

```

1 iBatis
2
3 1. Homepage http://ibatis.apache.org/
4
5 2. History
6 1) 반복적이고 지루한 JDBC 프로그래밍을 단순화하기 위해 Clinton Begin 이 만든 작은 라이브러리에서 출발
7 2) 2004년 ASF 에 기부
8 3) Apache iBatis retired at the ASF(2010-06-16)
9 a. iBatis project team has moved to mybatis hosted at google code.
10 - http://www.mybatis.org
11 b. Apache iBATIS moved into the Attic in June 2010.
12 4) 지금은 Google Code 를 떠나 Github(github.com/mybatis)로 이사했음.
13
14 3. What is iBatis?
15 1) iBATIS was a data mapper framework that made it easier to use a relational database with
16 object-oriented applications.
17 2) iBATIS is a persistence framework which automates the mapping between SQL databases and
18 objects in Java, .NET, and Ruby on Rails.
19 3) iBATIS makes it easier to build better database oriented application more quickly and with less
20 code.
21 4) There were both Java and .Net implementations.
22 5) The mappings are decoupled from the application logic by packaging the SQL statements in
23 XML configuration files.
24 6) iBATIS is a lightweight framework and persistence API good for persisting POJOs( Plain Old Java
25 Objects).
26 7) iBATIS is what is known as a data mapper and takes care of mapping the parameters and
27 results between the class properties and the columns of the database table.
28 8) A significant difference between iBATIS and other persistence frameworks such as Hibernate is
29 that iBATIS emphasizes use of SQL, while other frameworks typically use a custom query language
30 such has the Hibernate Query Language (HQL) or Enterprise JavaBeans Query Language (EJB QL).
31 5) Archived versions of iBATIS may be downloaded from the Apache Archives(
32 http://archive.apache.org/dist/ibatis/).
33 6) 개발과 유지보수가 쉽도록 소스 코드에 박혀있는 SQL을 별도의 파일로 분리하는 것이 핵심
34 7) DAO.java속에 있는 SQL 문을 자바 코드로 부터 분리하여 xml 파일로 생성함.
35
36 4. Download & Configuration
37 1) Visit https://archive.apache.org/dist/ibatis/binaries/ibatis.java/
38 2) 목록에서 ibatis-2.3.4.726.zip click
39 3) Downloads and unzip downloaded file
40 4) If Web Application develope, copy ibatis-2.3.4.726.jar to /WEB-INF/lib
41 5) Javadoc
42 - http://ibatis.apache.org/docs/java/user/
43
44 5. Create src\SqlMapConfig.xml
45 1) Consider the following:
46 - We are going to use JDBC to access the database test.
47 - JDBC driver for Oracle XE is "oracle.jdbc.driver.OracleDriver".
48 - Connection URL is "jdbc:oracle:thin:@localhost:1521:XE".
49 - We would use username and password is "scott" and "tiger".
50 - Our sql statement mappings for all the operations would be described in "Employee.xml".
51 2) Based on the above assumption we have to create an XML configuration file with name
52 src\SqlMapConfig.xml with the following content.
53 3) Copy ibatis-2.3.4.726/simple_example/com/mydomain/data/SqlMapConfig.xml to
54 src\SqlMapConfig.xml. And Rewrite like below.
55
56 <?xml version="1.0" encoding="UTF-8"?>
57 <!DOCTYPE sqlMapConfig
58 PUBLIC "-//ibatis.apache.org//DTD SQL Map Config 2.0//EN"
59 "http://ibatis.apache.org/dtd/sql-map-config-2.dtd">
60 <sqlMapConfig>
61 <properties resource="dbinfo.properties" />
62 <settings useStatementNamespaces="true"/>
63 <transactionManager type="JDBC">
64 <dataSource type="SIMPLE">
65 <property name="JDBC.Driver" value="${db.driverClass}"/>
66 <property name="JDBC.ConnectionURL" value="${db.url}"/>
67 <property name="JDBC.Username" value="${db.username}"/>

