\*IBatis 툴 설정 하기

1.ibator 플러그인 다운로드 후 설치  
ibator 플러그인 홈페이지:http://ibatis.apache.org/ibator.html  
ibator 플러그인 설치 주소: http://ibatis.apache.org/tools/ibator (New Install)  
  
2.자바 일반 프로젝트 생성 - 생성된 프로젝트 오른쪽 마우스 - new - other - apache iBatis ibator - configurationFile - next - finish

<자동 설정 파일 생성>  
<?xml version=*"1.0"* encoding=*"UTF-8"* ?>

<!DOCTYPE ibatorConfiguration PUBLIC "-//Apache Software Foundation//DTD Apache iBATIS Ibator Configuration 1.0//EN" "http://ibatis.apache.org/dtd/ibator-config\_1\_0.dtd" >

<ibatorConfiguration >

<ibatorContext id=*"context1"* >

<jdbcConnection driverClass=*"???"* connectionURL=*"???"* userId=*"???"* password=*"???"* />

<javaModelGenerator targetPackage=*"???"* targetProject=*"???"* />

<sqlMapGenerator targetPackage=*"???"* targetProject=*"???"* />

<daoGenerator targetPackage=*"???"* targetProject=*"???"* type=*"GENERIC-CI"* />

<table schema=*"???"* tableName=*"???"* >

<columnOverride column=*"???"* property=*"???"* />

</table>

</ibatorContext>

</ibatorConfiguration>

설정하기 (ibatorConfig.xml)

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

<!DOCTYPE ibatorConfiguration PUBLIC "-//Apache Software Foundation//DTD Apache iBATIS Ibator Configuration 1.0//EN" "http://ibatis.apache.org/dtd/ibator-config\_1\_0.dtd" >

<ibatorConfiguration >

  <classPathEntry location="C:\ojdbc14.jar" /><!-- 실제 드라이버가 있는 주소 -->  
 <classPathEntry location=" C:\ojdbc14.jar"/>

<ibatorContext id=*"context1"* >

<jdbcConnection driverClass=*"* oracle.jdbc.driver.OracleDriver*"*

connectionURL=*"*jdbc:oracle:thin:@127.0.0.1:1521:XE*"*

userId=*"hr"* password=*"1004"* />

<javaModelGenerator targetPackage=*"*kr.co.bit.lecture.board.model"

targetProject=*"*iBator*"* /><!--  어떤 프로젝트의 어떤 패키지로 생성되어 들어갈것인가--> <sqlMapGenerator targetPackage=*"*kr.co.bit.lecture.board.dao.ibatis.maps*"*

targetProject=*"*iBator*"* /><!--매핑파일 생성 위치지정 -->  
 <daoGenerator targetPackage=*"*kr.co.bit.lecture.board.dao.ibatis.maps*"*

targetProject=*"*iBator*"*

type=*"GENERIC-CI"* /><!-- boardDaoiBatis만들기 -->  
 <table schema=*"* scott*"* tableName=*"*EMP*"* domainObjectName="Board"><!-- 스키마 생성 (domainObjectName추가 옵션)-->

</table>

</ibatorContext>

</ibatorConfiguration>

:만약 패키지가 많으면(패키지를 나누면 Board라는 패키지 말고 다른곳에 저장하고 싶으면) <ibatorContext></ibatorContext>까지 여러개를 만들면 됨  
4.ibatorConfig.xml 설정후  
ibatorConfig.xml- 오른쪽 단추 - 제너레이트~~- 자동으로 src폴더에 자동 생성

## [iBatis - 기본 설정](http://krespo.net/32)

**1. iBatis Library를 추가한다.  
2. SqlMapConfig.xml 파일을 생성한다**.  
<?xml version="1.0" encoding="UTF-8"?>   
<!DOCTYPE sqlMapConfig  PUBLIC "-//ibatis.apache.org//DTD SQL Map Config 2.0 //EN"  
 "<http://ibatis.apache.org/dtd/sql-map-config-2.dtd>">

