**iBatis는 ORM(Object Relational Mapping)패턴을 쉽게 사용하기 위한 Library이다.**

기존 JDBC 구현은

Class.forName(JDBC Driver명)

Connection con = DriverManager.getConnection(DBURL, 계정, 비밀번호);

PreparedStatement ps = con.preparedStatement(쿼리);

=> JDBC Driver, DB 연결정보, SQL 쿼리, Java Code가 뒤섞여 있어 복잡하다는 단점이 있다.

예를들어 Table 수정이나 Column 수정이 일어날 때 DAO에서 Board 찾아서 하나하나 수정해야 한다.

﻿■  iBatis

    -  반복되는 코드 제거

    -  자바 클래스와 SQL구문에 대한 매핑 문제

    -  XML 파일을 사용한 자바빈과 SQL 구문의 매핑

    -  DB 생성 PK, 관련 객체의 자동 로딩, 캐싱, 지연 로딩등을 지원

■  저장 (persistence) 프레임워크

    -  SQL에 대한 개발 및 유지보수와 이식성 문제

    -  도메인 객체를 DB에 매핑

    -  OR 매핑을 사용한 자동 SQL 생성

         \*  클래스-테이블, 속성-컬럼, 관계-FK/조인 테이블 매핑 정의

    -  EJB2의 엔티티 빈의 한계

    -  JDO와 Hibernate

■  iBatis 설치 및 구성

    -  <http://code.google.com/p/mybatis/wiki/Downloads?tm=2>

    -  JDK 1.4 이상

    -  application 클래스 path 상에 설치

■  iBatis configuration 개념

    -  SqlMapConfig : 전역설정 및 트렌젝션 관리 정보

    -  SqlMap : 입출력 값과 DB간의 매핑정보

■  SQL Map 설정파일 (SqlMapConfig.xml)

    -  중앙 허브의 역할

    -  DB connection 정보 제공



■  typeAlias 요소

    -  alias로 type명을 대체

    -  iBatis는 정의된 alias로 언제든지 type에 접근가능

       <typeAlias alias="Category" type="kr.nextree.nexbay.domain.object.Category"/>

    -  미리 정의된 typeAlias

       1) Transaction manager : JDBC, JTA, EXTERNAL

       2) Data types : string, long, int, date 등

       3) Data source factory : SIMPLE, DBCP, JNDI

       4) Cache controller : FIFO, LRU, MEMORY, OSCACHE

       5) XML  result : DOM, domCollection, Xml, XmlCollection

iBatis는 ORM 패턴대로 DB연결정보, SQL쿼리를 각각 xml파일로 분리 시킨다고 하였다

SqlMapConfig.xml은 DB 연결정보만을 따로 넣어 놓았고

SQL 쿼리를 넣는 곳은 ibatis1.xml로 설정하여 코딩하였다.

**<sql>ㆍㆍㆍ</sql> 태그는 독립적으로 실행되지 않고 자주 사용되는 쿼리의 일부분을 정의해놓은 것.**

**<sql id="where-shipped-after-value"> ﻿=> where 조건을 미리 정의 해놓은 쿼리문**

**<![CDATA[where shipDate > #{value}]]>      => CDATA 안에 <,&,> 포함쿼리를 구현 내가 지정한 날짜보다 큰 Date를 조건으로**

**</sql>**

**<select id="실행시 사용 id">ㆍㆍㆍ</select> 태그는 실행할 select 쿼리를 지정**

**<include refid="select-count" /> select 태그 안에 <include>태그로 실제 사용할 sql쿼리를 id를 넣어서 지정**

|  |
| --- |
|  |

**■  Nonquery SQL문의 SqlMap API**

   -  insert, update, delete 메소드 제공

      1) insert 메소드

         Object insert(String id, Object parameterObject) throws SQLException;

         => 입력 후 key를 리턴하는 용도로 드물게 사용된다. 특별한 설정이 필요

﻿      2) update 메소드

          int update(String id, Object parameterObject) throws SQLException;

          => 영향을 받은 row 수를 리턴 ﻿(JDBC driver가 지원하는 경우)

      3) delete 메소드

          int delete(String id, Object parameterObject) throws SQLException;

          => 삭제된 row 수를 리턴

      ※ Object prarmeterObject는 테이블에 삽입 할 내용을 속성값으로 갖는 객체(클래스 필요)﻿

      ※ 리턴형이 Object인데 대부분의 경우 null 리턴.