```

```

57         <property name="JDBC.Password" value="\${db.password}"/>
58
59         <!--<property name="JDBC.Driver" value="oracle.jdbc.driver.OracleDriver"/>
60         <property name="JDBC.ConnectionURL" value="jdbc:oracle:thin:@localhost:1521:XE"/>
61         <property name="JDBC.Username" value="scott"/>
62         <property name="JDBC.Password" value="tiger"/>-->
63     </dataSource>
64 </transactionManager>
65 <sqlMap resource="com/javasoft/libs/model/Employee.xml"/>
66 </sqlMapConfig>
67 4) There are other optional properties which you can set using SqlMapConfig.xml file:
68 - <property name="JDBC.AutoCommit" value="true"/>
69 - <property name="Pool.MaximumActiveConnections" value="10"/>
70 - <property name="Pool.MaximumIdleConnections" value="5"/>
71 - <property name="Pool.MaximumCheckoutTime" value="150000"/>
72 - <property name="Pool.MaximumTimeToWait" value="500"/>
73 - <property name="Pool.PingQuery" value="select 1 from Employee"/>
74 - <property name="Pool.PingEnabled" value="false"/>
75
76 6. iBatis Create Operation
77 1) To perform any CRUD(Create, Read, Write, Update and Delete) operation using iBATIS, you
78    would need to create a POJOs(Plain Old Java Objects) class corresponding to the table.
79 2) This class describes the objects that will "model" database table rows.
80 3) The POJO class would have implementation for all the methods required to perform desired
81    operations.
82 4) We have following EMPLOYEE table in Oracle XE:
83     CREATE TABLE Employee (
84         id NUMBER(4),
85         first_name VARCHAR2(20),
86         last_name VARCHAR2(20),
87         salary NUMBER(8),
88         CONSTRAINTS employee_id_pk PRIMARY KEY (id)
89     );
90 5) Employee POJO Class:
91     package com.javasoft.libs.model;
92     public class EmployeeDTO {
93         private int id;
94         private String first_name;
95         private String last_name;
96         private int salary;
97         /* Define constructors for the Employee class. */
98         public EmployeeDTO() {}
99         public EmployeeDTO(String fname, String lname, int salary) {
100             this.first_name = fname;
101             this.last_name = lname;
102             this.salary = salary;
103         }
104         public int getId() {
105             return id;
106         }
107         public void setId(int id) {
108             this.id = id;
109         }
110         public String getFirst_name() {
111             return first_name;
112         }
113         public void setFirst_name(String first_name) {
114             this.first_name = first_name;
115         }
116         public String getLast_name() {
117             return last_name;
118         }
119         public void setLast_name(String last_name) {
120             this.last_name = last_name;
121         }
122         public int getSalary() {
123             return salary;

```

```

122     }
123     public void setSalary(int salary) {
124         this.salary = salary;
125     }
126 }
127

```

#### 6) com.javasoft.libs.model.Employee.xml File:

- To define SQL mapping statement using iBATIS, we would use <insert> tag and inside this tag definition we would define an "id" which will be used in IbatisInsert.java file for executing SQL INSERT query on database.

- Copy ibatis-2.3.4.726/simple\_example/com/mydomain/data/Account.xml to com.javasoft.libs.model.Account.xml

- Rename Account.xml to Employee.xml

- Rewrite like below

```

134 <?xml version="1.0" encoding="UTF-8"?>
135
136 <!DOCTYPE sqlMap
137     PUBLIC "-//ibatis.apache.org//DTD SQL Map 2.0//EN"
138     "http://ibatis.apache.org/dtd/sql-map-2.dtd">
139
140 <sqlMap namespace="Employee">
141     <typeAlias alias="employeeDTO" type="com.javasoft.libs.model.EmployeeDTO"/>
142     <insert id="insert" parameterClass="employeeDTO">
143         <selectKey keyProperty="id" resultClass="int">
144             SELECT seq_employee_id.NEXTVAL AS ID FROM DUAL
145         </selectKey>
146         INSERT INTO EMPLOYEE(id, first_name, last_name, salary)
147         VALUES (#id#, #first_name#, #last_name#, #salary#)
148     </insert>
149 </sqlMap>
150

```

#### 7) src\IbatisInsert.java File:

- This file would have application level logic to insert records in the Employee table:

```

153 import com.ibatis.common.resources.Resources;
154 import com.ibatis.sqlmap.client.SqlMapClient;
155 import com.ibatis.sqlmap.client.SqlMapClientBuilder;
156 import com.javasoft.libs.model.EmployeeDTO;
157 import java.io.*;
158 import java.sql.SQLException;
159 import java.util.*;
160 public class IbatisInsert{
161     public static void main(String[] args) throws IOException,SQLException{
162         Reader rd = Resources.getResourceAsReader("SqlMapConfig.xml");
163         SqlMapClient smc = SqlMapClientBuilder.buildSqlMapClient(rd);
164         /* This would insert one record in Employee table. */
165         System.out.println("Going to insert record.....");
166         EmployeeDTO em = new EmployeeDTO("Zara", "Ali", 5000);
167         smc.insert("Employee.insert", em);
168         System.out.println("Record Inserted Successfully ");
169     }
170 }
171

```

- insert.html

```

173 <!DOCTYPE html>
174 <html>
175     <head>
176         <meta charset="UTF-8">
177         <title>Insert Form</title>
178     </head>
179     <body>
180         <h1>Join membership</h1>
181         <form method="post" action="insert.jsp">
182             <p><label for="first_name">First Name : </label>
183             <input type="text" id="first_name" name="first_name" size="20"/></p>
184             <p><label for="last_name">Last Name : </label>
185             <input type="text" id="last_name" name="last_name" size="20"/></p>

