

MyBatis

강사 : 강병준

MyBatis

- ❖ 마이바티스는 개발자가 지정한 SQL, 저장프로시저를 지원하는 퍼시스턴스 프레임워크
- ❖ 마이바티스는 JDBC로 처리하는 상당부분의 코드와 파라미터 설정 및 결과 매핑을 대신 수행해줍니다.
- ❖ 마이바티스는 데이터베이스 레코드에 원시타입과 Map 인터페이스 그리고 자바 POJO(Plain Old Java Object) 를 설정해서 매핑하기 위해 XML과 애노테이션을 사용할 수 있습니다.
- myBatis 다운로드: <http://blog.mybatis.org/p/products.html>

- Maven Dependency

```
<!-- mybatis -->
```

```
<dependency>
```

```
  <groupId>org.mybatis</groupId>
```

```
  <artifactId>mybatis</artifactId>
```

```
  <version>3.5.2</version>
```

```
  <!-- <version>3.2.3</version> -->
```

```
</dependency>
```

```
<dependency>
```

```
  <groupId>org.mybatis</groupId>
```

```
  <artifactId>mybatis-spring</artifactId>
```

```
  <version>1.3.1</version>
```

```
  <!-- <version>1.2.1</version> -->
```

```
</dependency>
```

MyBatis

※ MyBatis의 특징

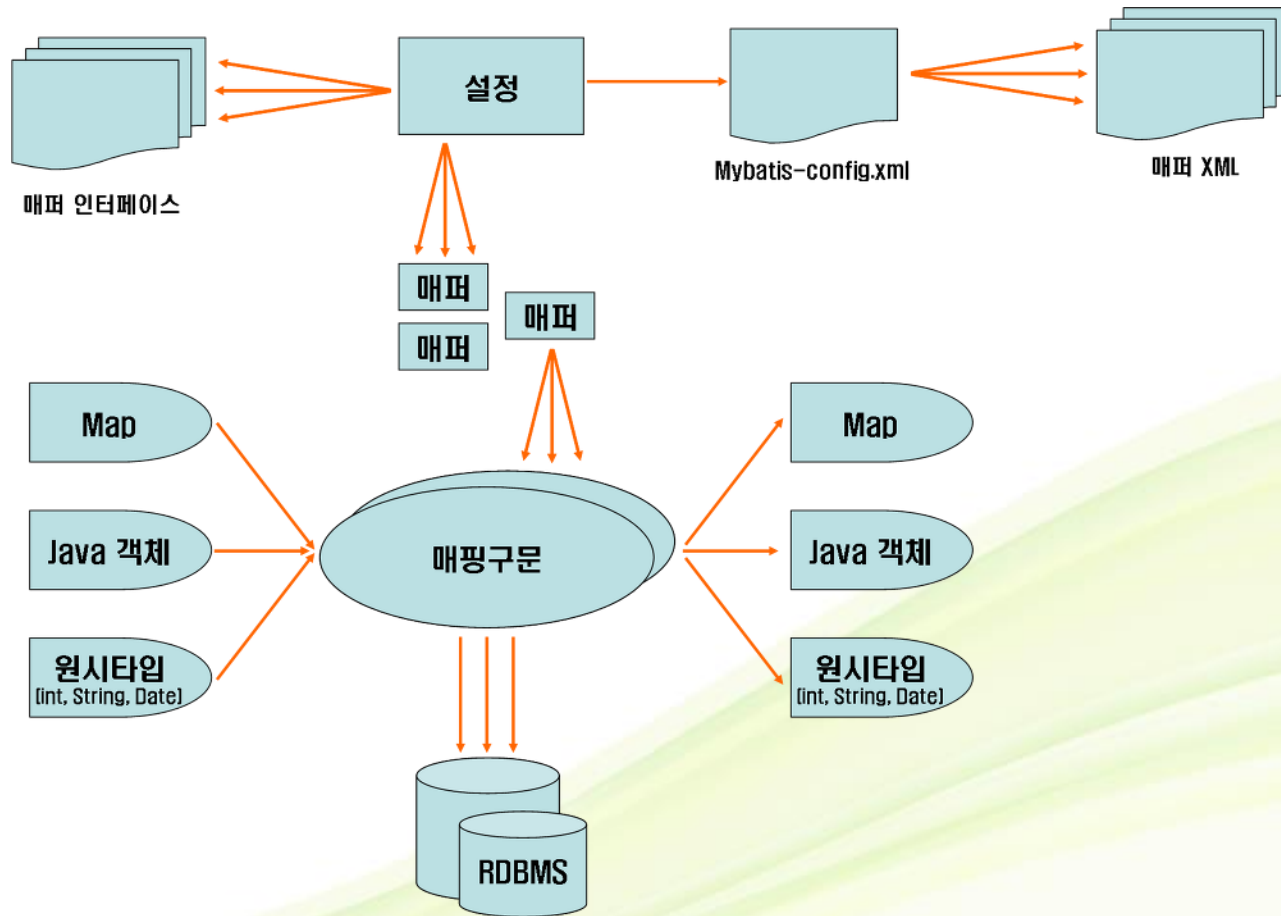
- ❖ JDBC 프레임워크
- ❖ SQL은 개발자가 작성하고 MyBatis는 JDBC를 이용해서 이를 실행하는 역할
- ❖ ORM(Object Relation Mapping)이 아닌 SQL 매퍼
- ❖ ORM은 데이터베이스 객체를 자바 객체로 매핑해서 객체 간의 관계를 바탕으로 SQL을 자동으로 생성해주는데 대표적인 프레임워크가 하이버네이트
- ❖ SQL 문장과 파라미터를 PreparedStatement를 이용해서 자동 매핑
- ❖ Select 구문의 결과를 객체(DTO 또는 Map)로 자동 매핑
- ❖ 트랜잭션 관리가 용이
- ❖ 스프링과 같은 외부 트랜잭션 관리자를 사용할 수 있도록 한다.
- ❖ 스프링 연동 모듈을 제공해서 스프링과 연동 가능
- ❖ JVM위에서 동작하는 함수형 언어인 스칼라와 마이크로소프트 사의 닷넷 그리고 루비를 지원

MyBatis

❖ 마이바티스 구조

- 설정 파일: 데이터베이스 설정과 트랜잭션 등 마이바티스가 동작하는 규칙을 정의
- 매퍼: SQL을 XML에 정의한 매퍼 XML 파일 또는 SQL을 인터페이스 메소드에 annotation으로 정의한 매퍼 인터페이스
- 결과 매핑과 매핑 구문: 조회 결과를 자바 객체에 설정하는 규칙을 나타내는 결과 매핑과 SQL을 XML에 정의한 매핑 구문
- 지원하는 파라미터 타입: Map 객체, 자바 클래스, 원시 타입(Wrapper Class, String)
- 지원하는 결과 타입: 파라미터 타입과 동일

MyBatis



MyBatis 환경 설정

```
<?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">
      <!-- ① 트랜잭션 관리자 -->
      <transactionManager type="JDBC" />
      <!-- ② 데이터베이스 설정 -->
      <dataSource type="POOLED">
        <property name="driver" value="oracle.jdbc.driver.OracleDriver" />
        <property name="url" value="jdbc:oracle:thin:@localhost:1521:mybatis" />
        <property name="username" value="mybatis" />
        <property name="password" value="asdf" />
      </dataSource>
    </environment>
  </environments>
  <!-- ③ 매퍼정보 설정 -->
  <mappers>
    <mapper resource="ldg/mybatis/repository/mapper/CommentMapper.xml" />
  </mappers>
</configuration>
```

MyBatis 환경 설정

① 트랜잭션 관리자

마이바티스는 두가지 타입의 TransactionManager를 제공

- JDBC - 이 설정은 간단하게 JDBC 커밋과 롤백을 처리하기 위해 사용

- MANAGED

- ❖ 이 설정은 어떤 것도 하지 않습니다.
- ❖ 결코 커밋이나 롤백을 하지 않습니다.
- ❖ 대신 컨테이너가 트랜잭션의 모든 생명주기를 관리합니다.

- 마이바티스를 스프링과 함께 사용하는 경우에는 구성할 필요가 없습니다 스프링 모듈 자체의 설정 때문에 어떤 TransactionManager 이전에 설정된 구성을 무시합니다.

MyBatis 환경 설정

② 데이터베이스 설정

dataSource엘리먼트는 표준 JDBC DataSource인터페이스를 사용하여 JDBC Connection객체의 소스를 설정합니다.

❖ UNPOOLED - 이 구현체는 매번 요청에 대해 커넥션을 열고 닫는 간단한 DataSource로 느리기는 하지만 성능을 크게 필요로 하지 않는 간단한 애플리케이션에 적합하며 5개의 프로퍼티로 설정한다.

- driver - JDBC드라이버의 패키지 경로를 포함한 결제 자바 클래스명
- url - 데이터베이스 인스턴스에 대한 JDBC URL
- username - 데이터베이스에 로그인 할 때 사용할 사용자명
- password - 데이터베이스에 로그인 할 때 사용할 패스워드

MyBatis 환경 설정

- ❖ POOLED - DataSource에 풀링이 적용된 JDBC 커넥션을 위한 구현체로 새로운 Connection 인스턴스를 생성하기 위해 매번 초기화하는 것을 피하게 해주는 방식으로 빠른 응답을 요구하는 웹 애플리케이션에서 사용
 - poolMaximumActiveConnections - 주어진 시간에 존재할 수 있는 활성화된(사용중인) 커넥션의 수. 디폴트는 10
 - poolMaximumIdleConnections - 주어진 시간에 존재할 수 있는 유힬 커넥션의 수
 - poolTimeToWait - 풀이 로그 상태를 출력하고 비정상적으로 긴 경우 커넥션을 다시 얻으려고 시도하는 로우 레벨 설정. 디폴트는 20000ms(20 초)
 - poolPingEnabled - 핑쿼리를 사용할지 말지를 결정. 사용한다면 오류가 없는(그리고 빠른) SQL 을 사용하여 poolPingQuery 프로퍼티를 설정해야 합니다. 디폴트는 false
 - poolPingConnectionsNotUsedFor - poolPingQuery가 얼마나 자주 사용될지 설정하는 것으로 디폴트는 0