      ※ 시퀀스를 이용해서 Primary key 설정했을 때는 Primary key를 리턴

**#iBatis란?**

iBatis(아이바티스)는SQL에 기반한 데이터베이스와 자바,닷넷,루비등을 연결시켜주는 역할을 하는 영속성 프레임워크이다. 이러한 연결은 프로그램의 소스코드에서 SQL문장을 분리하여 별도의 XML 파일로 저장하고 이 둘을 서로 연결시켜주는 방식으로 작동한다.

**#데이터매퍼**

iBatis 데이터매퍼 API는 프로그래머에게 자바 빈즈 객체를 PreparedStatement 파라미터와 ResultSets으로 쉽게 매핑할 수 있도록 한다. 이는 자바코드의 20%를 사용하여 JDBC기능의 80%제공하는 간단한 프레임워크라는 뜻이다.

데이터 매퍼는 자바빈즈, Map구현체, 원시래퍼타입(String, Integer..) 그리고 SQL문을 위한 XML문서를 매핑하는 XML서술자를 사용하는 매우 간단한 프레임워크를 제공한다. 데이터 매퍼가 관리하는 생명주기는 다음과 같다.

1. 파라미터(자바빈즈,Map 또는 원시래퍼)로써 객체를 제공한다. 파라미터 객체는 update문에서 입력값을 세팅하기 위해 사용되거나 쿼리문의 where절을 셋팅하기 위해서 사용된다.
2. 매핑된 구문을 실행한다. 이 단계는 PreparedStatement 인스턴스를 생성할 것이고 제공된 파라미터 객체를 사용해서 파라미터를 셋팅한다. 그 후 구문을 실행하고 ResultSet으로부터 결과 객체를 생성한다.
3. update의 경우에 영향을 미친 rows의 숫자를 반환한다. 조회작업인 경우에 한 개의 객체 또는 컬렉션 객체를 반환한다. 파라미터처럼 결과 객체는 자바빈즈, Map 원시타입 래퍼또는 XML이 될 수 있다.

**#SQL Map XML 설정**

SQL Maps 는 데이터소스, 데이터 매퍼에 대한 설정, 쓰레드 관리와 같은 SQL Maps와 다른 옵션에 대한 설정을 제공하는 중앙집중적인 XML 설정 파일을 사용해서 설정된다.

**#sqlMapConfig.xml**

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<!DOCTYPE sqlMapConfig PUBLIC "-//iBATIS.com//DTD SQL Map Config 2.0//EN" "http://www.ibatis.com/dtd/sql-map-config-2.dtd">