```

```

186         <p><label for="salary">Salary : </label>
187         <input type="number" id="salary" name="salary" size="10" /></p>
188         <input type="submit" value="join" />
189     </form>
190 </body>
191 </html>
192 - insert.jsp
193 <%@ page language="java" contentType="text/html; charset=utf-8"
    pageEncoding="utf-8"%>
194 <%@ page import="com.ibatis.common.resources.Resources" %>
195 <%@ page import="com.ibatis.sqlmap.client.SqlMapClient" %>
196 <%@ page import="com.ibatis.sqlmap.client.SqlMapClientBuilder" %>
197 <%@ page import="java.io.Reader" %>
198 <%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
199 <fmt:requestEncoding value="utf-8" />
200 <jsp:useBean id="emp" class="com.javasoft.libs.model.EmployeeDTO" />
201 <jsp:setProperty name="emp" property="*" />
202 <%
203     Reader rd = Resources.getResourceAsReader("SqlMapConfig.xml");
204     SqlMapClient smc = SqlMapClientBuilder.buildSqlMapClient(rd);
205
206     smc.insert("Employee.insert", emp);
207     out.println("Success");
208 %>
209
210 7. iBatis Read Operation
211 1) com.javasoft.libs.model.Employee.xml File:
212 - To define SQL mapping statement using iBATIS, we would add <select> tag in Employee.xml
    file and inside this tag definition we would define an "id" which will be used in IbatisRead.java
    file for executing SQL SELECT query on database.
213
214 2)
215 <select id="getAll" resultClass="employeeDTO">
216     SELECT * FROM EMPLOYEE
217     ORDER BY id DESC
218 </select>
219 3) src\IbatisRead.java File:
220 - This file would have application level logic to read records from the Employee table:
221 import com.ibatis.common.resources.Resources;
222 import com.ibatis.sqlmap.client.SqlMapClient;
223 import com.ibatis.sqlmap.client.SqlMapClientBuilder;
224 import com.javasoft.libs.model.EmployeeDTO;
225 import java.io.*;
226 import java.sql.SQLException;
227 import java.util.*;
228 public class IbatisRead{
229     public static void main(String[] args) throws IOException,SQLException{
230         Reader rd = Resources.getResourceAsReader("SqlMapConfig.xml");
231         SqlMapClient smc = SqlMapClientBuilder.buildSqlMapClient(rd);
232         /* This would read all records from the Employee table. */
233         System.out.println("Going to read records.....");
234         List <EmployeeDTO> ems = (List<EmployeeDTO>)smc.queryForList("Employee.getAll",
            null);
235
236         for (EmployeeDTO e : ems) {
237             System.out.print(" " + e.getId());
238             System.out.print(" " + e.getFirst_name());
239             System.out.print(" " + e.getLast_name());
240             System.out.print(" " + e.getSalary());
241             System.out.println("");
242         }
243         System.out.println("Records Read Successfully ");
244     }
245 }
246 4) select.jsp
247 <%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
248 <%@ page import="com.ibatis.common.resources.Resources" %>
    <%@ page import="com.ibatis.sqlmap.client.SqlMapClient" %>

```

```

249 <%@ page import="com.ibatis.sqlmap.client.SqlMapClientBuilder" %>
250 <%@ page import="java.io.Reader" %>
251 <%@ page import="java.util.List" %>
252 <%@ page import="com.javasoft.libs.model.EmployeeDTO" %>
253 <%
254     Reader rd = Resources.getResourceAsReader("SqlMapConfig.xml");
255     SqlMapClient smc = SqlMapClientBuilder.buildSqlMapClient(rd);
256
257     List<EmployeeDTO> list = (List<EmployeeDTO>)smc.queryForList("Employee.getAll", null);
258
259     out.println("<h2>Account List</h2>");
260     out.println("<table border='1' style='table-layout:fixed;height:auto;margin:auto'>");
261     out.println("<thead>");
262     out.println("<tr>");
263     out.println("<th>ID</th> <th>First Name</th> <th>Last Name</th> <th>Salary</th>");
264     out.println("</tr>");
265     out.println("</thead>");
266     out.println("<tbody>");
267     for (EmployeeDTO e : list) {
268         out.println("<tr>");
269         out.print("<td>" + e.getId() + "</td>");
270         out.print("<td>" + e.getFirst_name() + "</td>");
271         out.print("<td>" + e.getLast_name() + "</td>");
272         out.print("<td>" + String.format("%,d", e.getSalary()) + "</td>");
273         out.println("</tr>");
274     }
275     out.println("</tbody>");
276     out.println("</table>");
277 %>

```

## 8. iBatis Update Operation

### 1) Employee.xml File:

- To define SQL mapping statement using iBATIS, we would add <update> tag in Employee.xml file and inside this tag definition we would define an "id" which will be used in IbatisUpdate.java file for executing SQL UPDATE query on database.

```

282 <update id="update" parameterClass="employeeDTO">
283     UPDATE EMPLOYEE
284     SET    first_name = #first_name#
285     WHERE id = #id#
286 </update>