MyBatis 환경 설정

- 데이터베이스 접속 드라이버와 URL작성 방법

Oracle

드라이버 클래스명: oracle.jdbc.driver.OracleDriver

JDBC URL: jdbc:oracle:thin:@localhost:1521:SID이름

MySQL

드라이버 클래스명: com.mysql.jdbc.Driver

JDBC URL: jdbc:mysql://localhost:3306/데이터베이스이름

MSSQL

드라이버 클래스명: com.microsoft.jdbc.sqlserver.SQLServerDriver

JDBC URL: jdbc:microsoft:sqlserver://203.247.166.172;databasename=데이터베이스이름

MyBatis 환경 설정

③ 매퍼정보 설정

- SQL을 선언해둔 XML 파일의 위치나 인터페이스 형태의 매퍼 위치를 지정해줘야 합니다.
- XML 위치는 클래스 패스를 기준으로 지정하면 됩니다.

```
<mappers>
```

```
  <mapper resource="org/mybatis/builder/AuthorMapper.xml"/>
```

```
  <mapper resource="org/mybatis/builder/BlogMapper.xml"/>
```

```
  <mapper resource="org/mybatis/builder/PostMapper.xml"/>
```

```
</mappers>
```

```
<mappers>
```

```
  <mapper class="org.mybatis.builder.AuthorMapper"/>
```

```
  <mapper class="org.mybatis.builder.BlogMapper"/>
```

```
  <mapper class="org.mybatis.builder.PostMapper"/>
```

```
</mappers>
```

MyBatis 환경 설정

※ MyBatis의 매퍼 파일

```
<?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="매퍼이름">
  <resultMap id="별명" type=" 실제 자료형">
    </resultMap>

  <select id=" 호출할 이름 " resultMap=" 결과 자료형" 또는 resultType="자료형" >
    select 구문
  </select>
  <insert id=" 호출할 이름" parameterType=" 매개변수 자료형">
    INSERT INTO
      AddressBook
      VALUES ({프로퍼티이름 또는 키이름})
  </insert>

</mapper>
```

MyBatis 환경 설정

※ MyBatis의 자료형 매핑

Number – Integer, Double, Float, Byte, Short, Long, BigDecimal

Char, Varchar, Clob – String

Date – java.sql.Date, java.util.Date

Time – java.sql.Time, java.util.Date

TimeStamp – java.sql.Time, stamp

Blob – byte []

MyBatis 환경 설정

※ MyBatis의 매퍼 파일 작성 시 유의 사항

- wildcard 문자를 이용해서 검색하는 경우에 필드명이 "title" 이고 해당 필드에서 검색할 내용을 "keyword"를 포함하는 내용이라고 하면 아래와 같이 검색할 수 있습니다.

[MySQL]

title like CONCAT('%',#{keyword},'%')

[Oracle]

title like '%' || #{keyword} || '%'

[MSSQL]

title like '%' + #{keyword} + '%'

- MyBatis에서 SQL을 작성할 때 비교 연산자를 직접 사용하면 에러

Where #{num} < 5

Mybatis mapper.xml안에서 비교 연산자(<= , >=, <, >) 을 써야 할 상황에서, 위와 같이 연산자를 직접 쓰게되면 오류 발생합니다. < 를 태그의 시작으로 인식해서 그렇습니다.

Where #{num} <![CDATA[<]> 5

MyBatis 환경 설정

- ❖ 모든 MyBatis 애플리케이션은 SqlSessionFactory 인스턴스를 사용합니다.
- ❖ SqlSessionFactory 인스턴스는 SqlSessionFactoryBuilder를 사용하여 만들 수 있습니다.
- ❖ SqlSessionFactoryBuilder는 XML설정파일에서 SqlSessionFactory 인스턴스를 빌드할 수 있습니다.

```
String resource = "환경설정파일 경로";  
Reader reader = Resources.getResourceAsReader(resource);  
SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(reader);
```

위의 Factory를 이용해서 SqlSession 객체를 생성합니다.
위의 객체를 가지고 openSession()를 호출하면 SqlSession객체가 생성됩니다.

```
SqlSession sqlSession = sqlSessionFactory.openSession();
```

MyBatis 환경 설정

❖ sql 실행 메소드 호출

- ✓ sqlSession.insert, delete, update, selectOne, selectList(매퍼이름.id, 매개변수)를 호출하면 됩니다.
- ✓ 이 때 insert와 delete, update는 영향 받은 행의 개수를 정수로 리턴합니다.
- ✓ selectOne은 설정한 resultType으로 리턴
- ✓ selectList는 설정한 resultType의 List로 리턴
- ✓ selectOne 메소드는 결과가 없거나 2개 이상으로 나오면 예외를 발생
- ✓ Map으로 select 구문의 결과를 받는 경우 컬럼 이름이 키가 되고 데이터가 값
- ✓ 오라클의 데이터의 타입이 Number이면 Map에서는 BigDecimal로 저장
- ✓ Map으로 select 구문의 결과를 받는 경우 키의 이름은 오라클은 모두 대문자 MySQL은 소문자로 저장
- ✓ insert, delete, update 작업 시 autoCommit 되지 않으므로 commit()을 호출해서 직접 commit을 수행 해 주어야 합니다.

MyBatis 환경 설정

- ✓ sql 사용이 종료되면 close()를 호출해서 닫아 주어야 합니다.
- ✓ Java 1.7 & MyBatis 3.2 이상의 버전을 사용하면 아래처럼 작성해도 됩니다.

```
try (SqlSession session = sqlSessionFactory.openSession()) {  
    session.insert(...);  
    session.commit();  
}
```

MyBatis

1. MyBatis는 개발자가 지정한 SQL, 저장프로시저 그리고 몇가지 고급 매핑을 지원하는 퍼시스턴스 프레임워크이다. MyBatis는 JDBC코드와 수동으로 셋팅하는 파라미터와 결과 매핑을 제거한다. MyBatis는 데이터베이스 레코드에 원시타입과 Map인터페이스 그리고 자바 POJO를 설정하고 매핑하기 위해 XML과 애노테이션을 사용할 수 있다
2. 모든 MyBatis 애플리케이션은 SqlSessionFactory 인스턴스를 사용한다. SqlSessionFactory 인스턴스는 SqlSessionFactoryBuilder 를 사용하여 만들수 있다. SqlSessionFactoryBuilder 는 XML 설정파일에서 SqlSessionFactory 인스턴스를 빌드할 수 있다.

XML에서 SqlSessionFactory 빌드하기

```
<?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>
```

XML을 사용하지 않고 SqlSessionFactory 빌드하기

1. XML보다 자바를 사용해서 직접 설정하길 원한다면, XML파일과 같은 모든 설정을 제공하는 Configuration 클래스를 사용하면 된다.

```
DataSource dataSource = BlogDataSourceFactory.getBlogDataSource();  
TransactionFactory transactionFactory = new JdbcTransactionFactory();  
Environment environment = new Environment("development", transactionFactory, dataSource);  
Configuration configuration = new Configuration(environment);  
configuration.addMapper(BlogMapper.class);  
SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(configuration);
```

이 설정에서 추가로 해야 할 일은 Mapper 클래스를 추가하는 것이다. Mapper 클래스는 SQL 매핑 애노테이션을 가진 자바 클래스이다. 어쨌든 자바 애노테이션의 몇가지 제약과 몇가지 설정 방법의 복잡함에도 불구하고, XML매핑은 세부적인 매핑을 위해 언제나 필요하다.

SqlSessionFactory 에서 SqlSession 만들기

1. SqlSessionFactory 이름에서 보듯이, SqlSession 인스턴스를 만들수 있다. SqlSession 은 데이터베이스에 대해 SQL 명령어를 실행하기 위해 필요한 모든 메서드를 가지고 있다. 그래서 SqlSession 인스턴스를 통해 직접 SQL구문을 실행할 수 있다.

```
SqlSession session = sqlMapper.openSession();
try {
    Blog blog =
        (Blog) session.selectOne( "org.mybatis.example.BlogMapper.selectBlog", 101);
} finally {
    session.close();
}
```

주어진 SQL구문의 파라미터와 리턴값을 설명하는 인터페이스(예를 들면, BlogMapper.class)를 사용하여, 문자열 처리 오류나 타입 캐스팅 오류 없이 좀더 타입에 안전하고 깔끔하게 실행할 수 있다.

```
SqlSession session = sqlSessionFactory.openSession();
try {
    BlogMapper mapper = session.getMapper(BlogMapper.class);
    Blog blog = mapper.selectBlog(101);
} finally {
    session.close();    }
```

SessionFactory

한번 만든뒤, SqlSessionFactory는 애플리케이션을 실행하는 동안 존재해야만 한다. 그래서 삭제하거나 재생성할 필요가 없다. 애플리케이션이 실행되는 동안 여러 차례 SqlSessionFactory 를 다시 빌드하지 않는 것이 가장 좋은 형태이다. 가장 간단한 방법은 싱글턴 패턴이나 static 싱글턴 패턴을 사용하는 것이다.

SqlSession

각각의 스레드는 자체적으로 SqlSession 인스턴스를 가져야 한다. SqlSession 인스턴스는 공유되지 않고 스레드에 안전하지도 않다. 그러므로 가장 좋은 스코프는 요청 또는 메서드 스코프이다. SqlSession 을 static 필드나 클래스의 인스턴스 필드로 지정해서는 안된다. HTTP 요청을 받을때마다 만들고, 응답을 리턴할때마다 SqlSession을 닫을 수 있다. SqlSession을 닫는 것은 중요하다. 언제나 finally 블록에서 닫아야만 한다.

```
try { // do work
} finally { session.close(); }
```

Mapper 인스턴스

Mapper는 매핑된 구문을 바인딩 하기 위해 만들어야 할 인터페이스이다. mapper 인터페이스의 인스턴스는 SqlSession에서 생성한다.

```
SqlSession session = sessionFactory.openSession();
try { BlogMapper mapper = session.getMapper(BlogMapper.class);
      // do work
} finally { session.close(); }
```

매퍼 설정 XML

MyBatis XML 설정파일은 다양한 셋팅과 프로퍼티를 가진다.

