## [iBatis - iterate 태그로 동적 Query를 생성해 보자](http://krespo.net/100)

Query문 중 가끔 동적으로 어떠한 쿼리를 생성해야 할 경우가 있습니다..

예를 들면 어떠한 리스트에 있는 값을 통해 쿼리문을 생성한다던지 하는 경우 iBatis에서는 iterate 태그로 for문과 같은 역할을 하여 동적 쿼리를 생성 할 수 있습니다.

|  |  |
| --- | --- |
| 2  3  4  5  6  7  8 | <select id="getData" parameterClass="java.util.List"  resultClass="int">      SELECT    count(0)    FROM data\_tb      <iterate prepend="WHERE id IN" open="(" close=")"  conjunction="," >              #[]#       </iterate>  </select> |

위와 같이 parameterClass를 java의 List형태로 전달해 주면 동적쿼리를 생성할수 있습니다. 만약 list에 1,2,3,4 의 데이터가 들어가 있으면 결과적으로

다음과 같은 쿼리가 실행되는 것과 같습니다.

[?](http://krespo.net/100)

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <select id="getData" parameterClass="java.util.List" resultClass="int">      SELECT          count(0)      FROM data\_tb      WHERE id IN (1,2,3,4)  </select> |

이렇게  iterate를 사용하여 동적으로 쿼리문을 생성할 수 있습니다.  
또한 parameterClass가 list형태가 아닌 Map이 List를 가지고 있는 형태도 iterate의 property 속성을 통해 접근할 수 있습니다 만약 HashMap에 paramList라는 이름으로 list형이 들어가 있다고 가정하면  
예)  
 HashMap<String, Object> paramMap = new HashMap<String, Object>();  
List<Integer> paramList = new ArrayList<Integer>();  
paramMap.put("paramList", paramList);

[?](http://krespo.net/100)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | <select id="getData" parameterClass="HashMap" resultMap="myResult">      SELECT          \*      FROM data\_tb      <iterate property="paramList" prepend="WHERE id IN" open="("  close=")" conjunction="," >          #paramList[]#    </iterate>   </select> |

위와 같이 HashMap에 들어있는 list는 iterate의 property 속성을 통해 List를 가져 옵니다. ##사이는 HashMap에 put했던 list이름을 넣어주셔야 작동합니다.

## [iBATIS DynamicQuery 사용법](http://pelican7.egloos.com/2583678" \o "iBATIS DynamicQuery 사용법) [iBATIS](http://pelican7.egloos.com/category/iBATIS)

* by 가시고기
* 2010/04/15 10:20
* [pelican7.egloos.com/2583678](http://pelican7.egloos.com/2583678)
* 덧글수 : 0

이전에 살펴본 DynamicQuery부분에 사용한 옵션 외에 다양한 옵션이 제공 된다. 일반적으로 이런 옵션들을 이용할때 DAO에 쿼리가 박혀있을 경우 매우 복잡해진다. 따라서 스토어드 프로시저를 이용한다거나 하는 방식을 택하게 되는데, iBATIS에서 지원하는 옵션들을 살펴보자

1. <!--property가 존재 하는지 검사(map의경우 property를 key로 검색실패) -->
2. **<isPropertyAvailable** property=""**></isPropertyAvailable>**
3. <!--property가 존재 하지 않는지 검사(map의경우 property를 key로 검색실패) -->
4. **<isNotPropertyAvailable** property=""**></isNotPropertyAvailable>**
6. <!--property가 null인지 검사 -->
7. **<isNotNull** property=""**></isNotNull>**
8. <!--property가 null이 아닌지 검사 -->
9. **<isNull** property=""**></isNull>**
11. <!-- property가 비어있는지 검사("".equals) 검사-->
12. **<isEmpty** prepend=""**></isEmpty>**
13. <!-- property가 비어있지 않은지 검사(!"".equals) 검사-->
14. **<isNotEmpty** prepend=""**></isNotEmpty>**
16. <!-- property가 비교값과 같은지 검사-->
17. **<isEqual** property="" compareValue="" compareProperty=""**></isEqual>**
18. <!-- property가 비교값과 같지 않은지 검사-->
19. **<isNotEqual** property="" compareValue="" compareProperty=""**></isNotEqual>**
21. <!-- property가 비교값보다 큰지 검사 -->
22. **<isGreaterThan** property="" [compareValue="" || compareProperty=""]**></isGreaterThan>**
23. <!-- property가 비교값보다 큰거나 같은지  검사 -->
24. **<isGreaterEqual** property="" [compareValue="" || compareProperty=""]**></isGreaterEqual>**
25. <!-- propety가 작은지 검사 -->
26. **<isLessThan** property="" [compareValue="" || compareProperty=""]**></isLessThan>**
27. <!-- propety가 작거나 같은지  검사 -->
28. **<isLessEqual** property="" [compareValue="" || compareProperty=""]**></isLessEqual>**
30. <!-- property를 반복하며 추가. 반복시 값의 사이에 conjunction정의 문자 삽입 -->
31. **<iterate** property="" open="(" close=")" conjunction=","**></iterate>**