<sqlMapConfig>

       <!-- 외부 프로퍼티 설정 -->

       <properties resource=*" examples/sqlmap/maps/SqlMapConfig.properties "* />

       <!-- 트랜잭션관리 및 sqlMapClient 상세설정 -->

       <settings    cacheModelsEnabled=*"true"*

                     enhancementEnabled=*"true"*

                     lazyLoadingEnabled=*"true"*

                     maxRequests=*"128"*

                    maxSessions=*"10"*

                    maxTransactions=*"5"*

                    useStatementNamespaces=*"false"*

                    defaultStatementTimeout=*"5"*

                    statementCachingEnabled=*"true"*

                    classInfoCacheEnabled=*"true"* />

     <!-- 결과객체 생성을 위한 factory 클래스 선언 -->

       <resultObjectFactory type=*"com.mydomain.MyResultObjectFactory"*>

             <property name=*"someProperty"* value=*"someValue"* />

       </resultObjectFactory>

      <!-- 긴 경로를 포함한 클래스명을 짧은이름으로 사용하기 위한 별칭설정 -->

       <typeAlias alias=*"article"* type=*"board.model.BoardArticle"* />

      <!-- SimpleDataSource를 이용한 SQL Map응 사용하기위한 데이터소스 설정. -->

       <transactionManager type=*"JDBC"*>

             <dataSource type=*"SIMPLE"*>

                    <property name=*"JDBC.Driver"* value=*"${driver}"* />

                    <property name=*"JDBC.ConnectionURL"* value=*"${url}"* />

                    <property name=*"JDBC.Username"* value=*"${username}"* />

                    <property name=*"JDBC.Password"* value=*"${password}"* />

                    <property name=*"JDBC.DefaultAutoCommit"* value=*"true"* />

                    <property name=*"Pool.MaximumActiveConnections"* value=*"10"* />

                    <property name=*"Pool.MaximumIdleConnections"* value=*"5"* />

                    <property name=*"Pool.MaximumCheckoutTime"* value=*"120000"* />

                    <property name=*"Pool.TimeToWait"* value=*"500"* />

                    <property name=*"Pool.PingQuery"* value=*"select 1 from ACCOUNT"* />

                    <property name=*"Pool.PingEnabled"* value=*"false"* />

                    <property name=*"Pool.PingConnectionsOlderThan"* value=*"1"* />

                    <property name=*"Pool.PingConnectionsNotUsedFor"* value=*"1"* />

             </dataSource>

       </transactionManager>

       <!-- SQL Map에 의해 로드되는 모든 SQL Map파일을 인식한다. -->

       <sqlMap resource=*"/board/dao/boardMapper.xml"* />

</sqlMapConfig>

**#<properties> 요소**

<properties resource=*" examples/sqlmap/maps/SqlMapConfig.properties"* />

  SQL Maps는 SQL Maps XML 설정파일과 함께 속하는 표준적인 자바 속성파일을 지정하는 하나의 <properties>요소를 가질 수 있다. 그렇게 함으로써 속성 파일내에 각각의 이름지어진 값들은 설정파일 내에서 하나의 변수로 사용될 수 있다. 예를들어 프로퍼티파일에 다음과 같은 값이 있다면.

driver = org.com.microsoft.sqlserver.jdbc.SQLServerDriver

설정파일에서는 이렇게 사용할 수 있다.

<property name=*"JDBC.Driver"* value=*"${driver}"* />

**#<settings>요소**

 <!-- 트랜잭션관리 및 sqlMapClient 상세설정 -->

       <settings    cacheModelsEnabled=*"true"*

                    enhancementEnabled=*"true"*

                    lazyLoadingEnabled=*"true"*

                    maxRequests=*"128"*

                    maxSessions=*"10"*

                    maxTransactions=*"5"*

                    useStatementNamespaces=*"false"*

                    defaultStatementTimeout=*"5"*

                    statementCachingEnabled=*"true"*

                    classInfoCacheEnabled=*"true"* />

<srttings>요소는 XML 파일을 빌드하는 SqlMapClient 인스턴스를 위해 다양한 옵션과 최적화를 설정 할 수 있도록 한다. setting 요소와 그것의 모든 속성 값은 모두 선택사항이며 각각의 속성에 대한 설명을 표로 작성하였다.