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

properties

이 설정은 외부에 옮길 수 있다. 자바 프로퍼티 파일 인스턴스에 설정할 수도 있고, properties 요소의 하위 요소에 둘수도 있다. 예를 들면:

```
<properties resource="org/mybatis/example/config.properties">
  <property name="username" value="dev_user"/>
  <property name="password" value="F2Fa3!33TYyg"/>
</properties>
```

속성들은 파일 도처에 둘수도 있다. 예를 들면:

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

속성은 SqlSessionFactory.build() 메서드에 전달될 수 있다.

```
SqlSessionFactory factory = sqlSessionFactoryBuilder.build(reader, props);
// ... or ...
```

```
SqlSessionFactory factory =
    sqlSessionFactoryBuilder.build(reader, environment, props);
```

속성이 한개 이상 존재한다면, MyBatis는 일정한 순서로 로드한다.:

1. properties 요소에 명시된 속성
2. properties요소의 클래스패스 자원이나 url속성
3. 메서드 파라미터로 전달된 속성을 읽는다

settings

셋팅	설명	사용가능한 값들	디폴트
cacheEnabled	설정에서 각 mapper 에 설정된 캐시를 전역적으로 사용할지 말지에 대한 여부	true false	true
lazyLoadingEnabled	늦은 로딩을 사용할지에 대한 여부. 사용하지 않는다면 모두 즉시 로딩할 것이다.	true false	true
aggressiveLazyLoading	활성화 상태로 두게 되면 늦은(lazy) 로딩 프로퍼티를 가진 객체는 호출에 따라 로드될 것이다. 반면에 개별 프로퍼티는 요청할때 로드된다.	true false	true
multipleResultSetsEnabled	한개의 구문에서 여러개의 ResultSet 을 허용할지의 여부(드라이버가 해당 기능을 지원해야 함)	true false	true
useColumnLabel	칼럼명 대신에 칼럼라벨을 사용. 드라이버마다 조금 다르게 작동한다. 문서와 간단한 테스트를 통해 실제 기대하는 것처럼 작동하는지 확인해야 한다.	true false	true
useGeneratedKeys	생성키에 대한 JDBC 지원을 허용. 지원하는 드라이버가 필요하다. true 로 설정하면 생성키를 강제로 생성한다. 일부 드라이버(예를들면, Derby)에서는 이 설정을 무시한다.	true false	False
autoMappingBehavior	MyBatis 가 칼럼을 필드/프로퍼티에 자동으로 매핑할지와 방법에 대해 명시. PARTIAL 은 간단한 자동매핑만 할뿐, 내포된 결과에 대해서는 처리하지 않는다. FULL 은 처리가능한 모든 자동매핑을 처리한다.	NONE, PARTIAL, FULL	PARTIAL
defaultExecutorType	디폴트 실행자(executor) 설정. SIMPLE 실행자는 특별히 하는 것이 없다. REUSE 실행자는 PreparedStatement 를 재사용한다. BATCH 실행자는 구문을 재사용하고 수정을 배치처리한다.	SIMPLE REUSE BATCH	SIMPLE
defaultStatementTimeout	데이터베이스로의 응답을 얼마나 오래 기다릴지를 판단하는 타임아웃을 셋팅	양수	셋팅되지 않음(null)


```
<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="enhancementEnabled" value="false"/>
<setting name="defaultExecutorType" value="SIMPLE"/>
<setting name="defaultStatementTimeout" value="25000"/>
</settings>
```

typeAliases

타입 별칭은 자바 타입에 대한 좀더 짧은 이름이다. 오직 XML 설정에서만 사용되며, 타이핑을 줄이기 위해 존재한다. 예를들면:

```
<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>
```

이 설정에서, "Blog" 는 도처에서 "domain.blog.Blog" 대신 사용될 수 있다.

select

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

```
</select>
```

이 구문의 이름은 selectPerson 이고 int 타입의 파라미터를 가진다. 그리고 결과 데이터는 HashMap에 저장된다. 파라미터 표기법을 보자.

#{id}

이 표기법은 MyBatis 에게 PreparedStatement 파라미터를 만들도록 지시한다. JDBC 를 사용할 때 PreparedStatement에는 "?" 형태로 파라미터가 전달된다.

결과적으로 위 설정은 아래와 같이 작동하게 되는 셈이다.

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

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

속성	설명
id	구문을 찾기 위해 사용될 수 있는 명명공간내 유일한 구분자
parameterType	구문에 전달될 파라미터의 패키지 경로를 포함한 전체 클래스명이나 별칭
parameterMap	외부 parameterMap 을 찾기 위한 비권장된 접근방법. 인라인 파라미터 매핑과 parameterType 을 대신 사용하라.
resultType	이 구문에 의해 리턴되는 기대타입의 패키지 경로를 포함한 전체 클래스명이나 별칭. collection 이 경우, collection 타입 자체가 아닌 collection 이 포함된 타입이 될 수 있다. resultType 이나 resultMap 을 사용하라.
resultMap	외부 resultMap 의 참조명. 결과맵은 MyBatis 의 가장 강력한 기능이다. resultType 이나 resultMap 을 사용하라.
flushCache	이 값을 true 로 셋팅하면, 구문이 호출될때마다 캐시가 지워질것이다(flush). 디폴트는 false 이다.
useCache	이 값을 true 로 셋팅하면, 구문의 결과가 캐시될 것이다. 디폴트는 true 이다.
timeout	예외가 던져지기 전에 데이터베이스의 요청 결과를 기다리는 최대시간을 설정한다. 디폴트는 셋팅하지 않는 것이고 드라이버에 따라 다소 지원되지 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. MyBatis 에게 Statement, PreparedStatement 또는 CallableStatement 를 사용하게 한다. 디폴트는 PREPARED 이다.
resultSetType	FORWARD_ONLY SCROLL_SENSITIVE SCROLL_INSENSITIVE 중 하나를 선택할 수 있다. 디폴트는 셋팅하지 않는 것이고 드라이버에 따라 다소 지원되지 않을 수 있다.

insert, update, delete

```
<insert id="insertAuthor" parameterType="domain.blog.Author"
flushCache="true" statementType="PREPARED" keyProperty=""
keyColumn="" useGeneratedKeys="" timeout="20000">
<update id="insertAuthor" parameterType="domain.blog.Author"
flushCache="true" statementType="PREPARED" timeout="20000">
<delete id="insertAuthor" parameterType="domain.blog.Author"
flushCache="true" statementType="PREPARED" timeout="20000">
```

속성	설명
id	구문을 찾기 위해 사용될 수 있는 명명공간내 유일한 구분자
parameterType	구문에 전달될 파라미터의 패키지 경로를 포함한 전체 클래스명이나 별칭
parameterMap	외부 parameterMap 을 찾기 위한 비권장된 접근방법. 인라인 파라미터 매핑과 parameterType 을 대신 사용하라.
flushCache	이 값을 true 로 셋팅하면, 구문이 호출될때마다 캐시가 지워질것이다(flush). 디폴트는 false 이다.
Timeout	예외가 던져지기 전에 데이터베이스의 요청 결과를 기다리는 최대시간을 설정한다. 디폴트는 셋팅하지 않는 것이고 드라이버에 따라 다소 지원되지 않을 수 있다.
statementType	STATEMENT, PREPARED 또는 Callable 중 하나를 선택할 수 있다. MyBatis 에게 Statement, PreparedStatement 또는 CallableStatement 를 사용하게 한다. 디폴트는 PREPARED 이다.
useGeneratedKeys	(입력(insert)에만 적용) 데이터베이스에서 내부적으로 생성한 키(예를 들어, MySQL 또는 SQL Server 와 같은 RDBMS 의 자동 증가 필드)를 받는 JDBC getGeneratedKeys 메서드를 사용하도록 설정하다. 디폴트는 false 이다.
keyProperty	(입력(insert)에만 적용) getGeneratedKeys 메서드나 insert 구문의 selectKey 하위 요소에 의해 리턴된 키를 셋팅할 프로퍼티를 지정. 디폴트는 셋팅하지 않는 것이다.
keyColumn	(입력(insert)에만 적용) 생성키를 가진 테이블의 칼럼명을 셋팅. 키 칼럼이 테이블이 첫번째 칼럼이 아닌 데이터베이스(PostgreSQL 처럼)에서만 필요하다.

Insert, update, delete 구문의 예제이다.

```
<insert id="insertAuthor" parameterType="domain.blog.Author">
```

```
insert into Author (id,username,password,email,bio)  
values ({id},{username},{password},{email},{bio})
```

```
</insert>
```

```
<update id="updateAuthor" parameterType="domain.blog.Author">
```

```
update Author set username = {username}, password = {password},  
email = {email}, bio = {bio} where id = {id}
```

```
</update>
```

```
<delete id="deleteAuthor" parameterType="int">
```

```
delete from Author where id = {id}
```

```
</delete>
```

insert 는 key 생성과 같은 기능을 위해 몇가지 추가 속성과 하위 요소를 가진다.
먼저, 사용하는 데이터베이스가 자동생성키(예를 들면, MySQL과 SQL 서버)를 지원한다면, useGeneratedKeys="true" 로 설정하고 대상 프로퍼티에 keyProperty 를 셋팅할 수 있다. 예를 들어, Author 테이블이 id칼럼에 자동생성키를 적용했다고 하면, 구문은 아래와 같은 형태일 것이다.

```
<insert id="insertAuthor" parameterType="domain.blog.Author"
```

```
useGeneratedKeys="true" keyProperty="id">
```

```
insert into Author (username,password,email,bio)  
values ({username},{password},{email},{bio})
```

```
</insert>
```

iBatis 는 자동생성키 칼럼을 지원하지 않는 오라클 등은 다른 방법 또한 제공한다.

```
<insert id="InsertOrganization" parameterClass="Organization" resultClass="int">  
  <selectKey property="Id" type="pre" resultClass="int">  
    select seq_orgs.nextval as value from dual </selectKey>  
    INSERT INTO Organizations (Org_Id, Org_Code, Org_Name) VALUES (#Id#,  
      #Code#, #Name#)  
  </insert>
```

위 예제에서, selectKey 구문이 먼저 실행되고, Author id프로퍼티에 셋팅된다. 그리고 나서 insert 구문이 실행된다. 이걸 복잡한 자바코드 없이도 데이터베이스에 자동생성키의 행위와 비슷한 효과를 가지도록 해준다.