그냥 눈으로만 봐도 충분히 숙지할 수 있는 사항이므로 별도의 설명은 하지 않겠다. 조건에 property는 당연히 들어가는것이 원칙이다. 비교할 대상이나 검사할 대상이 없다면 뭐하러 이런 조건을 사용 하겠는가? 어렵지 않은 내용이지만 한가지 숙지해야 할 사항이 있다.   
  
바로 이항연산자 부분에서 compareValue와 compareProperty인데 다른 부분은 property를 이용해 비교했다. 같은 property가 들어가는걸로 보아 compareProperty는 프로퍼티 중에서 비교하는 것을알수 있다.즉 '같이 넘어온 객체의 프로퍼티와 비교'하여 구문을 추가할지 선택하는것이다.  
  
compareValue는 정적인 값이다. 예를들어 조건이 '성적이 100점 이하'일 경우 100은 정적인 값이다. 만약 이것을 compareProperty로 이용한다면 사용할때마다 '100'의 값을 셋팅해야할 것이다. 이럴때  compareValue를 이용하면 프로그램 코드에 별도로 관리할 필요가 없으므로 유용하게 사용할수 있다. 그럼 예제를 살펴보자.

**boardManager.xml**

1. **<typeAlias** alias="boardbean" type="pupustory.ibatis.beans.BoardBean"**/>**
3. **<resultMap** class="boardbean" id="boardbean"**>**
4. **<result** property="postId" column="post\_Id"**/>**
5. **<result** property="postTitle" column="post\_Title"**/>**
6. **<result** property="postWriter" column="post\_Writer"**/>**
7. **<result** property="postBody" column="post\_Body"**/>**
8. **</resultMap>**
10. **<parameterMap** class="boardbean" id="boardbean"**>**
11. **<parameter** property="postId" **/>**
12. **<parameter** property="postTitle"**/>**
13. **<parameter** property="postWriter"**/>**
14. **<parameter** property="postBody"**/>**
15. **</parameterMap>**
17. **<select** id="select.board" resultMap="boardbean"
18. parameterClass="boardbean" **>**
19. SELECT
20. A.POST\_ID AS POST\_ID
21. ,A.POST\_TITLE AS POST\_TITLE
22. ,A.POST\_WRITER AS POST\_WRITER
23. ,B.POST\_BODY AS POST\_BODY
24. FROM TB\_MAIN\_BOARD A, TB\_SUB\_BOARD B
25. WHERE A.POST\_ID = B.POST\_ID
26. **<dynamic** open="" close=""**>**
27. **<isNull**  prepend=" AND " property="postId"**>**
28. 11=1
29. **</isNull>**
30. **<isNotNull** prepend=" AND "  property="postId"**>**
31. a.post\_id = #postId#
32. **</isNotNull>**
34. **<isNotNull** prepend=" AND " property="postTitle"**>**
35. post\_title like '%' || #postTitle# || '%'
36. **</isNotNull>**
38. **<isNotNull** prepend=" AND " property="postWriter"**>**
39. post\_Writer like '%' || #postWriter# || '%'
40. **</isNotNull>**
42. **<isNotNull** prepend=" AND " property="postBody"**>**
43. post\_body like '%' || #postBody# || '%'
44. **</isNotNull>**

47. **</dynamic>**
48. **</select>**
50. **<insert** id="insert.main.board" parameterClass="boardbean"**>**
51. insert into tb\_main\_board
52. values(#postId#,#postTitle#,#postWriter#)
53. **</insert>**
55. **<insert** id="insert.sub.board" parameterClass="boardbean"**>**
56. insert into tb\_sub\_board(post\_id, post\_body)
57. values(#postId#,#postBody#)
58. **</insert>**
59. **</sqlMap>**

