파라미터 표기법을 보자.

```
#{id}
```

select결과



비권장됨! 예전에 파라미터를 매핑하기 위해 사용되었으나 현재는 사용하지 않음

SQL 조각

식별


```
String selectPerson = "SELECT * FROM PERSON WHERE ID=?";
PreparedStatement ps = conn.prepareStatement(selectPerson);
ps.setInt(1,id);
```

물론 JDBC를 사용하면 결과를 가져와서 객체의 인스턴스에 매핑하기 위한 많은 코드가 필요하겠지만 마이바티스는 그 코드들을 작성하지 않아도 되게 해준다.

select 엘리먼트는 각각의 구문이 처리하는 방식에 대해 세부적으로 설정하도록 많은 속성을 설정할 수 있다.

```
<select
  id="selectPerson"
  parameterType="int"
  parameterMap="deprecated"
  resultType="hashmap"
  resultMap="personResultMap"
  flushCache="false"
  useCache="true"
  timeout="10"
  fetchSize="256"
  statementType="PREPARED"
  resultSetType="FORWARD_ONLY">
```

id 구문을 찾기 위해 사용될 수 있는 네임스페이스 내 유일한 구분자

parameterType 구문에 전달될 파라미터의 패키지 경로를 포함한 전체 클래스명이나 별칭

parameterMap 외부 parameterMap을 찾기 위한 비권장된 접근방법. 인라인 파라미터 매핑과 parameterType을 대신 사용하라.

resultType 이 구문에 의해 리턴되는 기대타입의 패키지 경로를 포함한 전체 클래스명이나 별칭. collection인 경우 collection 타입 자체가 아닌 collection 이 포함된 타입이 될 수 있다. resultType이나 resultMap을 사용하라.

resultMap 외부 resultMap의 참조명. 결과는 마이바티스의 가장 강력한 기능이다. resultType이나 resultMap을 사용하라.

flushCache 이 값을 true로 셋팅하면 구문이 호출될때마다 로컬, 2nd 레벨 캐시가 지워질것이다(flush). 디폴트는 false이다.

useCache 이 값을 true로 셋팅하면 구문의 결과가 2nd 레벨 캐시에 캐시 될 것이다. 디폴트는 true이다.

timeout 예외가 던져지기 전에 데이터베이스의 요청 결과를 기다리는 최대시간을 설정한다. 디폴트는 셋팅하지 않는 것이고 드라이버에 따라 다소 지원 되지 않을 수 있다.

fetchSize 지정된 수만큼의 결과를 리턴하도록 하는 드라이버 힌트 형태의 값이다. 디폴트는 셋팅하지 않는 것이고 드라이버에 따라 다소 지원되지 않을 수 있다.

statementType STATEMENT, PREPARED 또는 CALLABLE 중 하나를 선택할 수 있다. 마이바티스에게 Statement, PreparedStatement 또는 CallableStatement를 사용하게 한다. 디폴트는 PREPARED이다.

resultSetType FORWARD_ONLY|SCROLL_SENSITIVE|SCROLL_INSENSITIVE|DEFAULT(same as unset)중 하나를 선택할 수 있다. 디폴트는 셋팅하지 않는 것이고 드라이버에 따라 다소 지원되지 않을 수 있다.

databaseId 설정된 databaseIdProvider가 있는 경우 마이바티스는 databaseId 속성이 없는 모든 구문을 로드하거나 일치하는 databaseId와 함께 로드 될 것이다. 같은 구문에서 databaseId가 있거나 없는 경우 모두 있다면 뒤에 나온 것이 무시된다.

resultOrdered 이 설정은 내포된 결과를 조회하는 구문에서만 적용이 가능하다. true로 설정하면 내포된 결과를 가져오거나 새로운 주요 결과 레코드를 리턴할 때 함께 가져오도록 한다. 이전의 결과 레코드에 대한 참조는 더 이상 발생하지 않는다. 이 설정은 내포된 결과를 처리할때 조금 더 많은 메모리

insert, update and delete

데이터를 변경하는 구문인 insert, update, delete는 매우 간단하다.

```
<insert
  id="insertAuthor"
  parameterType="domain.blog.Author"
  flushCache="true"
  statementType="PREPARED"
  keyProperty=""
  keyColumn=""
  useGeneratedKeys="true"
  timeout="20">
```

insert 자동증가한 컬럼의 값을 받아옴

```
<update
  id="updateAuthor"
  parameterType="domain.blog.Author"
  flushCache="true"
  statementType="PREPARED"
  timeout="20">
```

```
<delete
  id="deleteAuthor"
  parameterType="domain.blog.Author"
  flushCache="true"
  statementType="PREPARED"
  timeout="20">
```

Insert, Update 와 Delete 엘리먼트 속성

sqlSession에는 20개 이상의 메소드가 있다. 좀더 역량이 모아서 보도록 하자.

구문을 실행하는 메소드

namespace.{id}

이 메소드들은 SQL 매핑 XML 파일에 정의된 SELECT, INSERT, UPDATE 그리고 DELETE 구문을 실행하기 위해 사용된다. 메소드 이름 자체가 그 역할을 설명하도록 명명되었다. 메소드 각각은 구문의 ID 와 파라미터 객체(원시타입, 자바빈, POJO 또는 Map)을 가진다.

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

selectOne과 selectList의 차이점은 selectOne메소드는 오직 하나의 객체만을 리턴해야 한다는 것이다. 한개 이상을 리턴하거나 null 이 리턴된다면 예외가 발생할 것이다. 얼마나 많은 객체가 리턴될지 모른다면 selectList를 사용하라. 객체의 존재여부를 체크하고 싶다면 개수를 리턴하는 방법이 더 좋다. selectMap은 결과 목록을 Map으로 변환하기 위해 디자인된 특별한 경우이다. 이 경우 결과 객체의 프로퍼티 중 하나를 키로 사용하게 된다. 모든 구문이 파라미터를 필요로 하지는 않기 때문에 파라미터 객체를 요구하지 않는 형태로 오버로드되었다.