myBatis

```
<selectKey keyProperty="id" resultType="int" order="BEFORE"  
statementType="PREPARED">
```

속성	설명
keyProperty	selectKey 구문의 결과가 셋팅될 대상 프로퍼티
resultType	결과의 타입. MyBatis 는 이 기능을 제거할 수 있지만 추가하는게 문제가 되지는 않을것이다. MyBatis 는 String 을 포함하여 키로 사용될 수 있는 간단한 타입을 허용한다.
order	BEFORE 또는 AFTER 를 셋팅할 수 있다. BEFORE 로 셋팅하면, 키를 먼저 조회하고 그 값을 keyProperty 에 셋팅한 뒤 insert 구문을 실행한다. AFTER 로 셋팅하면, insert 구문을 실행한 뒤 selectKey 구문을 실행한다. Oracle 과 같은 데이터베이스에서는 insert 구문 내부에서 일관된 호출 형태로 처리한다.
statementType	위 내용과 같다. MyBatis 는 Statement, PreparedStatement 그리고 CallableStatement 을 매핑하기 위해 STATEMENT, PREPARED 그리고 CALLABLE 구문타입을 지원한다.

pom.xml

1. Mybatis설정 추가

```
<dependency>
  <groupId>org.mybatis</groupId>
  <artifactId>mybatis</artifactId>
  <version>3.2.2</version>
</dependency>
<dependency>
  <groupId>org.mybatis</groupId>
  <artifactId>mybatis-spring</artifactId>
  <version>1.2.0</version>
</dependency>
```

회원관리

```
create table member3 (  
    id varchar2(20) primary key,  
    email varchar2(30),  
    password varchar2(100),  
    name varchar2(30),  
    fileName varchar2(50),  
    del char(1) default 'n',  
    regdate date
```

```
);
```

```
drop table member3;
```

```
drop table memberphoto;
```

```
create table memberphoto (  
    num number(10) primary key,  
    id varchar2(20) references member3(id),  
    fileName varchar2(50)
```

```
);
```

```
create sequence memberphoto_seq;
```

```
create or replace FUNCTION get_seq  
    RETURN NUMBER
```

```
IS
```

```
BEGIN
```

```
    RETURN memberphoto_seq.nextval;
```

```
END;
```

```
/
```


SessionChk

```
public class SessionChk extends HandlerInterceptorAdapter{
    public boolean preHandle(HttpServletRequest request, HttpServletResponse
response, Object handler) throws Exception {
        HttpSession session = request.getSession();
        if (session.getAttribute("id") == null) {
            response.sendRedirect("loginForm.do");
            return false;
        }
        return true;
    }
}
```

```
<beans:bean id="sc" class="com.ch.mybatis.service.SessionCheck"> </beans:bean>
<interceptors>
    <interceptor>
        <mapping path="/main.html"/>
        <mapping path="/view.html"/>
        <mapping path="/update.html"/>
    <mapping path="/updateForm.html"/>
        <mapping path="/delete.html"/>
        <beans:ref bean="sc"/>
    </interceptor>
</interceptors>
```

Member

```
package com.ch.mybatis.model;
import java.sql.Date;
import org.springframework.web.multipart.MultipartFile;
import lombok.Data;
@Data
public class Member {
    private String id;
    private String password;
    private String fileName;
    private Date regdate;
    // upload용 DB에서는 사용 안함
    private MultipartFile file;
    private String email;
    private String name;
    private String del;
}

package com.ch.mybatis.model;
import lombok.Data;
@Data
public class MemberPhoto {
    private int num;
    private String id;
    private String fileName;
}
```

회원관리

```
<?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>
    <typeAliases>
        <typeAlias alias="member" type="com.ch.mybatis.model.Member" />
        <typeAlias alias="memberphoto"
type="com.ch.mybatis.model.MemberPhoto" />
    </typeAliases>
    <mappers>
        <mapper resource="member.xml" />
    </mappers>
</configuration>
```

db.properties

```
driver=oracle.jdbc.driver.OracleDriver
url=jdbc:oracle:thin:@127.0.0.1:1521:xe
userId=scott
password=tiger
maxPoolSize=20
```

회원관리

```
<?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="members">
    <!-- Use type aliases to avoid typing the full classname every time. -->
    <resultMap id="memberResult" type="member">
        <result property="id" column="id" />
        <result property="email" column="email" />
        <result property="password" column="password" />
        <result property="name" column="name" />
        <result property="fileName" column="fileName" />
        <result property="del" column="del" />
        <result property="regdate" column="regdate" />
    </resultMap>
    <select id="list" resultMap="memberResult">
        select * from member3 order by id
    </select>
    <select id="select" parameterType="string" resultType="member">
        select *
        from member3 where id=#{id}
    </select>
```

회원관리

```
<insert id="insert" parameterType="member">
    insert into member3 values
        ({id},{email},{password},{name},{fileName},'n',sysdate)
</insert>
<update id="update" parameterType="member">
    update member3 set email=#{email}, password=#{password},name=#{name}
    <if test="fileName!=null">
        ,fileName=#{fileName}
    </if>
    where id=#{id}
</update>
<update id="delete" parameterType="string">
    update member3 set del='y' where      id=#{id}
</update>
<insert id="insertFileList" parameterType="java.util.List">
    INSERT ALL
    <foreach collection="list" item="item">
        INTO memberphoto          (num, id, fileName)
        VALUES (memberphoto_seq.nextval,
            ${item.id}, '${item.fileName}' )
    </foreach>
    select * from dual
</insert>
</mapper>
```

root-context.xml

beans, context, security namespaces에서 체크

```
<context:property-placeholder location="classpath:db.properties" />
<!-- 암호화 -->
<bean id="passwordEncoder"
class="org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder"></bean>
<!-- 파일 업로드 용 -->
<bean id="multipartResolver"
class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
</bean>
<!-- myBatis시작 -->
<bean id="sqlSessionFactory"
class="org.mybatis.spring.SqlSessionFactoryBean">
<property name="dataSource" ref="dataSource" />
<property name="configLocation"
value="classpath:configuration.xml" />
</bean>
<bean id="session" class="org.mybatis.spring.SqlSessionTemplate">
<constructor-arg index="0" ref="sqlSessionFactory" />
</bean>
```

root-context.xml

<!-- DB연결 풀 -->

<!--

<bean id="dataSource"

class="com.mchange.v2.c3p0.ComboPooledDataSource"

destroy-method="close">

<property name="driverClass" value="\${driver}" />

<property name="jdbcUrl" value="\${url}" />

<property name="user" value="\${userId}" />

<property name="password" value="\${password}" />

<property name="maxPoolSize" value="\${maxPoolSize}" />

</bean> -->

<bean id="hikariConfig"

class="com.zaxxer.hikari.HikariConfig">

<property name="driverClassName" value="\${driver}" />

<property name="jdbcUrl" value="\${url}" />

<property name="username" value="\${userId}" />

<property name="password" value="\${password}" />

</bean>

<bean id="dataSource" class="com.zaxxer.hikari.HikariDataSource"

destroy-method="close">

<constructor-arg ref="hikariConfig"> </constructor-arg>

</bean>

</beans>

servlet-context.xml

```
<annotation-driven />
<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.ch.mybatis" />
<beans:bean id="sc" class="com.ch.mybatis.service.SessionChk"> </beans:bean>
    <interceptors>
        <interceptor>
            <mapping path="/main.html"/>
            <mapping path="/view.html"/>
            <mapping path="/updateForm.html"/>
            <mapping path="/update.html"/>
            <mapping path="/delete.html"/>
            <beans:ref bean="sc"/>
        </interceptor>
    </interceptors>
```


회원 가입

아이디 	<input type="text" value="k1"/>	<input type="button" value="중복체크"/>
암호 	<input type="password"/>	
암호확인 	<input type="password"/>	
이름	<input type="text" value="주니"/>	
이메일 	<input type="text" value="kbj010@hanmail.net"/>	
사진 	<input type="button" value="파일 선택"/> a.jpg	
<input type="button" value="제출"/>		

회원관리

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ include file="header.jsp" %>
<!DOCTYPE html><html><head><meta charset="UTF-8">
<title>Insert title here</title>
<script type="text/javascript">
    function idChk() {
        if (!frm.id.value) {      alert("아이디 입력후에 체크하시오");
            frm.id.focus(); return false;    }
        $.post('idChk.html', 'id='+frm.id.value, function(data) {
            $('#disp').html(data);
        });
    }
    function chk() {
        if (frm.password.value != frm.password2.value) {
            alert('암호와 암호확인이 다릅니다');
            frm.password.force();
            frm.password.value = "";
            return false;
        }
    }
</script></head><body>
```

회원관리

```
<div class="container" align="center">
    <h2 class="text-primary">회원 가입</h2>
    <form action="join.html" method="post" name="frm",
    enctype="multipart/form-data" onsubmit="return chk()">
    <table class="table table-bordered">
        <tr><td>아이디<span class="glyphicon glyphicon-user"/></td>
        <td><input type="text" name="id" required="required" autofocus="autofocus">
        <input type="button" onclick="idChk()" class="btn btn-info btn-sm"
        value="중복체크"><span id="disp" class="err"></span></td></tr>
        <tr><th>암호<span class="glyphicon glyphicon-lock"/></th>
        <td><input type="password" name="password" required="required"></td></tr>
        <tr><th>암호확인<span class="glyphicon glyphicon-lock"/></th>
        <td><input type="password" name="password2"
        required="required"></td></tr>
        <tr><th>이름</th><td><input type="text" name="name"
        required="required"></td></tr>
        <tr><th>이메일<span class="glyphicon glyphicon-envelope"/>
        </th><td><input type="email" name="email"
        required="required"></td></tr>
        <tr><th>사진<span class="glyphicon glyphicon-picture"/>
        </th><td><input type="file" name="file"
        required="required"></td></tr>
        <tr><th colspan="2"><input type="submit"></th></tr>
    </table></form></div></body></html>
```

회원관리

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ include file="header.jsp" %>
<!DOCTYPE html> <html> <head> <meta charset="UTF-8">
<title>Insert title here</title> </head> <body>
<c:if test="${result > 0}">
    <script type="text/javascript">
        alert("입력완료 ㅋㅋ");          location.href="loginForm.html";
    </script>
</c:if>
<c:if test="${result == -1}">
    <script type="text/javascript">
        alert("이미 사용중인 ID라는 와 입력해 으이그 !!");
        history.go(-1);
    </script>
</c:if>
<c:if test="${result == 0}">
    <script type="text/javascript">
        alert("입력 실패 ㅠㅠ");
        history.go(-1);
    </script>
</c:if>
</body> </html>
```