```

### 2) IbatisUpdate.java File:

- This file would have application level logic to update records into the Employee table:

```

289 import com.ibatis.common.resources.Resources;
290 import com.ibatis.sqlmap.client.SqlMapClient;
291 import com.ibatis.sqlmap.client.SqlMapClientBuilder;
292 import com.javasoft.libs.model.EmployeeDTO;
293 import java.io.*;
294 import java.sql.SQLException;
295 import java.util.*;
296 public class IbatisUpdate{
297     public static void main(String[] args) throws IOException,SQLException{
298         Reader rd = Resources.getResourceAsReader("SqlMapConfig.xml");
299         SqlMapClient smc = SqlMapClientBuilder.buildSqlMapClient(rd);
300
301         System.out.println("Going to update record.....");
302         EmployeeDTO emp = new EmployeeDTO();
303         emp.setId(1);
304         emp.setFirst_name( "Roma");
305         smc.update("Employee.update", emp);
306         System.out.println("Record updated Successfully ");
307         System.out.println("Going to read records.....");
308         List <EmployeeDTO> list = (List<EmployeeDTO>)smc.queryForList("Employee.getAll", null);
309
310         for (EmployeeDTO e : list) {
311             System.out.print(" " + e.getId());
312             System.out.print(" " + e.getFirst_name());
313             System.out.print(" " + e.getLast_name());

```

```

314         System.out.print(" " + e.getSalary());
315         System.out.println("");
316     }
317     System.out.println("Records Read Successfully ");
318 }
319 }
320 3)update.jsp
321 <%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
322 <%@ page import="com.ibatis.common.resources.Resources" %>
323 <%@ page import="com.ibatis.sqlmap.client.SqlMapClient" %>
324 <%@ page import="com.ibatis.sqlmap.client.SqlMapClientBuilder" %>
325 <%@ page import="java.io.Reader" %>
326 <%@ page import="java.util.List" %>
327 <%@ page import="com.javasoft.libs.model.EmployeeDTO" %>
328 <%
329     Reader rd = Resources.getResourceAsReader("SqlMapConfig.xml");
330     SqlMapClient smc = SqlMapClientBuilder.buildSqlMapClient(rd);
331
332     EmployeeDTO emp = new EmployeeDTO();
333     emp.setId(3);
334     emp.setFirst_name("Roma");
335     smc.update("Employee.update", emp);
336
337     List <EmployeeDTO> list = (List<EmployeeDTO>)smc.queryForList("Employee.getAll", null);
338
339     out.println("<h2>Account List</h2>");
340     out.println("<table border='1' style='table-layout:fixed;height:auto;margin:auto'>");
341     out.println("<thead>");
342     out.println("<tr>");
343     out.println("<th>ID</th><th>First Name</th><th>Last Name</th><th>Salary</th>");
344     out.println("</tr>");
345     out.println("</thead>");
346     out.println("<tbody>");
347     for (EmployeeDTO e : list) {
348         out.println("<tr>");
349         out.print("<td>" + e.getId() + "</td>");
350         out.print("<td>" + e.getFirst_name() + "</td>");
351         out.print("<td>" + e.getLast_name() + "</td>");
352         out.print("<td>" + String.format("%,d", e.getSalary()) + "</td>");
353         out.println("</tr>");
354     }
355     out.println("</tbody>");
356     out.println("</table>");
357 %>
358

```

## 9. iBatis Delete Operation

### 1) Employee.xml File:

- To define SQL mapping statement using iBATIS, we would add <delete> tag in Employee.xml file and inside this tag definition we would define an "id" which will be used in IbatisDelete.java file for executing SQL DELETE query on database.

```

362 <delete id="delete" parameterClass="int">
363     DELETE FROM EMPLOYEE WHERE id = #id#
364 </delete>

```

### 2) src\IbatisDelete.java File:

- This file would have application level logic to delete records from the Employee table:

```

367 import com.ibatis.common.resources.Resources;
368 import com.ibatis.sqlmap.client.SqlMapClient;
369 import com.ibatis.sqlmap.client.SqlMapClientBuilder;
370 import com.javasoft.libs.model.EmployeeDTO;
371 import java.io.IOException;
372 import java.io.Reader;
373 import java.sql.SQLException;
374 import java.util.List;
375
376 public class IbatisDelete{
377     public static void main(String[] args) throws IOException,SQLException{
378         Reader rd = Resources.getResourceAsReader("SqlMapConfig.xml");