|  |  |
| --- | --- |
| **maxRequests** | 한꺼번에 SQL문을 수행 할 수 있는 쓰레드의 수이다. 셋팅값보다 많은 쓰레드는 다른 쓰레드가 수행을 완료할때까지 블록상태가 되며, 각각의 DBMS는 다른 제한을 가지지만, 이러한 제한이 없는 DBMS는 없다. 보통 최소 maxTransactions 속성의 10배이며, 항상 maxSessions와 maxTransactions 속성보다는 크다. 동시 요청값의 최대치를 줄이면 성능향상을 기대할 수 있다.ex) maxRequests=*"128"* |
| **maxSessions** | 주어진 시간동안 활성 될 수 있는 세션의 수이다. 세션은 명식적 또는 프로그램적으로 요청 될 수 있으며 , 쓰레드가 SqlMapClient 인스턴스를 사용할 때마다 자동적으로 생성 될수 있다.maxTransactions보다 같거나 커야하고 maxRequests보다 작아야 한다. 동시 세션값의 최대치를 줄이면 메모리 사용량을 줄일 수 있다.ex) maxSessions=*"10"* |
| **maxTranSactions** | 한번에 SqlMapClient.startTransaction()에 들어갈 수 있는 쓰레드의 최대 갯수이다. 셋팅값도다 많은 쓰레드는 앞서 maxRequests와 마찬가지로 블록되며 가각의 DBMS는 다른 제한을가진다. maxSessions보다 작거나 같아야하고 maxRequests보다 작아야한다. 동시 트랜잭션의 최대치를 줄이면 성능향상을 기대 할 수 있다.  ex) maxTransactions=*"5"* |
| **casheModelsEnabled** | SqlMapClient를 위한 모든 캐쉬모델을 가능하게 하거나 가능하지 않게 한다. 디버깅시 도움이 된다.  ex) cacheModelsEnabled=*"true"* |
| **lazyLoadingEnabled** | SqlMapClinet를 위한 모든 LazyLoading을 가능하게 하거나 가능하지 않게 한다. 이역시 디버깅시에 도움이 된다.  ex)  lazyLoadingEnabled=*"true"* |
| **enhancementEnabled** | 이 셋팅은 향상된 LazyLoading처럼 최적화된 바자빈즈 속성 접근을 위해 런타임시 바이트코드 향상을 가능하게 한다.  ex) enhancementEnabled=*"true"* |
| **useStatementNamespaces** | sqlMap 이름과 statement이름으로 구성된 전체이름으로 매핑구문을 참조할때 사용한다.  ex) useStatementNamespaces=*"false"* |
| **defaultStatementTimeout** | 모든구문에 대한 JDBC쿼리 타임아웃처럼 적용될 정수 값이다.이 값은 매핑구문의 "statement" 속성으로 무시될 수있으며, 매핑구문의 "statement" 속성이 설정되지 않는 다면 쿼리 타임아웃이 세팅되지 않게된다.모든 드라이버가 이 셋팅을 지원하지는 않으니 반드시 확인해야한다.  ex) defaultStatementTimeout=*"5"* |
| **classInfoCacheEnabled** | ibaits가 클래스의 캐시를 관리하할지 안할지 설정하게 된다. 많은 클래스가 재사용된다면 재빨리 시작하도록 한다.  ex) classInfoCacheEnabled=*"true"* |
| **statementCachingEnabled** | iBatis가 prepared 구문의 지역적인 캐시를 관리하며 설정할시 명확한 성능향상을 보여준다.  ex) statementCachingEnabled=*"true"* |

**#<resultObjectFactory>요소**

<!-- 결과객체 생성을 위한 factory 클래스 선언 -->

       <resultObjectFactory type=*"com.mydomain.MyResultObjectFactory"*>

             <property name=*"someProperty"* value=*"someValue"* />

       </resultObjectFactory>

 SQL구문의 실행으로 결결과 객체를 생성하기 위한 factory 클래스를 명시한다. resultObjectFactory요소는 사용하지 않을 수도 있으며, 이 요소를 명시하지 않는다면 iBatis는 결과 객체를 생성하기 위해 내부 기법을 사용한다.

iBatis는 다음의 경우 결과객체를 생성하게 된다.

1. ResultSet에서 반환되는 레코드를 매핑할때 (가장 공통적인 경우)
2. resultMap으로 결과 요소의 내포된 select구문을 사용할 때, 내포된 select 구문이 parameterClass를 선언한다면, iBatis는 애포된 select를 실행하기 전에 클래스의 인스턴스를 생성하고 값을 채운다.
3. 저장프로시저를 실행할때 iBatis는 output파라미터를 위한 객체를 생성한다.

**#<typeAilas>요소**

<!-- 긴 경로를 포함한 클래스명을 짧은이름으로 사용하기 위한 별칭설정 -->

       <typeAlias alias=*"article"* type=*"board.model.BoardArticle"* />

긴 전체 경로를 포함한 클래스명을 참조하기 좋은 짧은 이름으로 명시하도록 한다. 즉 위의 설정시 board.model.BoardArticle 클래스를 사용할 위치에 article을 사용할 수 있게 한다.