MemberController

@Controller

```
public class MemberController {  
    @Autowired  
    private MemberService ms;  
    @RequestMapping("joinForm")  
    public String joinForm() {  
        return "joinForm";  
    }  
    @RequestMapping(value="idChk",  
                    produces="text/html;charset=utf8")  
    @ResponseBody  
    public String idChk(String id) {  
        String msg = "";  
        Member member = ms.select(id);  
        if (member==null) msg="사용가능합니다";  
        else msg="다른 아이디를 선택하세요";  
        return msg;  
    }  
}
```

MemberController

@RequestMapping("join")

public String join(Member member, Model model, HttpSession session) throws
IOException {

int result = 0;

Member mem = ms.select(member.getId());

if (mem == null) {

String fileName = member.getFile().getOriginalFilename();

String real =

session.getServletContext().getRealPath("/upload");

FileOutputStream fos =

new FileOutputStream(new File(real+"/"+fileName));

fos.write(member.getFile().getBytes());

fos.close();

member.setFileName(fileName);

result = ms.insert(member);

} else result = -1;

model.addAttribute("result", result);

return "join";

}

로그인



확인

회원가입

loginform

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ include file="header.jsp" %>
<!DOCTYPE html> <html> <head> <meta charset="UTF-8">
<title>Insert title here</title> </head> <body>
<div class="container" align="center">
    <h2 class="text-primary">로그인</h2>
<form action="login.html">
<table class="table table-striped table-bordered">
    <tr> <td>아이디</td> <td><input type="text" name="id"
        required="required" autofocus="autofocus"> </td> </tr>
    <tr> <td>암호</td> <td><input type="password" name="password"
        required="required"> </td> </tr>
    <tr> <td colspan="2"><input type="submit" value="확인"
        class="btn btn-warning"> </td> </tr>
</table>
    <a href="joinForm.html" class="btn btn-info">회원가입</a>
</form>
</div>
</body> </html>
```


login

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ include file="header.jsp" %>
<!DOCTYPE html> <html> <head> <meta charset="UTF-8">
<title>Insert title here</title> </head> <body>
<c:if test="${result > 0}">
    <script type="text/javascript">
        alert("로그인 성공 ㅋㅋ");
        location.href="main.html";
    </script>
</c:if>
<c:if test="${result == 0}">
    <script type="text/javascript">
        alert("암호도 몰라! 꺼져");
        history.go(-1);
    </script>
</c:if>
<c:if test="${result == -1}">
    <script type="text/javascript">
        alert("웬놈이냐 !!");
        history.go(-1);
    </script>
</c:if>
</body> </html>
```

회원관리

```
@RequestMapping("loginForm")
public String loginForm() {
    return "loginForm";
}

@RequestMapping("login")
public String login(Member member, Model model, HttpSession session) {
    int result = 0;
    Member mem = ms.select(member.getId());
    if (mem==null) result = -1;
    else if (mem.getPassword().equals(member.getPassword())) {
        result = 1;
        session.setAttribute("id", member.getId());
    }
    model.addAttribute("result", result);
    return "login";
}

@RequestMapping("main")
public String main(Model model, HttpSession session) {
    String id = (String)session.getAttribute("id");
    Member member = ms.select(id);
    model.addAttribute("member", member);
    return "main";
}
```

회원관리

주님님 환영

조회

수정

탈퇴

로그아웃

```
@RequestMapping("main")
public String main(Model model, HttpSession session) {
    String id = (String)session.getAttribute("id");
    Member member = ms.select(id);
    model.addAttribute("member", member);
    return "main";
}
```

```
<%@ include file="header.jsp" %>
<!DOCTYPE html> <html> <head> <meta charset="UTF-8">
<title>Insert title here</title> </head> <body>
<div class="container" align="center">
    <h2>${member.name }님 환영</h2>
<table class="table table-hover">
    <tr> <td> <a href="view.html" class="btn btn-info">조회</a> </td> </tr>
    <tr> <td> <a href="updateForm.html" class="btn btn-warning">수정
</a> </td> </tr>
    <tr> <td> <a href="delete.html" class="btn btn-danger">탈퇴</a> </td> </tr>
    <tr> <td> <a href="logout.html" class="btn btn-success">로그아웃
</a> </td> </tr>
</table> </div> </body> </html>
```

회원관리

회원 상세정보

아이디	k1
이름	주니
이메일	kbj010@hanmail.net
등록일	2020-08-01

사진



메인

view.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ include file="header.jsp" %>
<!DOCTYPE html> <html> <head> <meta charset="UTF-8">
<title>Insert title here</title> </head> <body>
<div class="container" align="center">
    <h2 class="text-primary">회원 상세정보</h2>
<table class="table table-striped">
    <tr> <td>아이디</td> <td>${member.id}</td> </tr>
    <tr> <td>이름</td> <td>${member.name}</td> </tr>
    <tr> <td>이메일</td> <td>${member.email}</td> </tr>
    <tr> <td>등록일</td> <td>${member.regdate}</td> </tr>
    <tr> <td>사진</td> <td>
</table>
<a href="main.html" class="btn btn-default">메인</a>
</div>
</body>
</html>
```

```
@RequestMapping("view")
public String view(Model model, HttpSession session) {
    String id = (String)session.getAttribute("id");
    Member member = ms.select(id);
    model.addAttribute("member", member);
    return "view";
}
```

회원관리

회원 정보수정

아이디 	k1
암호 	<input type="password"/>
암호확인 	<input type="password"/>
이름	<input type="text" value="주니"/>
이메일 	<input type="text" value="kbj010@hanmail.net"/>
사진 	<input type="button" value="파일 선택"/> 선택된 파일 없음



updateForm

```
@RequestMapping("updateForm")
public String updateForm(Model model, HttpSession session) {
    String id = (String)session.getAttribute("id");
    Member member = ms.select(id);
    model.addAttribute("member", member);
    return "updateForm";
}

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ include file="header.jsp" %>
<!DOCTYPE html> <html> <head> <meta charset="UTF-8">
<title>Insert title here</title>
<script type="text/javascript">
    function chk() {
        if (frm.password.value != frm.password2.value) {
            alert("암호가 암호확인과 다릅니다");
            frm.password.value = "";
            frm.password.focus();
            return false;
        }
    }
</script> </head> <body>
```


updateForm

```
<div class="container" align="center">
  <h2 class="text-primary">회원 정보수정 </h2>
  <form action="update.html" method="post" name="frm" enctype="multipart/form-data"
    onsubmit="return chk()">
    <input type="hidden" name="id" value="{member.id }">
  <table class="table table-bordered">
    <tr> <td>아이디 <span class="glyphicon glyphicon-user"/> </td> <td>{id}</td> </tr>
    <tr> <th>암호 <span class="glyphicon glyphicon-lock"/> </th>
      <td> <input type="password" name="password"
        required="required" autofocus="autofocus"> </td> </tr>
    <tr> <th>암호확인 <span class="glyphicon glyphicon-lock"/> </th>
      <td> <input type="password" name="password2"
        required="required"> </td> </tr>
    <tr> <th>이름 </th> <td> <input type="text" name="name"
      required="required" value="{member.name }"> </td> </tr>
    <tr> <th>이메일 <span class="glyphicon glyphicon-envelope"/>
      </th> <td> <input type="email" name="email"
        required="required" value="{member.email }"> </td> </tr>

    <tr> <th>사진 <span class="glyphicon glyphicon-picture"/>
      </th> <td> <input type="file" name="file"> </td> </tr>
    </td> <td colspan="2"> 
    <tr> <th colspan="2"> <input type="submit"> </th> </tr>
  </table> </form> </div> </body> </html>
```


update

```
@RequestMapping("update")
public String update(Member member, Model model, HttpSession session)
throws IOException {
    String fileName = member.getFile().getOriginalFilename();
    if (fileName!=null && !fileName.equals("")) {
        String real =
            session.getServletContext().getRealPath("/upload");
        FileOutputStream fos =
            new FileOutputStream(new File(real+"/"+fileName));
        fos.write(member.getFile().getBytes());
        fos.close();
        member.setFileName(fileName);
    }
    int result = ms.update(member);
    model.addAttribute("result", result);
    return "update";
}
```

update

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ include file="header.jsp" %>
<!DOCTYPE html> <html> <head> <meta charset="UTF-8">
<title>Insert title here</title> </head> <body>
<c:if test="${result > 0 }">
    <script type="text/javascript">
        alert("수정성공 대굴박");
        location.href="view.html";
    </script>
</c:if>
<c:if test="${result == 0 }">
    <script type="text/javascript">
        alert("똑바로 해, 이눔아 !!");
        history.go(-1);
    </script>
</c:if>
</body>
</html>
```

delete

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ include file="header.jsp" %>
<!DOCTYPE html> <html> <head> <meta charset="UTF-8">
<title>Insert title here</title> </head> <body>
<c:if test="${result > 0}">
    <script type="text/javascript">
        alert("탈퇴되었으니 꺼져 !!");
        location.href="loginForm.html";
    </script>
</c:if>
<c:if test="${result == 0}">
    <script type="text/javascript">
        alert("에효 실패 ㅠㅠ!");
        history.go(-1);
    </script>
</c:if>
</body>
</html>

@RequestMapping("delete")
public String delete(HttpSession session, Model model) {
    String id = (String)session.getAttribute("id");
    int result = ms.delete(id);
    if (result > 0) session.invalidate();
    model.addAttribute("result", result);
    return "delete";
}
```

logout

```
@RequestMapping("logout")
public String logout(HttpSession session) {
    session.invalidate();
    return "logout";
}
```

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html> <html> <head> <meta charset="UTF-8">
<title>Insert title here</title> </head> <body>
<script type="text/javascript">
    alert("로그아웃 되었습니다");
    location.href="loginForm.html";
</script>
</body>
</html>
```