```

```

379         SqlMapClient smc = SqlMapClientBuilder.buildSqlMapClient(rd);
380
381         System.out.println("Going to delete record.....");
382         int id = 1;
383         smc.delete("Employee.delete", id );
384         System.out.println("Record deleted Successfully ");
385         System.out.println("Going to read records.....");
386         List <EmployeeDTO> ems = (List<EmployeeDTO>)smc.queryForList("Employee.getAll",
            null);
387
388         for (EmployeeDTO e : ems) {
389             System.out.print(" " + e.getId());
390             System.out.print(" " + e.getFirst_name());
391             System.out.print(" " + e.getLast_name());
392             System.out.print(" " + e.getSalary());
393             System.out.println("");
394         }
395         System.out.println("Records Read Successfully ");
396     }
397 }
398 3)delete.jsp
399 <%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
400 <%@ page import="com.ibatis.common.resources.Resources" %>
401 <%@ page import="com.ibatis.sqlmap.client.SqlMapClient" %>
402 <%@ page import="com.ibatis.sqlmap.client.SqlMapClientBuilder" %>
403 <%@ page import="java.io.Reader" %>
404 <%@ page import="java.util.List" %>
405 <%@ page import="com.javasoft.libs.model.EmployeeDTO" %>
406
407 <%
408     Reader rd = Resources.getResourceAsReader("SqlMapConfig.xml");
409     SqlMapClient smc = SqlMapClientBuilder.buildSqlMapClient(rd);
410
411     int id = 3;
412     smc.delete("Employee.delete", id );
413
414     List <EmployeeDTO> list = (List<EmployeeDTO>)smc.queryForList("Employee.getAll", null);
415
416     out.println("<h2>Account List</h2>");
417     out.println("<table border='1' style='table-layout:fixed;height:auto;margin:auto'>");
418     out.println("<thead>");
419     out.println("<tr>");
420     out.println("<th>ID</th><th>First Name</th><th>Last Name</th><th>Salary</th>");
421     out.println("</tr>");
422     out.println("</thead>");
423     out.println("<tbody>");
424     for (EmployeeDTO e : list) {
425         out.println("<tr>");
426         out.print("<td>" + e.getId() + "</td>");
427         out.print("<td>" + e.getFirst_name() + "</td>");
428         out.print("<td>" + e.getLast_name() + "</td>");
429         out.print("<td>" + String.format("%,d", e.getSalary()) + "</td>");
430         out.println("</tr>");
431     }
432     out.println("</tbody>");
433     out.println("</table>");
434 %>

```

## 10. iBatis Result Maps

- 1) The resultMap element is the most important and powerful element in iBATIS.
- 2) You can reduce upto 90% JDBC coding by using iBATIS ResultMap and in some cases allows you to do things that JDBC does not even support.
- 3) The design of the ResultMaps is such that simple statements don't require explicit result mappings at all, and more complex statements require no more than is absolutely necessary to describe the relationships.
- 4) Employee.xml File:
  - Here we would modify Employee.xml file to introduce <resultMap></resultMap> tag.

- This tag would have an id which is required to run this resultMap in our <select> tag's resultMap attribute.

```
<!-- Using resultMap -->
<resultMap id="result" class="EmployeeDTO">
    <result property="id" column="id"/>
    <result property="first_name" column="first_name"/>
    <result property="last_name" column="last_name"/>
    <result property="salary" column="salary"/>
</resultMap>
<select id="useResultMap" resultMap="result">
    SELECT * FROM Employee WHERE id=#id#
</select>
```

#### 5) src\IbatisResultMap.java File:

- This file would have application level logic to read records from the Employee table using resultMap:

```
import com.ibatis.common.resources.Resources;
import com.ibatis.sqlmap.client.SqlMapClient;
import com.ibatis.sqlmap.client.SqlMapClientBuilder;
import com.javasoft.libs.model.EmployeeDTO;
import java.io.Reader;
import java.io.IOException;
import java.sql.SQLException;

public class IbatisResultMap{
    public static void main(String[] args) throws IOException,SQLException{
        Reader rd = Resources.getResourceAsReader("SqlMapConfig.xml");
        SqlMapClient smc = SqlMapClientBuilder.buildSqlMapClient(rd);
        int id = 2;
        System.out.println("Going to read record.....");
        EmployeeDTO e = (EmployeeDTO)smc.queryForObject("Employee.useResultMap", id);
        System.out.println("ID: " + e.getId());
        System.out.println("First Name: " + e.getFirst_name());
        System.out.println("Last Name: " + e.getLast_name());
        System.out.println("Salary: " + e.getSalary());
        System.out.println("Record read Successfully ");
    }
}
```

#### 11. iBatis Stored Procedures

1) This is very much possible to call a stored procedure using iBATIS configuration.

2)

```
CREATE OR REPLACE PROCEDURE sp_employee_insert
(
    v_fname    IN    Employee.first_name%TYPE,
    v_lname    IN    Employee.last_name %TYPE,
    v_salary    IN    Employee.salary%TYPE
)
IS
BEGIN
    INSERT INTO Employee(id, first_name, last_name, salary)
    VALUES (SEQ_EMPLOYEE_ID.NEXTVAL, v_fname, v_lname, v_salary);

    COMMIT;
END;
/
```

#### 3) Employee.xml File:

- Here we would modify Employee.xml file to introduce <procedure></procedure> and <parameterMap></parameterMap> tags.