**BoardBean.java**

1. **package** pupustory.ibatis.beans;
3. **public** **class** BoardBean {
4. String postId;
5. String postTitle;
6. String postWriter;
7. String postBody;
8. **public** String getPostBody() {
9. **return** postBody;
10. }
11. **public** **void** setPostBody(String postBody) {
12. **this**.postBody = postBody;
13. }
14. **public** String getPostId() {
15. **return** postId;
16. }
17. **public** **void** setPostId(String postId) {
18. **this**.postId = postId;
19. }
20. **public** String getPostTitle() {
21. **return** postTitle;
22. }
23. **public** **void** setPostTitle(String postTItle) {
24. **this**.postTitle = postTItle;
25. }
26. **public** String getPostWriter() {
27. **return** postWriter;
28. }
29. **public** **void** setPostWriter(String postWriter) {
30. **this**.postWriter = postWriter;
31. }
32. **public** String toString() {
33. **return** "postId["+postId+"]"
34. +"postTitle["+postTitle+"]"
35. +"postWriter["+postWriter+"]"
36. +"postBody["+postBody+"]";
37. }
38. }

**BoardBiz.java**

1. **package** pupustory.ibatis.board;
3. **import** pupustory.ibatis.manager.DBManager;
4. **import** pupustory.ibatis.beans.BoardBean;
5. **import** java.sql.SQLException;
6. **public** **class** BoardBiz {
7. DBManager manager = DBManager.getInstance();
8. // 가상 데이터 삽입
9. **public** **void** insertVirData() **throws** SQLException{
10. BoardBean bean = **null**;
11. **final** **int** MAX\_BOARD\_DATA = 10;
13. manager.startTransaction();
14. manager.startBatch();
16. **for** (**int** i=0;i<MAX\_BOARD\_DATA;i++) {
17. bean = **new** BoardBean();
18. bean.setPostId(i+"");
19. bean.setPostTitle("제목! "+i);
20. bean.setPostWriter("pupustory"+i);
21. bean.setPostBody("본문 히히히히 헤헤"+i);
22. manager.getMapper().insert("insert.main.board", bean);
23. manager.getMapper().insert("insert.sub.board", bean);
24. }
26. manager.executeBatch();
27. manager.commitTransaction();
28. }
30. // 게시물 조회
31. **public** java.util.List getPost(BoardBean bean)
32. **throws** SQLException {
33. **return**
34. (java.util.List)manager.getMapper().queryForList("select.board",bean);
35. }
36. }

**StartApp.java**

1. **package** pupustory.ibatis.board;
2. **import** java.sql.SQLException;
3. **import** pupustory.ibatis.beans.BoardBean;
5. **public** **class** StartApp {
6. **public** **static** **void** main(String ar[]) **throws** SQLException{
7. BoardBiz biz = **new** BoardBiz();
8. //biz.insertVirData();
9. BoardBean bean = **null**;
10. java.util.List list = biz.getPost(bean);
11. **for** (**int** i=0;i<list.size();i++) {
12. System.out.println(list.get(i).toString());
13. }
15. bean = **new** BoardBean();
16. // id 3번 게시물 조회
17. bean.setPostId("3");
18. list = biz.getPost(bean);
19. **for** (**int** i=0;i<list.size();i++) {
20. System.out.println(list.get(i).toString());
21. }
23. // 본문에 '4'문자 포함한 게시물 조회
24. bean = **new** BoardBean();
25. //bean.setPostId("2");
26. bean.setPostBody("4");
27. list = biz.getPost(bean);
28. **for** (**int** i=0;i<list.size();i++) {
29. System.out.println(list.get(i).toString());
30. }
31. }
32. }

조회조건에 들어올 값을 확인하고, null이 아닐경우 검색해서 출력한다. equal같은 검색조건을 추가하려 했지만.. 귀찮기도해서 --; 어차피 사용법은 동일하다. 그리고 dynamic부분의 prepend는 삭제했다. 이유는 이미 join을 하기위해 where를 호출했고, 뒤에 올 조건이 있을지 없을지 미지수다. 따라서 삭제 했다. 1=1은 게시 전에 몇가지 만지작 거리다 남은 코드이므로 무시해도 무관하다.  
  
단항,이항 검색조건은 위와 같이 수행하면 되므로 어려울 것이 없다. 그렇다면 이번엔 이터레이터에 대해 알아볼 것이다. 이전에 설명 했지만 만약 사용자가 ' where board\_id in (....) '와 같이 몇건이 들어올지 모르는 경우에 유용 하다.   
  
이부분은 $$를 이용한 statment를 이용할 수도 있겠지만 이건 매우 위험하다. sql injection때문에 그렇다. 어플에서 .replace했다면 상관 없겠지만 그래도 프레임워크를 이용하니 .. 기능을 이용해 보도록 하자.  
  