Member Interface

```
package com.ch.mybatis.dao;  
import com.ch.mybatis.model.Member;  
public interface MemberDao {  
    Member select(String id);  
    int insert(Member member);  
    int update(Member member);  
    int delete(String id);  
}
```

```
package com.ch.mybatis.service;  
import com.ch.mybatis.model.Member;  
public interface MemberService {  
    Member select(String id);  
    int insert(Member member);  
    int update(Member member);  
    int delete(String id);  
}
```

MemberDaoImpl

```
package com.ch.mybatis.dao;
import org.mybatis.spring.SqlSessionTemplate;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;
import com.ch.mybatis.model.Member;
@Repository
public class MemberDaoImpl implements MemberDao {
    @Autowired
    private SqlSessionTemplate sst;
    public Member select(String id) {
        return sst.selectOne("membersns.select", id);
    }
    public int insert(Member member) {
        return sst.insert("membersns.insert", member);
    }
    public int update(Member member) {
        return sst.update("membersns.update", member);
    }
    public int delete(String id) {
        return sst.update("delete", id);
    }
}
```

MemberServiceImpl

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import com.ch.mybatis.dao.MemberDao;
import com.ch.mybatis.model.Member;
@Service
public class MemberServiceImpl implements MemberService {
    @Autowired
    private MemberDao md;
    public Member select(String id) {
        return md.select(id);
    }
    public int insert(Member member) {
        return md.insert(member);
    }
    public int update(Member member) {
        return md.update(member);
    }
    public int delete(String id) {
        return md.delete(id);
    }
}
```

암호화

<!-- 암호화 -->

Root.xml namespaces에 security체크하고 추가

```
<bean id="passwordEncoder"
```

```
class="org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder">
```

```
</bean>
```

MemberController에

private BCryptPasswordEncoder passwordEncoder; 추가

```
@RequestMapping("join")
```

```
@RequestMapping("update")
```

```
// BCrypt를 활용한 비밀번호 암호화
```

```
String encPassword = passwordEncoder.encode(member.getPassword());
```

```
member.setPassword(encPassword);
```

```
@RequestMapping("login")
```

```
// 암호화된 값 비교
```

```
if (passwordEncoder.matches(member.getPassword(), member2.getPassword()))
```


회원관리 사진 여러 장


```
joinForm.jsp를 복사하여 joinForm2.jsp를 만들고 속성에 multiple="multiple"을 추가
<form action="join2.html" method="post" name="frm" enctype="multipart/form-data"
onsubmit="return chk()">
<table class="table table-bordered">
    <tr> <td>아이디<span class="glyphicon glyphicon-user"> </span> </td>
<td> <input type="text" name="id" required="required" autofocus="autofocus">
<input type="button" onclick="idChk()" class="btn btn-info btn-sm" value="중복체크">
    <span id="disp" class="err"> </span> </td> </tr>
<tr> <th>암호<span class="glyphicon glyphicon-lock"> </span> </th>
    <td> <input type="password" name="password"
        required="required"> </td> </tr>
<tr> <th>암호확인<span class="glyphicon glyphicon-lock"> </span> </th>
    <td> <input type="password" name="password2"
        required="required"> </td> </tr>
<tr> <th>이름 </th> <td> <input type="text" name="name"
        required="required"> </td> </tr>
<tr> <th>이메일<span class="glyphicon glyphicon-envelope"> </span>
    </th> <td> <input type="email" name="email"
        required="required"> </td> </tr>
<tr> <th>사진<span class="glyphicon glyphicon-picture"> </span>
    </th> <td> <input type="file" name="file"
        required="required" multiple="multiple"> </td> </tr>
<tr> <th colspan="2"> <input type="submit"> </th> </tr>
</table> </form>
```

회원관리 사진 여러 장

```
@RequestMapping("join2")
public String join2(Member member, Model model, HttpSession session,
MultipartHttpServletRequest mr) throws IOException {
    int result = 0;    Member mem = ms.select(member.getId());
    if (mem == null) {
        String real = session.getServletContext().getRealPath("/upload");
        List<MultipartFile> list = mr.getFiles("file");
        List<MemberPhoto> photos = new ArrayList<MemberPhoto>();
        for (MultipartFile mf : list) {
            MemberPhoto mp = new MemberPhoto();
            String fileName = mf.getOriginalFilename();
            mp.setId(member.getId());    mp.setFileName(fileName);
            photos.add(mp);
            FileOutputStream fos =
                new FileOutputStream(new File(real+"/"+fileName));
            fos.write(mf.getBytes());    fos.close();
            member.setFileName(fileName);
        }
        result = ms.insert(member);
        ms.insertPhoto(photos);
    } else result = -1;
    model.addAttribute("result", result);
    return "join";
}
```

회원관리 사진 여러 장

회원 상세정보

아이디	k5
이름	byung kang
이메일	kbj010@hanmail.net
등록일	2020-08-01
사진	

메인

회원관리 사진 여러 장

```
CREATE or replace FUNCTION get_seq  
    RETURN NUMBER
```

```
IS  
BEGIN  
    RETURN memberphoto_seq.nextval;  
END;  
/
```

```
<resultMap id="memberphotoResult" type="memberphoto">  
    <result property="num" column="num" />  
    <result property="id" column="id" />  
    <result property="fileName" column="fileName" />  
</resultMap>  
<insert id="insertFileList" parameterType="java.util.List">  
    INSERT ALL  
    <foreach collection="list" item="item">  
        INTO memberphoto          (num, id, fileName)  
        VALUES (get_seq,    #{item.id}, #{item.fileName} )  
    </foreach>  
    select * from dual  
</insert>  
<select id="selectList" parameterType="string" resultMap="memberphotoResult">  
    select * from memberphoto where id=#{id}  
</select>
```

회원관리 사진 여러 장

```
@RequestMapping("view2")
public String view2(Model model, HttpSession session) {
    String id = (String)session.getAttribute("id");
    Member member = ms.select(id);
    List<MemberPhoto> list = ms.listPhoto(id);
    model.addAttribute("member", member);
    model.addAttribute("list", list);
    return "view2";
}
```

```
<table class="table table-striped">
    <tr> <td>아이디 </td> <td>${member.id}</td> </tr>
    <tr> <td>이름 </td> <td>${member.name}</td> </tr>
    <tr> <td>이메일 </td> <td>${member.email}</td> </tr>
    <tr> <td>등록일 </td> <td>${member.regdate}</td> </tr>
    <tr> <td>사진 </td> <td>
        <c:forEach var="photo" items="${list }">
            
        </c:forEach>
    </td>
</table>
```

Configuration.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>
  <typeAliases>
    <typeAlias alias="dept" type="com.ch.ch09.model.Dept" />
    <typeAlias alias="emp" type="com.ch.ch09.model.Emp" />
  </typeAliases>
  <mappers>
    <mapper resource="dept.xml" />
    <mapper resource="emp.xml"/>
  </mappers>
</configuration>
```


직원관리

dept.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="deptns">
  <resultMap type="dept" id="deptResult">
    <result property="deptno" column="deptno"/>
    <result property="dname" column="dname"/>
    <result property="loc" column="loc"/>
  </resultMap>
  <select id="selectList" resultType="dept">
    select * from dept order by deptno</select>
  <select id="select" parameterType="int" resultType="dept">
    select * from dept where deptno=#{deptno}
  </select>
  <insert id="insert" parameterType="dept">
    insert into dept values (#{deptno},#{dname},#{loc})</insert>
  <update id="update" parameterType="dept">
    update dept set dname=#{dname}, loc=#{loc}
    where deptno=#{deptno}
  </update>
  <delete id="delete" parameterType="int">
    delete from dept where deptno=#{deptno}</delete>
</mapper>
```

직원관리

Emp.xml

```
<mapper namespace="empns">
<resultMap type="emp" id="empResult">
    <result property="empno"    column="empno"/>
    <result property="ename"    column="ename"/>
    <result property="job"      column="job"/>
    <result property="mgr"      column="mgr"/>
    <result property="hiredate" column="hiredate"/>
    <result property="sal"      column="sal"/>
    <result property="comm"     column="comm"/>
    <result property="deptno"   column="deptno"/>
    <!-- 조인용 -->
    <result property="mgrName"   column="mgrName"/>
    <collection property="dept" resultMap="deptResult"/>
</resultMap>
<resultMap type="dept" id="deptResult">
    <result property="deptno"   column="deptno"/>
    <result property="dname"    column="dname"/>
    <result property="loc"      column="loc"/>
</resultMap>
<select id="allEmpList" resultMap="empResult">
    select e.*,d.* from emp e,dept d
        where e.deptno=d.deptno order by e.deptno, sal desc
</select>
```


직원관리

```
<select id="list" parameterType="int" resultMap="empResult">
    select * from emp where deptno=#{deptno} order by ename
</select>
<select id="select" parameterType="int" resultType="emp">
    select * from emp where empno=#{empno}
</select>
<select id="empList" resultMap="empResult">
    select * from emp order by ename
</select>
<insert id="insert" parameterType="emp">
    insert into emp values (#{empno},#{ename},#{job},#{mgr},
        #{hiredate},#{sal},#{comm},#{deptno})
</insert>
<delete id="delete" parameterType="int">
    delete from emp where empno=#{empno}
</delete>
<update id="update" parameterType="emp">
    update emp set ename=#{ename},job=#{job},mgr=#{mgr},
        hiredate=#{hiredate},sal=#{sal},comm=#{comm},
        deptno=#{deptno} where empno=#{empno}
</update>
</mapper>
```

root-context.xml

```
<context:property-placeholder
    location="classpath:db.properties" />
<!-- myBatis시작 -->
<bean id="sqlSessionFactory"
    class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="dataSource" ref="dataSource" />
    <property name="configLocation"
        value="classpath:configuration.xml" />
</bean>
<bean id="session" class="org.mybatis.spring.SqlSessionTemplate">
    <constructor-arg index="0" ref="sqlSessionFactory" />
</bean>
<!-- DB연결 풀 -->
<bean id="hikariConfig" class="com.zaxxer.hikari.HikariConfig">
    <property name="driverClassName" value="${driver}" />
    <property name="jdbcUrl" value="${url}" />
    <property name="username" value="${userId}" />
    <property name="password" value="${password}" />
</bean>
<bean id="dataSource" class="com.zaxxer.hikari.HikariDataSource">
    destroy-method="close">
    <constructor-arg ref="hikariConfig"></constructor-arg>
</bean>
```

직원관리