<sqlMapConfig>  
    <settings  
   cacheModelsEnabled="true"  
   enhancementEnabled="true"  
   lazyLoadingEnabled="true"  
   maxRequests="32"  
   maxSessions="10"  
   maxTransactions="5"  
   useStatementNamespaces="false"  />  
   
<!-- JNDI사용 -->  
   <transactionManager type="JDBC" >  
  <dataSource type="JNDI">  
   <property name="DataSource" value="java:comp/env/jdbc/XE"/>  
  </dataSource>  
   </transactionManager>  
 <sqlMap resource="com/myhome/info/sql/SqlMap.xml" />  
</sqlMapConfig>

위의 설정은 JNDI를 사용하기 때문에 JNDI설정을 해 주어야 한다.  
1. Server 의 server.xml파일에 다음을 추가한다.  
( <GlobalNamingResources> 태그 안에 선언한다)  
<Resource auth="Container"   
        driverClassName="oracle.jdbc.driver.OracleDriver"   
        factory="org.apache.commons.dbcp.BasicDataSourceFactory"   
        maxActive="20"   
        maxIdle="10"   
         maxWait="-1"   
        name="jdbc/XE"   
        username="user01"  
        password="user01"   
        type="javax.sql.DataSource"   
        url="jdbc:oracle:thin:@localhost:1521:XE" />  
  
2. Server 의 context.xml 파일의 가장 하단에 다음을 추가 한다.  
<ResourceLink global="jdbc/XE" name="jdbc/XE" type="javax.sql.DataSource"/>

※JNDI말고 다른 방법으로 사용 할 수 있다. SqlMapConfig.xml 파일에 다음을 추가 한다.  
1.**simple datasource type**  사용  
<transactionManager type="JDBC" commitRequired="false">  
        <dataSource type="SIMPLE">  
            <property name="JDBC.Driver" value="oracle.jdbc.driver.OracleDriver"/>  
            <property name="JDBC.ConnectionURL" value="jdbc:oracle:thin:@localhost:1521:XE"/>  
            <property name="JDBC.Username" value="user01"/>  
            <property name="JDBC.Password" value="user01"/>  
        </dataSource>  
    </transactionManager>  
  
2.**DBCP datasource type** 사용  
<properties resource=" info/model/dao/db.properties " />  
         <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"/>  
  </dataSource>  
  </transactionManager>   
  
**3.SqlMap.xml 파일을 생성 한다.**<?xml version="1.0" encoding="UTF-8"?>  
<!DOCTYPE sqlMap PUBLIC "-//ibatis.apache.org//DTD SQL Map 2.0//EN"  
 "<http://ibatis.apache.org/dtd/sql-map-2.dtd>">

<sqlMap>  
  <typeAlias alias="Info" type="com.myhome.info.dto.InfoDTO"/><!-- alias : 별칭을 잡겠다는 의미 -->  
  <insert id="infoInsert" parameterClass="Info"><!-- typeAlias한 Info를 넣는다 없으면 com.. 그대로 넣어도 된다. -->  
   INSERT INTO INFO  
    (NUM, NAME, SEX, TEL, WDATE) VALUES  
    (NUMSEQ.NEXTVAL, #name#, #sex#, #tel#, #wdate#)

<!-- ## Info 안에 가지는 프로퍼티 대소문자가 정확하게 맞아야 한다. -->  
  </insert>  
    
  <statement id="infoList" resultClass="Info">  
    SELECT NUM, NAME, SEX, TEL, WDATE  
    FROM INFO  
  </statement>  
  </sqlMap>