- Here <procedure></procedure> tag would have an id which we would use in our application to call the stored procedure.

```
<parameterMap id="setEmpInfoCall" class="map">
    <parameter property="v_fname" jdbcType="VARCHAR" javaType="java.lang.String"
    mode="IN"/>
    <parameter property="v_lname" jdbcType="VARCHAR" javaType="java.lang.String"
    mode="IN"/>
```



```

503         <parameter property="v_salary" jdbcType="INT" javaType="java.lang.Integer" mode="IN"/>
504     </parameterMap>
505     <procedure id="setEmpInfo" parameterMap="setEmpInfoCall">
506         { call sp_employee_insert(?,?,?) }
507     </procedure>
508 4)IbatisSP.java File:
509     - This file would have application level logic to read name of the employee from the Employee
      table using ResultMap:
510     import com.ibatis.common.resources.Resources;
511     import com.ibatis.sqlmap.client.SqlMapClient;
512     import com.ibatis.sqlmap.client.SqlMapClientBuilder;
513     import java.io.*;
514     import java.sql.SQLException;
515     import java.util.*;
516     public class IbatisSP{
517         public static void main(String[] args) throws IOException,SQLException{
518             Reader rd = Resources.getResourceAsReader("SqlMapConfig.xml");
519             SqlMapClient smc = SqlMapClientBuilder.buildSqlMapClient(rd);
520             int id = 1;
521             System.out.println("Going to read employee name.....");
522             Map<String, Object> map = new HashMap<String,Object>();
523             map.put("v_fname", "Peter");
524             map.put("v_lname", "Bok");
525             map.put("v_salary", new Integer(3000));
526             smc.insert("Employee.setEmpInfo", map);
527             System.out.println("Record Insert Successfully ");
528         }
529     }
530 5)sp.jsp
531     <%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
532     <%@ page import="com.ibatis.common.resources.Resources" %>
533     <%@ page import="com.ibatis.sqlmap.client.SqlMapClient" %>
534     <%@ page import="com.ibatis.sqlmap.client.SqlMapClientBuilder" %>
535     <%@ page import="java.io.Reader" %>
536     <%@ page import="java.util.List, java.util.Map, java.util.HashMap" %>
537     <%@ page import="com.javasoft.libs.model.EmployeeDTO" %>
538
539     <%
540         Reader rd = Resources.getResourceAsReader("SqlMapConfig.xml");
541         SqlMapClient smc = SqlMapClientBuilder.buildSqlMapClient(rd);
542         System.out.println("Going to read employee name.....");
543         Map<String, Object> map = new HashMap<String,Object>();
544         map.put("v_fname", "Peter");
545         map.put("v_lname", "Bok");
546         map.put("v_salary", new Integer(3000));
547         smc.insert("Employee.setEmpInfo", map);
548         System.out.println("Record name Successfully ");
549     %>
550
551 12. Oracle Cursor with iBatis
552 1)storeprocedure sp_employee_select
553     CREATE OR REPLACE PROCEDURE sp_employee_select
554     (
555         employee_cursor OUT SYS_REFCURSOR
556     )
557     AS
558     BEGIN
559         OPEN employee_cursor FOR
560         SELECT * FROM employee;
561     END;
562
563 2)Employee.xml
564     <resultMap class="com.javasoft.libs.model.EmployeeDTO" id="selectResultMap">
565         <result property="id" column="id" />
566         <result property="first_name" column="first_name" />
567         <result property="last_name" column="last_name" />
568         <result property="salary" column="salary" />

```

```

569     </resultMap>
570
571     <parameterMap id="selectAllMap" class="java.util.Map">
572         <parameter property="result" javaType="java.sql.ResultSet" jdbcType="ORACLECURSOR"
            mode="OUT"/>
573     </parameterMap>
574
575     <procedure id="selectAll" parameterMap="selectAllMap" resultMap="selectResultMap">
576         { call sp_employee_select(?) }
577     </procedure>
578 3)src\iBatisCursor.java
579     import java.io.IOException;
580     import java.io.Reader;
581     import java.sql.SQLException;
582     import java.util.HashMap;
583     import java.util.List;
584     import java.util.Map;
585
586     import com.ibatis.common.resources.Resources;
587     import com.ibatis.sqlmap.client.SqlMapClient;
588     import com.ibatis.sqlmap.client.SqlMapClientBuilder;
589     import com.javasoft.libs.model.EmployeeDTO;
590
591     public class IbatisReadCursor{
592         @SuppressWarnings("unchecked")
593         public static void main(String[] args)
594             throws IOException,SQLException{
595             Reader rd = Resources.getResourceAsReader("SqlMapConfig.xml");
596             SqlMapClient smc = SqlMapClientBuilder.buildSqlMapClient(rd);
597
598             Map map = new HashMap();
599             List<EmployeeDTO> list = smc.queryForList("Employee.selectAll", map);
600             for(int i = 0 ; i < list.size(); i++){
601                 EmployeeDTO employee = list.get(i);
602                 System.out.println("<ul>");
603                 System.out.println("<li>ID : " + employee.getId() + "</li>");
604                 System.out.println("<li>First Name : " + employee.getFirst_name() + "</li>");
605                 System.out.println("<li>Last Name : " + employee.getLast_name() + "</li>");
606                 System.out.println("<li>Salary : " + employee.getSalary() + "</li>");
607                 System.out.println("</ul>");
608                 System.out.println("-----");
609             }
610         }
611     }
612 4)cursor.jsp
613     <%@ page language="java" contentType="text/html; charset=UTF-8"
614         pageEncoding="UTF-8"%>
615     <%@ page import="com.ibatis.common.resources.Resources" %>
616     <%@ page import="com.ibatis.sqlmap.client.SqlMapClient" %>
617     <%@ page import="com.ibatis.sqlmap.client.SqlMapClientBuilder" %>
618     <%@ page import="java.io.Reader" %>
619     <%@ page import="java.util.List, java.util.Map, java.util.HashMap" %>
620     <%@ page import="com.javasoft.libs.model.EmployeeDTO" %>
621     <!DOCTYPE html>
622     <html>
623     <head>
624     <meta charset="UTF-8">
625     <title>Insert title here</title>
626     </head>
627     <body>
628     <%
629         Reader rd = Resources.getResourceAsReader("SqlMapConfig.xml");
630         SqlMapClient smc = SqlMapClientBuilder.buildSqlMapClient(rd);
631         Map map = new HashMap();
632         List<EmployeeDTO> list = smc.queryForList("Employee.selectAll", map);
633         for(int i = 0 ; i < list.size(); i++){
634             EmployeeDTO employee = list.get(i);