먼저 IN ( ...) 부분엔 정확히 얼마의 문자가 올지 모른다. 한가지 유추할 수 있는 것은 여기엔 배열이 적합한 것 이다. 새로운 예를 만드는 것 보다 위의 코드를 조금 변경해 작성하도록 하자. 먼저 빈즈에 POST\_ID값이 저장될 배열 변수를 하나 추가한다.

1. **private** String rePostId;
2. **public** String getRePostId() {
3. **return** rePostId;
4. }
5. **public** **void** setRePostId(String rePostId) {
6. **this**.rePostId = rePostId;
7. }
9. **public** String toString() {//toString()는 값 확인을 위해 변경 함
10. **return** "postId [" +rePostId+"]"
11. +" postTitle [" +postTitle+"]"
12. +" postWriter [" +postWriter+"]"
13. +" postBody [" +postBody+"]"
14. +" parentPostId [" +parentPostId+"]";
15. }

이제 중요한 sqlmap부분이다.

1. **<?xml** version="1.0" encoding="UTF-8" **?>**
2. <!DOCTYPE sqlMap
3. PUBLIC "-//ibatis.apache.org//DTD SQL Map 2.0//EN"
4. "http://ibatis.apache.org/dtd/sql-map-2.dtd"**>**
6. **<sqlMap** namespace="UserManager"**>**
7. **<typeAlias** alias="postreply" type="pupustory.ibatis.beans.PostReplyBean"**/>**
9. **<resultMap** class="postreply" id="postreply.result" **>**
10. **<result** property="rePostId" column="POST\_ID" **/>**
11. **<result** property="postTitle" column="POST\_TITLE" **/>**
12. **<result** property="postWriter" column="POST\_WRITER" **/>**
13. **<result** property="postBody" column="POST\_BODY" **/>**
14. **<result** property="parentPostId" column="PARENT\_POST\_ID" **/>**
15. **</resultMap>**
17. **<parameterMap** class="postreply" id="postreply.param"**>**
18. **<parameter** property="postId" **/>**
19. **<parameter** property="postTitle"**/>**
20. **<parameter** property="postWriter"**/>**
21. **<parameter** property="postBody" **/>**
22. **<parameter** property="parentPostId"**/>**
23. **</parameterMap>**
25. **<select** id="select.dynamic.query" resultMap="postreply.result"
26. parameterClass="postreply" **>**
27. SELECT \* FROM TB\_QNA\_BOARD
28. **<dynamic** prepend="WHERE"**>**
29. **<iterate** property="postId"
30. conjunction="," open="post\_id IN (" close=")" **>**
31. #postId[]#
32. **</iterate>**
33. **</dynamic>**
34. **</select>**
35. **</sqlMap>**

반복 부분을 iterator를 사용했다. 실행코드를 살펴보자.

1. **package** pupustory.ibatis.dynamicquery;
2. **import** pupustory.ibatis.beans.PostReplyBean;
3. **public** **class** StartApp {
4. **public** **static** **void** main(String ar[]) **throws** java.sql.SQLException {
5. PostReplyBean bean = **new** PostReplyBean();
7. DynamicQueryBiz biz = **new** DynamicQueryBiz();
8. String[] postIds = {"1","2","4","5","7","9"};
9. bean.setPostId(postIds);
10. java.util.List list = biz.getPostBoard(bean);
11. **if** (list.get(0) **instanceof** PostReplyBean) {
12. System.out.println("true");
13. }
14. **for** (**int** i=0;i<list.size();i++) {
15. bean = (PostReplyBean)list.get(i);
16. System.out.println(bean.toString());
17. }
18. }
19. }

어려운 구문은 없다. xml에서 open부분에 in ( 를 추가 했고, 배열마다 구분문자는 ','로 지정했다. 완료 후 ) 로 닫으니

SELECT \* FROM TB\_QNA\_BOARD WHERE POST\_ID IN ('1','2','4','5','7','9');

와 같은 sql 구문이 생성 된다. resultMap에선 실제 우리가 가져와야 할 빈의 변수와 맵핑하면 된다.

=======================

iBATIS는 객체 관계 매핑툴(ORM)이 아니고 쿼리 매핑툴(QM)이다. 정말 편리하고 유용한 매핑툴이다. 실상 간단한 어플리케이션에 활용하는데는 정말 5~10분이면 충분히 익힐수 있겠지만, 심도있게 익히려면 나름 시간과 노력이 투입되어야 할것같다.