**4.SQLManager.java 파일 생성**  
package com.myhome.manager;

import com.ibatis.common.resources.Resources;  
import com.ibatis.sqlmap.client.SqlMapClient;  
import com.ibatis.sqlmap.client.SqlMapClientBuilder;

public abstract class SQLManager {  
   
 private SqlMapClient sc = null;  
 public SQLManager(){  
  try{  
   sc = SqlMapClientBuilder.buildSqlMapClient(  
            Resources.getResourceAsReader("com/myhome/info/sql/SqlMapConfig.xml"));  
  }catch(java.io.IOException ie){  
   ie.printStackTrace();  
  }  
 }  
 public SqlMapClient getSqlMap(){  
  return sc;   
     }  
}  
  
**5.DAO에서 다음과 같이 사용**  
package com.myhome.info.dao;

import com.myhome.info.dto.InfoDTO;  
import com.myhome.manager.SQLManager;  
import java.sql.SQLException;  
import java.util.List;

public class InfoDAO extends SQLManager{  
 /\*등록부분\*/  
 public void register(InfoDTO dto) throws SQLException{  
  this.getSqlMap().insert("infoInsert",dto);  
 }  
   
 /\*회원 리스트 부분\*/  
 @SuppressWarnings("unchecked")  
 public List<InfoDTO> getAllQuery(){  
  List<InfoDTO> list = null;  
  try {  
   list = (List<InfoDTO>)this.getSqlMap().queryForList("infoList");  
  } catch (SQLException e) {  
   e.printStackTrace();  
  }  
  return list;  
 }  
}

☆SqlMapConfig.xml의 추가 사항  
  
1.전체적인 내용  
<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<!DOCTYPE sqlMapConfig

 PUBLIC "-//ibatis.apache.org//DTD SQL Map Config 2.0 //EN"

 "http://ibatis.apache.org/dtd/sql-map-config-2.dtd">

**<sqlMapConfig>**

**<settings/>**

**<properties resource=" db.properties " />**

**<transactionManager></transactionManager>**

**<sqlMap></sqlMap>**

**</sqlMapConfig>**

위에서 가장 중요한 것은 transactionManager 와 sqlMap부분이며 setting과 properties resource 는 선택 사항이다.  
2. settings의 내용들  
\*cacheModelsEnalbe ="true" : SqlMapClient를 위한 모든 캐쉬모델을 가능하게 하거나 가능하지 않게 한다.  
이것은 디버깅시 도움이 된다.  
\*enhancernetEnable="true" : 향상된 lazy로딩처럼 최적화된 자바빈즈 속성 접근을 위해 런타임시 바이트코드 향상을 가능하게 한다.  
\*lazyLoadingEnabled="true" : SqlMapClient 를 위한 모든 lazy로딩ㅇ르 가능하게 하거나 가능하지 않게 한다.  
\*maxRequests="32" : 한꺼번에 SQL을 실행할 스래드 수이고 2에n승으로 확장할 수 있고 128이 맥시멈이다.  
\*maxSession="10" : 주어진 시간동안 활성화 될 수 있는 세션 수  
\*maxTransaction="5" : 한꺼번에 SqlMapClient.startTransaction()에 들어갈 수 있는 스레드의 최대 갯수  
\*useStatementNamespaces ="false"

3. db.properties 파일 설정  
driver=oracle.jdbc.driver.OracleDriver

url=jdbc:oracle:thin:@localhost:1521:XE

username=user01

password=mypwd

IBATIS기본 설정

* ORM(Object Relational Mapping)
* 데이터베이스의 레코드를 직접 Java Class에 매핑하고 XML 형태의 SQL 문을 실행하는 Persistence Layer를 담당하는 Framework 개발 모델
* /WEB-INF/classes/sqlMapConfig.xml : DB 정보와 mapper file 정보 등의 환경 설정 파일
* mapper file (~Map.xml) : 실제 SQL문을 작성하는 XML 파일