```

```

635         out.println("<ul>");
636         out.println("<li>ID : " + employee.getId() + "</li>");
637         out.println("<li>First Name : " + employee.getFirst_name() + "</li>");
638         out.println("<li>Last Name : " + employee.getLast_name() + "</li>");
639         out.println("<li>Salary : " + employee.getSalary() + "</li>");
640         out.println("</ul>");
641     }
642     %>
643 </body>
644 </html>
645

```

### 646 13. iBatis Dynamic SQL

```

647 1) Using dynamic queries is a very powerful feature of iBatis. Sometime you have changing WHERE
    clause criterion based on your parameter object's state.
648 2) In such situation iBATIS provides a set of dynamic SQL tags that can be used within apped
    statements to enhance the reusability and flexibility of the SQL.
649 3) All the logic is put in .XML file using some additional tags.
650 4) Following is an example where SELECT statement would work in two ways:
651     - If you would pass an ID then it would return all the records corresponding to that ID
652     - otherwise it would return all the records where employee ID is set to NULL.
653 5) Employee.xml File:
654     - To define SQL mapping statement using iBATIS, we would add following modified <select> tag
    in Employee.xml file and inside this tag definition we would define an "id" which will be used in
    IbatisReadDy.java file for executing Dynamic SQL SELECT query on database.
655     <select id="findByID" resultClass="employeeDTO">
656         SELECT * FROM Employee
657         <dynamic prepend="WHERE ">
658             <isNotNull property="id">
659                 id = #id#
660             </isNotNull>
661         </dynamic>
662     </select>
663 6) Above SELECT statement would work in two ways (i) If you would pass an ID then it would
    return records corresponding to that ID (ii) otherwise it would return all the records.
664 7) src\IbatisReadDy.java File:
665     - This file would have application level logic to read conditional records from the Employee table:
666     import com.ibatis.common.resources.Resources;
667     import com.ibatis.sqlmap.client.SqlMapClient;
668     import com.ibatis.sqlmap.client.SqlMapClientBuilder;
669     import com.javasoft.libs.model.EmployeeDTO;
670     import java.io.*;
671     import java.sql.SQLException;
672     import java.util.*;
673     public class IbatisReadDy{
674         public static void main(String[] args) throws IOException,SQLException{
675             Reader rd=Resources.getResourceAsReader("SqlMapConfig.xml");
676             SqlMapClient smc=SqlMapClientBuilder.buildSqlMapClient(rd);
677             /* This would read all records from the Employee table.*/
678             System.out.println("Going to read records.....");
679             EmployeeDTC rec = new EmployeeDTO();
680             rec.setId(1);
681             List <EmployeeDTO> ems = (List<EmployeeDTO>)smc.queryForList("Employee.findByID",
                rec);
682
683             for (EmployeeDTO e : ems) {
684                 System.out.print(" " + e.getId());
685                 System.out.print(" " + e.getFirstName());
686                 System.out.print(" " + e.getLastName());
687                 System.out.print(" " + e.getSalary());
688                 em = e;
689                 System.out.println("");
690             }
691             System.out.println("Records Read Successfully ");
692         }
693     }
694 8)dynamic.jsp
695     <%@ page language="java" contentType="text/html; charset=UTF-8"

```

```

696     pageEncoding="UTF-8"%>
697 <%@ page import="com.ibatis.common.resources.Resources" %>
698 <%@ page import="com.ibatis.sqlmap.client.SqlMapClient" %>
699 <%@ page import="com.ibatis.sqlmap.client.SqlMapClientBuilder" %>
700 <%@ page import="java.io.Reader" %>
701 <%@ page import="java.util.List" %>
702 <%@ page import="com.javasoft.libs.model.EmployeeDTO" %>
703
704 <%
705     Reader rd=Resources.getResourceAsReader("SqlMapConfig.xml");
706     SqlMapClient smc=SqlMapClientBuilder.buildSqlMapClient(rd);
707
708     System.out.println("Going to read records.....");
709     EmployeeDTO emp = new EmployeeDTO();
710     emp.setId(22);
711     List <EmployeeDTO> list = (List<EmployeeDTO>)smc.queryForList("Employee.findById",
712     emp);
713
714     for (EmployeeDTO e : list) {
715         System.out.print(" " + e.getId());
716         System.out.print(" " + e.getFirst_name());
717         System.out.print(" " + e.getLast_name());
718         System.out.print(" " + e.getSalary());
719     }
720     System.out.println("Records Read Successfully ");
721 %>

```

#### 14. iBatis OGNL Expressions

1) iBATIS provides powerful OGNL based expressions to eliminate most of the other elements.