```
package mybatis1.model;

    public class Dept {
        private int deptno;
        private String dname;
        private String loc;

        public int getDeptno() {return deptno;}
        public void setDeptno(int deptno) {
            this.deptno = deptno;
        }
        public String getDname() {return dname;}
        public void setDname(String dname) {
            this.dname = dname;
        }
        public String getLoc() {return loc;}
        public void setLoc(String loc) {
            this.loc = loc;
        }
    }
```

직원관리

```
package com.ch.ch08.model;
import java.sql.Date;
import lombok.Getter;
import lombok.Setter;
@Setter
@Getter
public class Emp {
    private int empno;
    private String ename;
    private String job;
    private int mgr;
    private Date hiredate;
    private int sal;
    private int comm;
    private int deptno;
    // 관리자 이름
    private String mgrName;
    // join용
    private Dept dept;
}
```

직원관리

```
public int getEmpno() {return empno;}
public void setEmpno(int empno) {
    this.empno = empno;
}
public String getEname() {return ename;}
public void setEname(String ename) {
    this.ename = ename;
}
public String getJob() {return job;}
public void setJob(String job) {
    this.job = job;
}
public int getMgr() {return mgr;}
public void setMgr(int mgr) {
    this.mgr = mgr;
}
public Date getHireddate() {return hireddate;}
public void setHireddate(Date hireddate) {
    this.hireddate = hireddate;
}
```

직원관리

```
public int getSal() {  
    return sal;  
}  
public void setSal(int sal) {  
    this.sal = sal;  
}  
public int getComm() {  
    return comm;  
}  
public void setComm(int comm) {  
    this.comm = comm;  
}  
public int getDeptno() {  
    return deptno;  
}  
public void setDeptno(int deptno) {  
    this.deptno = deptno;  
}  
}
```

DeptDAO.java

```
package mybatis1.dao;
import java.util.List;
import mybatis1.model.Dept;
public interface DeptDao {
    List<Dept> list();
    Dept selectOne(int deptno);
    int update(Dept dept);
    int delete(int deptno);
    int insert(Dept dept);
}
```

db.properties

```
driver=oracle.jdbc.driver.OracleDriver
url=jdbc:oracle:thin:@127.0.0.1:1521:xe
userId=scott
password=tiger
maxPoolSize=20
```

DeptDaoImpl.java

@Repository

```
public class DeptDaoImpl implements DeptDao {  
    @Autowired  
    private SqlSessionTemplate sst;  
    public List<Dept> deptList() {  
        return sst.selectList("deptns.deptList");  
    }  
    public Dept select(int deptno) {  
        return sst.selectOne("deptns.select", deptno);  
    }  
    public int insert(Dept dept) {  
        return sst.insert("deptns.insert", dept);  
    }  
    public int update(Dept dept) {  
        return sst.update("deptns.update", dept);  
    }  
    public int delete(int deptno) {  
        return sst.delete("deptns.delete", deptno);  
    }  
}
```


DeptController

@Controller

public class DeptController {

 @Autowired

private DeptService ds;

 @RequestMapping("deptList")

public String deptList(Model model) {

 List<Dept> deptList = ds.deptList();

 model.addAttribute("deptList", deptList);

return "deptList";

 }

 @RequestMapping("insertForm")

public String insertForm() { return "insertForm"; }

 @RequestMapping(value="deptChk", produces="text/html;charset=utf-8")

 @ResponseBody

public String deptChk(int deptno) {

 String msg="";

 Dept dept = ds.select(deptno);

if (dept == null) msg="사용가능합니다";

else msg="이미 사용중입니다";

return msg;

 }

DeptController

```
@RequestMapping("insert")
public String insert(Dept dept, Model model) {
    int result = ds.insert(dept);
    model.addAttribute("result", result);
    return "insert";
}

@RequestMapping("updateForm")
public String updateForm(int deptno, Model model) {
    Dept dept = ds.select(deptno);
    model.addAttribute("dept", dept);
    return "updateForm";
}

@RequestMapping("update")
public String update(Dept dept, Model model) {
    int result = ds.update(dept);
    model.addAttribute("result", result);
    return "update";
}
```

DeptController

```
@RequestMapping("delete")  
public String delete(int deptno, Model model) {  
    int result = ds.delete(deptno);  
    model.addAttribute("result", result);  
    return "delete";  
}  
  
}
```

DeptService

```
package mybatis1.service;
import java.util.List;
import mybatis1.model.Dept;
public interface DeptService {
    List<Dept> list();
    Dept selectOne(int deptno);
    int update(Dept dept);
    int delete(int deptno);
    int insert(Dept dept);
}
```

DeptServiceImpl

@Service

```
public class DeptServiceImpl implements DeptService {  
    @Autowired  
    private DeptDao dd;  
    public List<Dept> list() {  
        return dd.list();  
    }  
    public Dept selectOne(int deptno) {  
        return dd.selectOne(deptno);  
    }  
    public int update(Dept dept) {  
        return dd.update(dept);  
    }  
    public int delete(int deptno) {  
        return dd.delete(deptno);  
    }  
    public int insert(Dept dept) {  
        return dd.insert(dept);  
    }  
}
```

dept.xml

```
<mapper namespace="deptns">
    <resultMap id="deptResult" type="dept">
        <result property="deptno" column="deptno" />
        <result property="dname" column="dname" />
        <result property="loc" column="loc" />
    </resultMap>
    <select id="deptList" resultMap="deptResult">
        select * from dept order by deptno
    </select>
    <select id="select" parameterType="int" resultType="dept">
        select * from dept where deptno=#{deptno}
    </select>
    <insert id="insert" parameterType="dept">
        insert into dept values("#{deptno}", #{dname}, #{loc})
    </insert>
    <update id="update" parameterType="dept">
        update dept set dname=#{dname}, loc=#{loc} where deptno=#{deptno}
    </update>
    <delete id="delete" parameterType="int">
        delete from dept where deptno=#{deptno}
    </delete>
```

list.jsp

```
<%@ include file= "header.jsp" %>
<body>
  <c:if test= "${not empty msg}">
    <span class= "err">${msg}</span>
  </c:if>
  <table> <caption>부서 목록</caption>
    <tr> <th>부서코드</th> <th>부서명</th> <th>근무지</th>
    <th>수정여부</th> <th>삭제여부</th> </tr>
    <c:forEach var= "dept" items= "${list}">
      <tr> <td>${dept.deptno}</td> <td>${dept.dname }</td>
      <td>${dept.loc }</td>
      <td><a href= "updateForm.do?deptno=${dept.deptno}">수정
      </a> </td>
      <td><a href= "delete.do?deptno=${dept.deptno}">삭제
      </a> </td> </tr>
    </c:forEach>
  </table>
  <a href= "insertForm.do">입력</a>
</body>
</html>
```

insertForm.jsp

```
<%@ include file= "header.jsp" %>
<body>
  <c:if test= "${not empty msg }">
    <span class= "err">${msg}</span>
  </c:if>
  <form action= "insert.do" method= "post">
    <table> <caption> 부서정보 입력 </caption>
      <tr> <th> 부서코드 </th> <td> <input type= "text"
        name= "deptno" value= "${dept.deptno }"> </td> </tr>
      <tr> <th> 부서명 </th> <td> <input type= "text" name= "dname"
        value= "${dept.dname }" required= "required"> </td> </tr>
      <tr> <th> 근무지 </th> <td> <input type= "text" name= "loc"
        value= "${dept.loc }" required= "required"> </td> </tr>
      <tr> <th colspan= "2"> <input type= "submit" value= "확인"> </th>
    </table>
  </form>
</body>
</html>
```


updateForm.jsp

```
<%@ include file= "header.jsp" %>
<body>
  <c:if test= "${not empty msg }">
    <span class= "err"> ${msg}</span>
  </c:if>
  <form action= "update.do" method= "post">
    <table> <caption> 부서정보 수정 </caption>
      <tr> <th> 부서코드 </th> <td> ${dept.deptno } <input type= "hidden"
        name= "deptno" value= "${dept.deptno }"> </td> </tr>
      <tr> <th> 부서명 </th> <td> <input type= "text" name= "dname"
        value= "${dept.dname }" required= "required"> </td> </tr>
      <tr> <th> 근무지 </th> <td> <input type= "text" name= "loc"
        value= "${dept.loc }" required= "required"> </td> </tr>
      <tr> <th colspan= "2"> <input type= "submit" value= "확인"> </th>
    </table>
  </form>
</body>
</html>
```

부서 목록

부서코드	부서명	근무지	수정	삭제
10	ACCOUNTING	NEW YORK	수정	삭제
11	놀부팀	인천	수정	삭제
20	RESEARCH	DALLAS	수정	삭제
30	SALES	CHICAGO	수정	삭제
40	OPERATIONS	BOSTON	수정	삭제

부서정보입력

전직원 목록

ACCOUNTING직원 목록

사번	이름	업무	입사일
7782	CLARK	MANAGER	1981-06-09
7839	KING	PRESIDENT	1981-11-17
7934	MILLER	CLERK	1982-01-23

직원정보 입력

부서 목록

부서 목록

부서코드	부서명	근무지	수정	삭제
10	ACCOUNTING	NEW YORK	수정	삭제
11	놀부팀	인천	수정	삭제
20	RESEARCH	DALLAS	수정	삭제
30	SALES	CHICAGO	수정	삭제
40	OPERATIONS	BOSTON	수정	삭제

직원 상세정보

사번	7782
이름	CLARK
업무	MANAGER
관리자	7839(KING)
입사일	1981-06-09
급여	2450
커미션	0
부서코드	10

[수정](#)[삭제](#)[직원목록](#)[부서목록](#)

ACCOUNTING직원 목록

사번	이름	업무	입사일
7782	CLARK	MANAGER	1981-06-09
7839	KING	PRESIDENT	1981-11-17

EmpDaoImpl

```
package com.ch.ch09.dao;
import java.util.HashMap;
@Repository
public class EmpDaoImpl implements EmpDao {
    @Autowired
    private SqlSessionTemplate sst;
    public List<Emp> empList(int deptno) {
        Map<String, Integer> map = new HashMap<String, Integer>();
        map.put("deptno", deptno);
        return sst.selectList("empns.empList", map);
    }
    public Emp select(int empno) {
        return sst.selectOne("empns.select", empno);
    }
    public int insert(Emp emp) {
        return sst.insert("empns.insert", emp);
    }
}
```

EmpDaoImpl