### sqlMapConfig.xml

|  |  |
| --- | --- |
| settings | * cacheModelsEnabled : true. 캐시 가능 * enhancementEnabled : false. 런타임시 바이트 코드 기능 향상 * lazyLoadingEnabled : true * maxRequests : 512 * maxSessions : 128 * maxTransactions : 32 * useStatementNamespaces : false |
| typeAlias | <typeAlias alias="Account" type="org.apache.domain.Account" /> |
| typeHandler | <typeHandler javaType="String" callback="com.StringTypeHandler"/> |
| properties | <properties resource="SqlMapConfig.properties" /> |
| sqlMap | <sqlMap resource="sqlMaps/board.xml" /> |
| transactionManager | <transactionManager type="JDBC">  <dataSource type="DBCP">  <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="false" />  </dataSource>  </transactionManager> |

### Mapper file

* Parameter, Query 문에서 해당 변수를 매개 변수로 사용
* #name#
* #name:VARCHAR#, #mailDomainSeq:INTEGER#
* #nameArray[]#
* Variable : $~$, 해당 변수로 Query문을 직접 생성
* ? :
* <![CDATA[ <= ]]>

|  |  |
| --- | --- |
| resultMap | <resultMap class="boardbean" id="boardbean">  <result property="postId" column="post\_Id"/>  <result property="seq" column="SEQ" jdbcType="INTEGER" />  <result property="detail" resultMap="Catetory.ChildMap"/>  </resultMap> |
| parameterMap | <parameterMap class="boardbean" id="boardbean">  <parameter property="postId" />  </parameterMap> |
| select | |  |  | | --- | --- | | id |  | | parameterClass | 전달되는 parameter의 class를 지정 | | parameterMap | 전달되는 parameter | | resultClass | 반환되는 resultMap의 class를 지정 | | resultMap | 반환되는 값 | | cacheModel |  | | Table Join시 | * resultMap 선언   <resultMap id="parent" class="parentMap">  <result property="child" resultMap="catetory.childMap"/>  </resultMap>   * SQL문 선언   from parentTable a  **left outer join childTable b on** a.SEQ = b.PARENT\_SEQ |   <select id="readDeletedScheduleListAfterSyncTime" parameterClass="map" resultMap="schedulerResult">  select \*  from parentTable a  left outer join childTable b on a.SEQ = b.PARENT\_SEQ  where mail\_user\_seq = #mailUserSeq#  and outlook\_sync = 'delete'  and modify\_time <![CDATA[ >= ]]> #syncTime#  order by start\_date asc  limit #skipResult#, #maxResult#  </select> |
| selectKey | <selectKey resultClass="int" keyProperty="bbsId">  select case when max(bbs\_id) is null then 1 else max(bbs\_id)+1 end  from tscheduler  </selectKey> |
| insert | <insert id="saveShareFolder" parameterClass="map">  insert into shf\_shr\_folder (mail\_user\_seq, folder\_uid)  values (#userSeq#, #fUid#)  </insert> |
| update | <update id="modifyAuthShareTargetFolder" parameterClass="webfoldershare">  update shf\_shr\_target set share\_auth = #auth#, modify\_time = #curTime#  where folder\_uid = #fuid# and share\_value = #email#  </update> |
| delete | <delete id="deleteShareTargetFolder" parameterClass="int">  delete from shf\_shr\_target where folder\_uid = #value#  </delete> |
| dynamic | * prepend : 동적 SQL문을 생성하기 전에 추가할 문자열 * open : 시작하기 전에 추가할 문자열 * close : 시작 후에 추가할 문자열 * conjunction : SQL문 생성 사이 사이에 추가할 문자열 * compareProperty : 비교되는 property * compareValue : 비교할 값   <dynamic prepend="WHERE">  <iterate property="postId"  conjunction="," open="post\_id IN (" close=")" >  #postId[]#  </iterate>  </dynamic>  <dynamic prepend="and">  <isGreaterThan property="userSeq" compareValue="0">  mail\_user\_seq = #userSeq#  </isGreaterThan>  </dynamic> |
| 연산 | * iterate : <iterate property="postId" conjunction="," open="post\_id IN (" close=")" > * isEqual : <isEqual property="type" compareValue="name"> * isNotEqual : <isNotEqual property="a" compareValue="K"></isNotEqual> * isGreaterEqual : <isGreaterEqual property="pageLineCnt" compareValue="0" prepend=","> * isGreaterThan : <isGreaterThan property="userSeq" compareValue="0"> * isLessEqual * isLessThan * isNull : <isNull property="a"></isNull> * isNotNull : <isNotNull property="userSeqs"> * isEmpty : <isEmpty property="a"></isEmpty> * isNotEmpty : <isNotEmpty property="a"></isNotEmpty> * isPropertyAvailable : <isPropertyAvailable property="memberName" prepend=","> * isNotPropertyAvailable : <isNotPropertyAvailable property="a"></isNotPropertyAvailable> * isParameterPresent : * isNotParameterPresent : |
| cacheModel | <cacheModel id="decisionPathCache" type="MEMORY" readOnly="true" serialize="false">  <flushInterval hours="24"/>  <flushOnExecute statement="SystemConfig.allCacheDelete"/>  <property name="reference-type" value="SOFT"/>  </cacheModel> |
| sql | * 여기서 선언한 SQL문을 include 태그로 호출하여 사용 한다.   <sql id="getMember\_Where\_Clause" >  </sql> |
| include | * sql 태그로 선언한 SQL문을 삽입 한다.   <include refid="getMember\_Where\_Clause" /> |
| procedure |  |
| statement | <statement id="test" parameterClass="package.ClassName" > |
| Collection |  |
| RowHandler |  |