SQL Maps은 설정 파일에서 사용되는 몇몇의 alias를 미리 정의해 두었다.

|  |  |
| --- | --- |
| **Transaction Manager Aliases** | |
| JDBC   JTA   EXTERNAL | com.ibatis.sqlmap.engine.transaction.jdbc.JdbcTransactionConfig   com.ibatis.sqlmap.engine.transaction.jta.JataTransactionConfig   com.ibatis.sqlmap.engine.transaction.external.ExternalTransactionConfig |
| **Data Source Factory Aliases** | |
| SIMPLE   DBCP   JNDI | com.ibatis.sqlmap.engine.datasource.SimpleDataSourceFactory   com.ibatis.sqlmap.engine.datasource.DbcpDataSourceFactory   com.ibatis.sqlmap.engine.datasource.JndiDataSourceFactory |

**#<transactionManeger>요소**

<transactionManager type=*"JDBC"*>

      ......

</transactionManager>

SQL Maps를 위한 트랜잭션 관리를 설정 할 수 있다.*type* 속성은 사용하귀 위한 트랜잭션 관리자를 표시하며, 그 값은 클래시명이거나 alias일수 있다.  트랜잭션 관리자는 JDBC, JTA, EXTERNAL이 존재한다.

|  |  |
| --- | --- |
| JDBC | 커넥션 commit()과 rollback()메소드를 통해 트랜잭션을 제어하기 위한 JDBC를 사용한다. |
| JTA | SQL Maps가 다른 데이터베이스나 트랜잭션 자원을 포함하는 넓은 범위의 트랜잭션을 포함하도록 하는 JTA전역트랜잭션을 사용한다. 이 설정은 JNDI자원으로부터 사용자 트랜잭션을 위치시키기 위한 UserTransaction 속성값을 요구한다. |
| EXTERNAL | 사용자정의 트랜잭션을 관리하도록 한다. 데이터소스를 설정 할 수 있지만, 프레임워크 생명주기의 부분처럼 트랜잭션이 커밋되거나 롤백되지 않는다. 이 세팅은 비트랜잭션(예로 읽기전용)데이터 베이스에 유용하다. |

<transactionManager>요소는 commitRequierd요소를 선택할 수 있다. 대개 iBatis는 insert,update 또는 delete 작업이 실행 되지 않고서는 트랜잭션을 커밋하지 않는다. 하지만 명시적으로 commitTransaction()메소드를 호출하면 커밋을 할 수 있게된다.

**#<dataSource>요소**

 <dataSource type=*"SIMPLE"*>

                    <property name=*"JDBC.Driver"* value=*"${driver}"* />

                    <property name=*"JDBC.ConnectionURL"* value=*"${url}"* />

                    <property name=*"JDBC.Username"* value=*"${username}"* />

                    <property name=*"JDBC.Password"* value=*"${password}"* />

                    <property name=*"JDBC.DefaultAutoCommit"* value=*"true"* />

                    <property name=*"Pool.MaximumActiveConnections"* value=*"10"* />

                    <property name=*"Pool.MaximumIdleConnections"* value=*"5"* />

                    <property name=*"Pool.MaximumCheckoutTime"* value=*"120000"* />

                    <property name=*"Pool.TimeToWait"* value=*"500"* />

                    <property name=*"Pool.PingQuery"* value=*"select 1 from ACCOUNT"* />

                    <property name=*"Pool.PingEnabled"* value=*"false"* />

                    <property name=*"Pool.PingConnectionsOlderThan"* value=*"1"* />

                    <property name=*"Pool.PingConnectionsNotUsedFor"* value=*"1"* />

 </dataSource>

<transactionManager>요소에 포함되는 요소이며 SQL Maps를 사용하기 위한 데이터소스를 설정하기 위한 속성의 집합니다. 기본적으로 제공되는 3가지 데이터소스가 존재하지만 원한다면 사용자가 직점 정의하여 사용할 수도 있다.

**#<sqlMap>요소**

<!-- SQL Map에 의해 로드되는 모든 SQL Map파일을 인식한다. -->

       <sqlMap resource=*"/board/dao/boardMapper.xml"* />

sqlMap 요소는 명시적으로 SQL Map이나 다른 SQL Map설정파일을 포함할 때 사용한다. SqlMapClient 인스턴스에의해 사용되는 각각의 SQL Map XML파일은 반드시 선언해야 한다.