```
public int delete(int empno) {  
    return sst.delete("empns.delete", empno);  
}  
public int update(Emp emp) {  
    return sst.update("empns.update", emp);  
}  
public List<Emp> jobList() {  
    return sst.selectList("empns.jobList");  
}  
public List<Emp> allList() {  
    return sst.selectList("empns.allList");  
}  
}
```

Emp

```
package com.ch.ch09.model;
import java.sql.Date;
public class Emp {
    private int empno;
    private String ename;
    private String job;
    private int mgr;
    private Date hiredate;
    private int sal;
    private int comm;
    private int deptno;

    // 부서테이블내용추가
    private String dname;
    private String loc;
```


EmpServiceImpl

@Service

```
public class EmpServiceImpl implements EmpService {  
    @Autowired  
    private EmpDao ed;  
    public List<Emp> empList(int deptno) {  
        return ed.empList(deptno);  
    }  
    public Emp select(int empno) { return ed.select(empno); }  
    public int insert(Emp emp) { return ed.insert(emp); }  
    public int delete(int empno) { return ed.delete(empno); }  
    public int update(Emp emp) {  
        return ed.update(emp);  
    }  
    public List<Emp> jobList() {  
        return ed.jobList();  
    }  
    public List<Emp> allList() {  
        return ed.allList();  
    }  
}
```

EmpController

@Controller

```
public class EmpController {  
    @Autowired  
    private EmpService es;  
    @Autowired  
    private DeptService ds;  
    @RequestMapping("empList")  
    public String empList(int deptno, Model model) {  
        Dept dept = ds.select(deptno);  
        List<Emp> empList = es.empList(deptno);  
        model.addAttribute("dept", dept);  
        model.addAttribute("empList", empList);  
        return "emp/empList";  
    }  
    @RequestMapping("empInsertForm")  
    public String empInsert(Model model) {  
        List<Dept> deptList = ds.deptList();  
        List<Emp> empAllList = es.empList(0);  
        List<Emp> jobList = es.jobList();  
        model.addAttribute("jobList", jobList);  
        model.addAttribute("deptList", deptList);  
        model.addAttribute("empList", empAllList);  
        return "emp/empInsertForm";  
    }  
}
```

EmpController

```
@RequestMapping(value="empnoChk",
    produces="text/html;charset=utf-8")
@ResponseBody
public String empnoChk(int empno) {
    String msg = "";
    Emp emp = es.select(empno);
    if (emp==null) msg="사용가능한 사번입니다";
    else msg="다른 사번을 사용하세요";
    return msg;
}

@RequestMapping("empInsert")
public String insert(Emp emp, Model model) {
    int result = es.insert(emp);
    model.addAttribute("result", result);
    model.addAttribute("emp", emp);
    return "emp/empInsert";
}
```

EmpController

```
@RequestMapping("empView")
public String empView(int empno, Model model) {
    Emp emp = es.select(empno);
    Emp emp2 = es.select(emp.getMgr());
    model.addAttribute("emp", emp);
    model.addAttribute("emp2", emp2);
    return "emp/empView";
}

@RequestMapping("empDelete")
public String empDelete(int empno, Model model) {
    Emp emp = es.select(empno);
    int result = es.delete(empno);
    model.addAttribute("emp", emp);
    model.addAttribute("result", result);
    return "emp/empDelete";
}
```

EmpController

```
@RequestMapping("empUpdateForm")
public String empUpdateForm(int empno, Model model) {
    List<Dept> deptList = ds.deptList();
    //List<Emp> empAllList = es.empAllList();
    List<Emp> empAllList = es.empList(0);
    Emp emp = es.select(empno);
    model.addAttribute("deptList", deptList);
    model.addAttribute("empList", empAllList);
    model.addAttribute("emp", emp);
    return "emp/empUpdateForm";
}

@RequestMapping("empUpdate")
public String empUpdate(Emp emp, Model model) {
    int result = es.update(emp);
    model.addAttribute("result", result);
    model.addAttribute("emp", emp);
    return "emp/empUpdate";
}
```

EmpController

```
@RequestMapping("allList")
public String allList(Model model) {
    List<Emp> allList = es.allList();
    model.addAttribute("allList", allList);
    return "allList";
}
}
```

allEmpList.jsp

```
<%@ include file="../header.jsp" %>
<!DOCTYPE html> <html> <head> <meta charset="UTF-8">
<title>Insert title here</title> </head> <body>
<div class="container">
<table class="table table-striped table-bordered">
    <caption class="text-primary">전직원 목록</caption>
    <tr> <th>사번</th> <th>이름</th> <th>업무</th> <th>관리자</th>
        <th>입사일</th> <th>급여</th> <th>보너스</th>
        <th>부서명</th> <th>근무지</th> </tr>
    <c:forEach var="emp" items="${list }">
        <tr> <td>${emp.empno }</td> <td>${emp.ename }</td>
            <td>${emp.job}</td> <td>${emp.mgrName}</td>
            <td>${emp.hiredate}</td> <td>${emp.sal}</td>
            <td>${emp.comm}</td> <td>${emp.dept.dname }</td>
            <td>${emp.dept.loc}</td> </tr>
    </c:forEach>
</table>
<a href="deptList.html" class="btn btn-info">부서목록</a>
</div>
</body>
</html>
```


[MyBatis] foreach 구문

1. List 일 경우

```
<insert id="insertBoard" parameterType="java.util.List">
    INSERT INTO T_BOARD (TITLE,CONTENTS, DISPLAY_YN, EMAIL,    REG_ID,REG_DATE )
VALUES (
    <foreach collection="list" item="item" separator=",">
        #{item.title},#{item.contents},#{item.displayYn},#{item.email},#{item.regId},
        sysdate
    </foreach>
)
</insert>
```

! collection="list" item="item" 는 default 이다.

2. Map 일 경우

```
List<Map<String, Object>> goodsList = new List<Map<String,Object>>();
goodsList.put("goodsList", goodsList);
<insert id="insertEventGoodsMng" parameterType="java.util.HashMap">
    <if test="goodsList.size != 0">
        INSERT INTO EVENT_GOODS_MNG (EVENT_GOODS_MNG_IDX,GOODS_IDX )
        VALUES
        <foreach collection="goodsList" item="item" separator=",">
            (  #{eventIdx} , #{item.goodsIdx} )
        </foreach>    </if>
</insert>
```

MyBatis foreach문 지원 태그

collection : 전달받은 인자. List or Array 형태만 가능

item : 전달받은 인자 값을 alias 명으로 대체

open : 구문이 시작될때 삽입할 문자열

close : 구문이 종료될때 삽입할 문자열

separator : 반복 되는 사이에 출력할 문자열

index : 반복되는 구문 번호이다. 0부터 순차적으로 증

<foreach collection="List or Array" item="alias" ></foreach>

사용 예시

```
<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>
```

-참고 자료 MyBatis 공식 사이트

배열 예시

공통 배열 데이터

```
String[] userArray = {"1","2","3"}
```

1-1. 배열 파라미터를 Map을 통해 넘겼을 경우

```
public List<Members> getAuthUserList(String[] userArray) {  
    HashMap<String, Object> map = new HashMap<String, Object>();  
    map.put("userArray",userArray);  
}
```

```
<select id="getAuthUserList" resultType="members">  
    SELECT m.*,a.name FROM members AS m , authority AS a  
    where m.authority = a.authority    and m.authority IN  
        <foreach collection="userArray" item="arr" open="(" close=")"  
separator=",">  
            #{arr}  
        </foreach>  
    ORDER BY m.authority;  
</select>
```

※ 주의 : collection을 꼭! 넘겨주는 배열 변수 값과 동일하게 작성하셔야 합니다.

배열 파라미터를 직접 넘겼을 경우

```
public List<Members> getAuthUserList(String[] userArray) {  
    return sst.selectList("getAuthUserList", userArray);  
}
```

user-Mapper.xml

```
<select id="getAuthUserList" resultType="members">  
    SELECT m.*,a.name FROM members AS m , authority AS a  
    where m.authority = a.authority and m.authority IN  
        <foreach collection="array" item="arr" open="(" close=")" separator=",">  
            #{arr}  
        </foreach>  
    ORDER BY m.authority;  
</select>
```

※ 주의 : collection을 꼭! "array"로 작성하셔야 합니다.

리스트 예시

공통 리스트 데이터

```
List<Members> chkList = userService.getUserList();  
//SELECT * FROM members 결과값
```

2-1. 리스트 Map을 통해 넘겼을 경우

```
public List<Members> getListTest(List<Members> chkList) {  
    HashMap<String, Object> map = new HashMap<String, Object>();  
    map.put("chkList",chkList);  
    return sst.selectList("getListTest", map);  
}
```

user-Mapper.xml

```
<select id="getListTest" resultType="members">  
    SELECT m.*,a.name FROM members AS m, authority AS a  
    where m.authority = a.authority and m.authority IN  
        <foreach collection="chkList" item="item" open="(" close=")" separator=",">  
            #{item.authority}  
        </foreach>  
    ORDER BY m.authority;  
</select>
```

리스트 안에 뽑고 싶은 결과값을 {key.value} 형태로 뽑으시면 됩니다.

※ 주의 : collection을 꼭! 넘겨주는 리스트 변수 값과 동일하게 작성하셔야 합니다.

리스트 파라미터를 직접 넘겼을 경우

```
public List<Members> getListTest(List<Members> chkList) {  
    return sst.selectList("getListTest", chkList);  
}
```

user-Mapper.xml

```
<select id="getListTest" resultType="members">  
    SELECT m.*,a.name FROM members AS m, authority AS a  
    where m.authority = a.authority and m.authority IN  
        <foreach collection="list" item="item" open="(" close=")" separator=",">  
            #{item.authority}  
        </foreach>  
    ORDER BY m.authority;  
</select>
```

리스트 안에 뽑고 싶은 결과값을 {key.value} 형태로 뽑으시면 됩니다.

※ 주의 : collection을 꼭! "list"로 작성하셔야 합니다.