* Mapper file 사용법
* Cache (CacheModel)
* Sub Query 사용
* List<MessageVO>

## DAO 프로그램

### SqlMapClient

Reader reader = Resources.getResourceAsReader("SqlMapConfig.xml");

SqlMapClient sqlMap = SqlMapClientBuilder.buildSqlMapClient(reader);

### query

* Select

User user = (User) sqlMap.queryForObject(id, parameter);

sqlMap.queryForList(id, parameter);

sqlMap.queryForMap(id, parameter, key);

* Insert

Map map = new HashMap();

map.put("mailUserSeq", userSeq);

map.put("agingDay", agingDay);

sqlMap.insert("~", map);

* Update

HashMap<String, Object> param = new HashMap<String, Object>();

param.put("mailDomain", domainSeq);

param.put("userSeq", userSeq);

sqlMap.update("~", param);

* Delete

sqlMap.delete("~", ~);

## [iBATIS - API를 이용한 예제](http://pelican7.egloos.com/2579625" \o "iBATIS - API를 이용한 예제) [iBATIS](http://pelican7.egloos.com/category/iBATIS)

각각의 경우에 맵핑된 statement의 이름은 첫번째 파라미터로 넘겨진다. 이 이름은 위에서 서술된 <statement>요소의 name속성에 대응된다. 추가적으로 파라미터객체는 옵션적으로 전달할수 있다. null파라미터객체는 만약 기대되는 파라미터가 없다면 전달될수 있다. 행위의 남겨진 차이점은 아래에서 간단하게 설명된다.

**insert(), update(), delete()**

 이 메소드들은 update statement를 위해 특별히 의미된다. 밑의 쿼리 메소드중에 하나를 사용해서 update statement를 수행하는 것은 불가능하다. 어쨌든 이것은 애매한 의미이고 드라이버에 의존적이다.executeUpdate()의 경우에 statement는 간단하게 수행되고 영향을 받는 많은 수의 row가 반환된다.

**queryForObject()**

executeQueryForObject()의 두가지 버전이 있다. 하나는 새롭게 할당된 객체를 반환하는 것이고 다른 하나는 파라미터처럼 전달된 미리할당된 객체를 사용하는것이다. 후자의 경우 하나의 statement보다 많은 수에 의해 생성되는 객체에 유용하다.