**#매핑구문 - 1**

SQL Map(매퍼파일) 은 매핑구문에 집중하게 된다. 매핑 구문에는 어떠한 SQL문을 사용 할 수 있고, 파라미터maps(input)과 결과maps(output)을 가질 수 있다. 또한 파마리터와 결과를 위한 클래스로 직접 설정할 수도 있으며, 메모리내에 생산된 결과를 캐싱하기위해 캐쉬모델을 사용하도록 설정 할 수도 있다.

<statement id=*”statementName”*

                           [ parameterClass=*”some.class.Name”*]

                           [ resultClass=*”some.class.Name”*]

                           [ parameterMap=*”nameOfParameterMap”*]

                           [ resultMap=*”nameOfResultMap”*]

                           [ cacheModel=*”nameOfCache”*]

                           [ timeout=*“5”*]>

             select \* from PRODUCT where PRD\_ID = [?|#propertyName#]

             order by [$simpleDynamic$]

</statement>

위 구문에서 [괄호] 부분은 옵션이고 몇몇의 경우에만 혼합하여 사용할 수 있다. 위의 구문의 <statement>요소는 어떤타입의 SQL문이든 사용할 수잇는 요소이다. 즉, select, insert, update, delete 구문을 모두 사용할 수있는 요소라 생각 하면 된다.

**#구문타입**

|  |  |  |  |
| --- | --- | --- | --- |
| Statemanet 요소 | 속성 | 하위 요소 | 메소드 |
| <statement> | id  parameterClass  resultClass  parameterMap  resultMap  cacheModel  resultSetType  fetchSize  xmlResultname  remapResults  timeout | 모든 동적요소 | insert  update  delete  모든 쿼리 메소드 |
| <insert> | id  parameterClass  parameterMap  timeout | 모든 동적 요소  <selectKey> | insert  update  delete |
| <update> | id  parameterClass  parameterMap  timeout | 모든 동적 요소 | insert  update  delete |
| <delete> | id | 모든 동적 요소 | insert |

**#SQL의 재사용**

 SqlMaps를 사용할 때, 종종 SQL문의 일부가 중복되는 것을 보곤한다. 예를들어, from절이나 제약조건들이 그런경우다. iBatis는 중복된 SQL문의 일부를 재사용할수 있는 요소를 제공한다.

       <select id=*"getTestInfoCnt"* resultClass=*"int"*>

             SELECT CONUT(\*) AS cnt

             FORM testDB

             WHERE TEST\_NO = 6

       </select>

       <select id=*"getTestInfoId"* resultClass=*"board.TestClass"*>

             SELECT id

             FORM testDB

             WHERE TEST\_NO = 6

       </select>

위와 같은 SQL이 존재할때 조건절인 WHERE 절이 중복된다. 이럴경우 조건절만 따로 빼서 <sql>요소를 통하여 하나의 요소로 정의한후 <include>요소를 통하여 재사용 할 수 있다.

       <sql id=*"testSql"*>

             FORM testDB

             WHERE TEST\_NO = 6

       </sql>

       <select id=*"getTestInfoCnt"* resultClass=*"int"*>

             SELECT CONUT(\*) AS cnt

             <include refid=*"testSql"*/>

       </select>

       <select id=*"getTestInfoId"* resultClass=*"board.TestClass"*>

             SELECT id

             <include refid=*"testSql"*/>

       </select>

**#자동 생성 키**

 대부분의 DBMS는 기본키(PK)필드의 자동생성을 지원한다. SQL Map은 <insert>요소의 <selectKey>를통해 자동생성키의 반환을 지원하며 선생성키(pre-genetated)와 후생성키(post-generated)모두 지원한다.