간단히 맛보기 형태로 정리해 보았다.

iBatis 자체가 작고 매우 간단하다. iBatis는 어떠한 서버나 미들웨어도 필요로 하지 않는다. iBatis는 다른 라이브러리에 의존성을 가지지 않는다. iBatis는 최소 설치시 약 375KB의 디스크 용량을 차지하는 오직 두개의 JAR파일로만 이루어져 있다.

SQL매핑 자체에 대한 설정이없기 때문에 몇분 안에 작동하는 퍼시스턴스 계층을 확보할 수 있다. iBatis는 데이터 모델이나 객체 모델에 대해 어떠한 변경도 하지 않으면서 좀 더 유연하게 클래스와 테이블을 우회적으로 매핑하는 추가적인 계층이다.

(우회적 계층 : SQL)

iBatis를 사용함으로서 객체 모델과 데이터 베이스간 의존성을 줄였다.

-아래는 id, firstName, lastName, emailAddress로 이루어진 Account 테이블에 대한 SQL매퍼이다.

<sqlMap namespace="Account">

 <!-- Use type aliases to avoid typing the full classname every time. -->

 <typeAlias alias="Account" type="com.mydomain.domain.Account"/>

 <!-- Result maps describe the mapping between the columns returned

      from a query, and the class properties.  A result map isn't

      necessary if the columns (or aliases) match to the properties

      exactly. -->

 <resultMap id="AccountResult" class="Account">

   <result property="id" column="ACC\_ID"/>

   <result property="firstName" column="ACC\_FIRST\_NAME"/>

   <result property="lastName" column="ACC\_LAST\_NAME"/>

   <result property="emailAddress" column="ACC\_EMAIL"/>

 </resultMap>

 <!-- Select with no parameters using the result map for Account class. -->

 <select id="selectAllAccounts" resultMap="AccountResult">

   select \* from ACCOUNT

 </select>

 <!-- A simpler select example without the result map.  Note the

      aliases to match the properties of the target result class. -->

 <select id="selectAccountById" parameterClass="int" resultClass="Account">

   select

     ACC\_ID as id,

     ACC\_FIRST\_NAME as firstName,

     ACC\_LAST\_NAME as lastName,

     ACC\_EMAIL as emailAddress

   from ACCOUNT

   where ACC\_ID = #id#

 </select>

 <!-- Insert example, using the Account parameter class -->

 <insert id="insertAccount" parameterClass="Account">

   insert into ACCOUNT (

     ACC\_ID,

     ACC\_FIRST\_NAME,

     ACC\_LAST\_NAME,

     ACC\_EMAIL

   values (

     #id#, #firstName#, #lastName#, #emailAddress#

   )

 </insert>

 <!-- Update example, using the Account parameter class -->

 <update id="updateAccount" parameterClass="Account">

   update ACCOUNT set

     ACC\_FIRST\_NAME = #firstName#,

     ACC\_LAST\_NAME = #lastName#,

     ACC\_EMAIL = #emailAddress#

   where

     ACC\_ID = #id#

 </update>

 <!-- Delete example, using an integer as the parameter class -->

 <delete id="deleteAccountById" parameterClass="int">

   delete from ACCOUNT where ACC\_ID = #id#

 </delete>

</sqlMap>

위를 실행하려면, 아래와 같은 구문으로서 실행이 가능하다. 이러한 API는 iBatis API에 . .

**-Query이용한 구문 실행**

Reader reader = Resources.*getResourceAsReader*("com/mydomain/data/SqlMapConfig.xml");

SqlMapClient *sqlMapper* = SqlMapClientBuilder.*buildSqlMapClient*(reader);

*sqlMapper*.queryForList("selectAllAccounts");

(Account) *sqlMapper*.queryForObject("selectAccountById", id);

*sqlMapper*.insert("insertAccount", account);

*sqlMapper*.update("updateAccount", account);

(queryForObject()는 쿼리가 하나 이상의 레코드를 반환한다면, 예외를 던진다. 이 메소드는 오직 하 레코드만 반환되는지 여부를 검사하기 때문이다 한개 이상의 레코드는 queryForList())

queryForMap()

**-Non-Query를 이용한 구문 실행**

insert()

update()

delete()

**-구문 일괄 수행**

sqlMapLCient.startBatch()

구문~!

sqlMapClient.executeBatch()

**-트랜잭션 수행**

sqlMapClient.startTransaction()

구문~!

sqlMapClient.commitTransaction()