**queryForList()**

queryForList()에는 세가지 버전이 있다. 첫번째는 쿼리를 실행하고 쿼리로부터 모든 결과를 반환하는것이고 두번째는 스킵되는 결과물의 수(이를 테면 시작지점)를 지정할수 있고 반환되는 레코드의 최대갯수도 지정할수 있다. 이것은 전체데이터를 반환하고 싶지 않은 굉장히 큰 데이터셋과 작업을 할 때 가치가 있다. 마지막으로 세번째는 row핸들러를 가진다. 이 메소드는 대개의 칼럼과 rows보다 result객체를 사용해서 row에 의해 결과를 처리하도록 한다. 이 메소드는 전형적인 이름과 파라미터 객체를 넘기지만 RowHandler를 가진다. row핸들러는 RowHandler인터페이스를 구현하는 클래스의 인스턴스이다. RowHandler인터페이스는 다음처럼 오직 하나의 메소드만 가진다.

 public void handleRow (Object object, List list);

이 메소드는 데이터베이스로부터 반환되는 각각의 row를 위한 RowHandler에서 호출될것이다. 이것은 쿼리결과를 처리하기위해 깔끔하고 간단하며 확장가능한 방법이다. RowHandler의 사용법 에제를 위해 아래 섹션의 예제를 보라. 리스트 파라미터는 queryForList()메소드로부터 반환될 List인터페이스의 인스턴스이다. 당신은 리스트의 result객체의 아무것도, 몇몇 또는 모든 것을 추가할수도 있다. 만약 당신이 100만개의 row로 작업을 한다면 리스트에 그것들을 모두 넣는 것은 좋은 생각이 아니다.

**queryForPaginatedList()**

이것은 이전, 다음 버튼으로 데이터를 탐색할 때 데이터의 일부를 관리할수 있는 리스트를 반환하는 매우 유용한 메소드이다. 이것은 쿼리로부터 반환된 레코드의 일부만을 표시하는 사용자 인터페이스를 구현하는데 유용하다. 10,000개의 필드를 반환하는 검색엔진이 있지만 한번에 100개만 표시해야 하는 예제가 있다. PaginatedList인터페이스는 페이지를 통한 탐색(nextPage(), previousPage(), gotoPage())과 페이지의 상태를 체크(isFirstPage(), isMiddlePage(), isLastPage(), isNextPageAvailable(), isPreviousPageAvailable(), getPageIndex(), getPageSize())하는 메소드를 포함한다. 유효한 레코드의 총 개수가 PaginatedList인터페이스로부터 접근가능하지 않더라도 이것은 기대되는 결과갯수를 세는 두번째 statement를 간단히 수행함으로써 쉽게 달성할 수 있을 것이다. 반면에 PaginatedList를 사용하면 지나친 부하가 발생할수도 있다.

**queryForMap()**

이 메소드는 리스트로 결과의 collection을 로드하는 대안을 제공한다. 대신에 이것은 keyProperty처럼 전달된 파라미터에 의해 결과를 키(key)화된 map으로 로드한다. 예를 들면 만약 Employee객체의 collection을 로드한다면 당신은 그것들을 employeeNumber프라퍼티에 의해 키(key)화된 map으로 로드할것이다. Map의 값은 전체 employee객체가 될수도 있고 valueProperty라고 불리는 두번째 파라미터내 정의된 employee개체로 부터의 다른 프라퍼티가 될수도 있다. 예를들면 당신은 employee숫자에 의해 키(key)화된 employee이름의 map을 원할지도 모른다. result객체처럼 Map타입을 사용하는 개념을 사용하는 메소드를 혼란스러워하지 마라. 이 메소드는 result객체가 자바빈즈나 Map인지에따라 사용될수 있다.