- if Statement
- choose, when, otherwise Statement
- where Statement
- foreach Statement

2) The if Statement:

- The most common thing to do in dynamic SQL is conditionally include a part of a where clause.

```

730 <select id="findActiveBlogWithTitleLike" parameterType="Blog" resultType="Blog">
731     SELECT * FROM BLOG WHERE state = 'ACTIVE.
732     <if test="title != null">
733         AND title like #{title}
734     </if>
735 </select>

```

- This statement would provide an optional text search type of functionality.
- If you passed in no title, then all active Blogs would be returned.
- But if you do pass in a title, it will look for a title with the given likecondition.
- You can include multiple if conditions as follows:
- The most common thing to do in dynamic SQL is conditionally include a part of a where clause.

```

741 <select id="findActiveBlogWithTitleLike" parameterType="Blog" resultType="Blog">
742     SELECT * FROM BLOG WHERE state = 'ACTIVE.
743     <if test="title != null">
744         AND title like #{title}
745     </if>
746     <if test="author != null">
747         AND author like #{author}
748     </if>
749 </select>

```

3) The choose, when, otherwise Statement:

- iBATIS offers a choose element which is similar to Java's switch statement.
- This helps choose only one case among many options.
- Following example would search only on title if one is provided, then only by author if one is provided.
- If neither is provided, let's only return featured blogs:

```

755 <select id="findActiveBlogWithTitleLike" parameterType="Blog" resultType="Blog">
756     SELECT * FROM BLOG WHERE state = 'ACTIVE.
757     <choose>
758         <when test="title != null">
759             AND title like #{title}
760         </when>

```

```

761         <when test="author != null and author.name != null">
762             AND author like #{author}
763         </when>
764         <otherwise>
765             AND featured = 1
766         </otherwise>
767     </choose>
768 </select>

```

#### 4) The where Statement:

- If we look previous examples, What happens if none of the conditions are met?
- You would end up with SQL that looked like this:  
SELECT \* FROM BLOG WHERE
- This would fail, but iBATIS has a simple solution with one simple change, everything works fine:

```

774 <select id="findActiveBlogLike" parameterType="Blog" resultType="Blog">
775     SELECT * FROM BLOG
776     <where>
777         <if test="state != null">
778             state = #{state}
779         </if>
780         <if test="title != null">
781             AND title like #{title}
782         </if>
783         <if test="author != null">
784             AND author like #{author}
785         </if>
786     </where>
787 </select>

```

- The where element knows to only insert WHERE if there is any content returned by the containing tags.
- Furthermore, if that content begins with AND or OR, it knows to strip it off.

#### 5) The foreach Statement:

- The foreach element is very powerful, and allows you to specify a collection, declare item and index variables that can be used inside the body of the element.
- It also allows you to specify opening and closing strings, and add a separator to place in between iterations.
- You can build an IN condition as follows:

```

794 <select id="selectPostIn" resultType="domain.blog.Post">
795     SELECT * FROM POST P WHERE ID in
796     <foreach item="item" index="index" collection="list" open="(" separator="," close=")">
797         #{item}
798     </foreach>
799 </select>

```

800

#### 801 14. iBatis vs Hibernate

802 1) There are major differences between iBatis and Hibernate but both the solutions work well, given their specific domain.

803 2) Personally I would suggest you should use iBATIS if:

- 804 - You want to create your own SQL's and are willing to maintain them.
- 805 - Your environment is driven by relational data model.
- 806 - You have to work existing and complex schema's.

807 3) And simply use Hibernate if:

- 808 - Your environment is driven by object model and wants generates SQL automatically.

809 4) To count there are few differences:

810 a. iBATIS is:

- 811 - Simpler
- 812 - Faster development time
- 813 - Flixable
- 814 - Much smaller in package size

815 b. Hibernate:

- 816 - Generates SQL for you which means you don't spend time on SQL
- 817 - Provides much more advance cache
- 818 - Highly scalable

819 5) Other difference is that iBATIS makes use of SQL which could be database dependent where as Hibernate makes use of HQL which is relatively independent of databases and it is easier to change db in Hibernate.

820 6) Hibernate maps your Java POJO objects to the Database tables where as iBatis maps the ResultSet from JDBC API to your POJO Objects.

- 821 7) If you are using stored procedures, well you can do it in Hibernate but it is little difficult in  
comparision of iBATIS.
- 822 8) As an alternative solution iBATIS maps results sets to objects, so no need to care about table  
structures.
- 823 9) This works very well for stored procedures, works very well for reporting applications, etc .
- 824 10) Finally, Hibernate and iBATIS both are open source Object Relational Mapping(ORM) tools  
available in the industry.
- 825 11) Use of each of these tools depends on the context you are using them.
- 826 12) Hibernate and iBatis both also have good support from SPRING framework so it should not be  
a problem to chose one of them.