<!-- Oracle SEQUENCE Example -->

       <insert id=*"test-ORACLE"* parameterClass=*"board.test"*>

             <selectKey resultClass=*"int"*>

                    SELECT TEST\_SEQUENCE.NEXTVAL AS ID FROM DUAL

             </selectKey>

             INSERT INTO testDB (ID, DESCRIPTION)

             VALUES (#id#,#description#)

       </insert>

<!-- Microsoft SQL Server IDENTITY Column Example -->

       <insert id=*"test-MS-SQL"* parameterClass=*"board.test"*>

             INSERT INTO testDB (ID, DESCRIPTION)

             VALUES (#id#,#description#)

             <selectKey resultClass=*"int"*>

                    SELECT @@IDENTITY AS ID

             </selectKey>

       </insert>

<selectKey>요소가 insert SQL 앞에있다면 insert 구문보다 먼저 실행된다. 하지만 insert문 뒤에 있다면 뒤에 실행된다. iBatis 버전 2.2.0이나 그 이후 버전에서, 원한다면 구문의 실행 순서를 명시적으로 바꿀 수 있다. <selectKey>요소의 type속성에 "pre"나 "post"를 명시해준다면 위치에 상관없이 type에 지정된 속성에 따라 <selectKey>요소가 실행이 되게 된다.

* **30909**

**iBatis 3 . 매핑구문 - 1**[**보기**](http://javaclass1.tistory.com/152)

**#parameterClass**

 parameterClass 속성값은 자바클래스의 전체경로(패키지)를 포함한 이름이다. parameterClass 속성은 옵션이지만 사용이 굉장이 추천되며, 프레임워크 성능을 향상시키는 만큼 statement에 전달하는 파라미터를 제한하는데 사용한다.

<insert id=*"insertTest"* parameterClass=*"board.Test"*>

       insert into testDB values (#id#, #description#, #price#)

</insert>

위의 구문은 board 패키지안의 Test라는 클래스의 id, description, price 프로퍼티를 자동으로 맵핑해준다.

**#parameterMap**

 parameterMap 속성은 좀더 상세한 파라미터값이 필요할때 사용하면 좋으며, JDBC PreparedStatement의 값 토큰과 매치되는 정렬된 파라미터 목록을 명시한다.

<parameterMap id=*"insert\_param\_map"* class=*"board.Test"*>

       <parameter property=*"id"* />

       <parameter property=*"description"* />

</parameterMap>

<insert id=*"insertTest"* parameterMap=*insert\_param\_map*>

       insert into testDb (ID, DESCRIPTION) values (?,?)

</insert>

위의 코드에서 parameterMap의 첫번째 프로퍼티는 SQL문에서의 첫번째 값토큰("?")에 매치되고 두번째 프로퍼티는 두번째 값토큰에 매치된다.

**#resultClass**

 resultClass 속성값은 자바 클래스의 전체경로를 포함(패키지)를 포함한 이름이며, ResultSetMetaData에 기반한 JDBC ResultSet에 자동 매핑되는 클래스를 명시하면 된다. 자바빈즈의 프로퍼티와 ResultSet의 컬럼이 매치될 때마다 프로퍼티는 컬럼값과 함께 생성이 된다.

<select id=*"getTest"* parameterClass=*"int"* resultClass=*"board.Test"*>

       SELECT

       ID as id,

       BIRTH\_DATE as birthDate,

       FROM testDB

       WHERE ID = #value#

</select>

위의 코드에서 Test클래스는 id와, birthDate 프로퍼티를 가진다. 위의 코드처럼 resultClass에 선언된 클래스의 프로퍼티와 SQL상에서 별칭으로 컬럼 이름을 매칭시켜주면 자동으로 Test클래스에 맞게 데이터를 변환하여 반환하게 된다.

**#resultMap**

 resultMap속성값은 명시된 reslutMap 요소의 이름이다. SQL을 사용하여 반환되는 resultSet으로 부터 인스턴스의 프로퍼티와 컬럼이 매핑되게끔 설정할 수 있다.

<resultMap id=*"get\_test\_result"* class=*"board.Test"*>

       <result property=*"id"* column=*"ID"* />

       <result property=*"birthDate"* column="*BIRTH\_DATE"* />

</resultMap>

<select id=*"getTest"* resultMap=*"get\_test\_result"*>

       select \* from testDB

</select>

위 코드에서 SQL쿼리로부터 반환되는 ResultSet은 resultMap정의를 사용해서 Test인스턴스에 매핑되게 한다. resultMap은 id프로퍼티가 ID 컬럼과, birthDate프로퍼티가 BIRTH\_DATE컬럼과 매핑되도록 설정하고 있다. 여기서 resultSet 내 반환되는 모든 컬럼에 매핑할 필요는 없다.