Example 1: Update (insert, update, delete) 수행하기.

sqlMap.startTransaction();  
Product product = new Product();  
product.setId (1);  
product.setDescription (“Shih Tzu”);  
int rows = sqlMap.insert (“insertProduct”, product);  
sqlMap.commitTransaction();

Example 2: 객체를 위한 쿼리 수행하기. (select)

sqlMap.startTransaction();  
Integer key = new Integer (1);  
Product product = (Product)sqlMap.queryForObject (“getProduct”, key);  
sqlMap.commitTransaction();

Example 3: 미리 할당된 result객체를 가지고 객체를 위한 쿼리 수행하기.

sqlMap.startTransaction();  
Customer customer = new Customer();  
sqlMap.queryForObject(“getCust”, parameterObject, customer);  
sqlMap.queryForObject(“getAddr”, parameterObject, customer);  
sqlMap.commitTransaction();

Example 4: 리스트를 위한 쿼리 수행하기. (select)

sqlMap.startTransaction();  
List list = sqlMap.queryForList (“getProductList”, null);  
sqlMap.commitTransaction();

Example 5: 자동 커밋

// When startTransaction is not called, the statements will  
// auto-commit. Calling commit/rollback is not needed.  
int rows = sqlMap.insert (“insertProduct”, product);

Example 6: 결과 경계를 가지고 리스트를 위한 쿼리 수행하기.

sqlMap.startTransaction();  
List list = sqlMap.queryForList (“getProductList”, null, 0, 40);  
sqlMap.commitTransaction();

Example 7: RowHandler를 가지고 쿼리 수행하기. (select)

public class MyRowHandler implements RowHandler {  
    public void handleRow (Object object, List list) throws SQLException

    {  
        Product product = (Product) object;  
        product.setQuantity (10000);  
        sqlMap.update (“updateProduct”, product);  
        // Optionally you could add the result object to the list.  
        // The list is returned from the queryForList() method.  
    }  
}  
sqlMap.startTransaction();

RowHandler rowHandler = new MyRowHandler();  
List list = sqlMap.queryForList (“getProductList”, null, rowHandler);  
sqlMap.commitTransaction();

Example 8: 페이지 처리된 리스트를 위한 쿼리 수행하기. (select)

PaginatedList list = sqlMap.queryForPaginatedList (“getProductList”, null, 10);  
list.nextPage();  
list.previousPage();

Example 9: map을 위한 쿼리 수행하기.

sqlMap.startTransaction();  
Map map = sqlMap.queryForMap (“getProductList”, null, “productCode”);  
sqlMap.commitTransaction();  
Product p = (Product) map.get(“EST-93”);

--------------------------------------------------------

dbconnect.properties

--------------------------------------------------------

//DB접속을 하기 위한 설정파일

//키=값

driver=com.mysql.jdbc.Driver

url=jdbc:mysql://localhost:3306/userdb

username=root

password=1234

--------------------------------------------------------

User.xml

--------------------------------------------------------

<?xml version="1.0" encoding="euc-kr" ?>

<!DOCTYPE sqlMap

    PUBLIC "-//ibatis.apache.org//DTD SQL Map 2.0//EN"

    "<http://ibatis.apache.org/dtd/sql-map-2.dtd>">

<sqlMap namespace="User">

  <!-- 리절트 맵 정의 -->

  <resultMap id="UserResult" class="model.User">

//DTO파일의 지정 여기서는 model패키지의 User.java

//아래는 java클래스의 멤버변수이다.

    <result property="userId" column="USER\_ID"/>

    <result property="userPW" column="USER\_PW"/>

    <result property="userName" column="USER\_NAME"/>

  </resultMap>

  <!-- select 쿼리문 정의 -->

  <select id="selectAllUsers" resultMap="UserResult">

    select \* from USER

  </select>

</sqlMap>

--------------------------------------------------------

sqlMapConfig.xml

--------------------------------------------------------

<?xml version="1.0" encoding="euc-kr" ?>

<!DOCTYPE sqlMapConfig PUBLIC "-//iBATIS.com//DTD SQL Map Config 2.0//EN"

"<http://www.ibatis.com/dtd/sql-map-config-2.dtd>">

<sqlMapConfig>

  <properties resource="dbconnect.properties" />

//외부에 존재하는 프로퍼티 파일을 불러오기 위해 사용한다.

//이 파일에는 DB에 접속하기 위한 드라이버,url,username,password등이 기록되어 있다.

//프로퍼티의 내용은 "키=값" 의 형태로 지정된다.

  <settings

    useStatementNamespaces="false"

    cacheModelsEnabled="true"

    enhancementEnabled="true"

    lazyLoadingEnabled="true"

    maxRequests="32"

    maxSessions="10"

    maxTransactions="5"

    />

//옵션과 최적화를 위한 값을 지정한다.

|  |  |
| --- | --- |
| 속성 | 설명 |
| cacheModelsEnabled | sqlMapClient를 위한 모든 캐시 모델의 가능 유무. Default는 true(enabled) |
| enhancementEnabled | 런타임 시 바이트코드 향상의 가능 유무 Default는 false(disabled) |
| lazyLoadingEnabled | 모든 늦은 로딩의 가능 유무. Default는 true(enabled)  (객체를 사용하기 전의 초기화를 할 것인지 결정) |
| maxRequests | 동시에 SQL문을 수행할 수 있는 쓰레드의 수. 세팅값보다 많은 쓰레드는 다른 쓰레드가 수행을 완료할 때까지 블록 Default는 512 |
| maxSessions | 주어진 시간동안 활성화 될 수 있는 세션의 수 Default는128 |
| maxTransactions | 한꺼번에 SqlMapClient.startTransaction()에 들어갈 수 있는 쓰레드의 최대 개수.  세팅값보다 많은 쓰레드는 다른 쓰레드가 나올때 까지 블록 Default는 32 |
| useStatementNamespaces | 이 세팅을 가능하게 하면 sqlmap이름과 statement이름으로 구성된 전체적인 이름(fully qualified name)으로 맵핑된 statement를 참조해야 함. Default는 false(disabled)  예)quertForObject("sqlMapName.statementName"); |

  <transactionManager type="JDBC">

//트렌젝션에 관련된 값을 세팅하고 하위 소스값을 지정하는 데이터 소스를 가진다.

    <dataSource type="DBCP">

//하위 소스 값을 지정하는 데이터 소스

|  |  |
| --- | --- |
| JDBC | connection commit()과 rollback() 메소드를 통해 트렌젝션을 제어하기위해 JDBC를 사용한다. |
| JTA | sql maps가 다른 데이터베이스나 트렌젝션 자원을 포함하는 더욱 넓은 범위의 트렌젝션을 포함 하도록하는 JTA 전역 트렌젝션을 사용한다. |
| EXTERNAL | 개발자가 직접 트렌젝션을 관리 할 때 사용하는 것으로 주로 분산 컴포넌트 기반의 시스템 환경에서 컨테이너가 트렌젝션을 관리하는 경우에 사용한다.  datasourse의 type속성값은 SIMPLE ,DBCP,JDNI중 하나를 사용할 수 있다. 이 값은 각각 iBatis SimpleDataSourse커넥션 풀링,Jakarta DBCP,JNDI를 통한 데이터 소스를 사용한다. |

//dbconnect.properties의 값을 사용하기 위해서는 ${키}의 형태로 사용한다.

      <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="false" />

    </dataSource>

  </transactionManager>

   <sqlMap resource="User.xml" />

//자바 파일DTO와 xml을 맵핑해주는 기능을 한다.

//DTO의 이름과 XML의 이름이 같아야 맵핑된다.

</sqlMapConfig>

여기서는 사용되지 않았지만 typeAlias라는 것이 있다.

이것은 별칭을 주어 사용 하는 것인데

JDBC,JTA,EXTERNAL의 세가지는 트렌젝션메니저에서 사용되고

SIMPLE,DBCP,JNDI는 데이터소스에서 사용되고 있다.

이분들은 사용자가 정의 해서 사용할 수 없다.