IBATIS - DYNAMIC SQL

Dynamic SQL is a very powerful feature of iBATIS. Sometimes you have to change the WHERE clause criterion based on your parameter object's state. In such situations, iBATIS provides a set of dynamic SQL tags that can be used within mapped statements to enhance the reusability and flexibility of the SQL.

All the logic is put in .XML file using some additional tags. Following is an example where the SELECT statement would work in two ways −

- If you pass an ID, then it would return all the records corresponding to that ID.
- Otherwise, it would return all the records where employee ID is set to NULL.

```
<?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 namespace="Employee">

   <select >
      SELECT * FROM EMPLOYEE

      <dynamic prepend="WHERE ">
         <isNull property="id">
            id IS NULL
         </isNull>

         <isNotNull property="id">
            id = #id#
         </isNotNull>
      </dynamic>

   </select>
</sqlMap>
```

You can check a condition using the <isNotEmpty> tag as follows. Here a condition would be added only when a passed property is not empty.

```
..................
<select >
   SELECT * FROM EMPLOYEE

   <dynamic prepend="WHERE ">
      <isNotEmpty property="id">
         id = #id#
      </isNotEmpty>
   </dynamic>

</select>
..................
```

If you want a query where we can select an id and/or the first name of an Employee, your SELECT statement would be as follows −

```
..................
<select >
   SELECT * FROM EMPLOYEE

   <dynamic prepend="WHERE ">
      <isNotEmpty prepend="AND" property="id">
         id = #id#
      </isNotEmpty>
      <isNotEmpty prepend="OR" property="first_name">
         first_name = #first_name#
      </isNotEmpty>
   </dynamic>
</select>
..................
```

Dynamic SQL Example

The following example shows how you can write a SELECT statement with dynamic SQL. Consider, we have the following EMPLOYEE table in MySQL −

```
CREATE TABLE EMPLOYEE (
   id INT NOT NULL auto_increment,
   first_name VARCHAR(20) default NULL,
   last_name  VARCHAR(20) default NULL,
   salary     INT  default NULL,
   PRIMARY KEY (id)
);
```

Let’s assume this table has only one record as follows −

```
mysql> select * from EMPLOYEE;
+----+------------+-----------+--------+
| id | first_name | last_name | salary |
+----+------------+-----------+--------+
|  1 | Zara       | Ali       |   5000 |
+----+------------+-----------+--------+
1 row in set (0.00 sec)
```

Employee POJO Class

To perform read operation, let us have an Employee class in Employee.java as follows −

```
public class Employee {
   private int id;
   private String first_name;
   private String last_name;
   private int salary;

   /* Define constructors for the Employee class. */
   public Employee() {}

   public Employee(String fname, String lname, int salary) {
      this.first_name = fname;
      this.last_name = lname;
      this.salary = salary;
   }

   /* Here are the method definitions */
   public int getId() {
      return id;
   }

   public String getFirstName() {
      return first_name;
   }

   public String getLastName() {
      return last_name;
   }

   public int getSalary() {
      return salary;
   }

} /* End of Employee */
```

Employee.xml File

To define SQL mapping statement using iBATIS, we would add the following modified <select> tag in Employee.xml and inside this tag definition, we would define an "id" which will be used in IbatisReadDy.java for executing Dynamic SQL SELECT query on database.

```
<?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 namespace="Employee">
   <select >
      SELECT * FROM EMPLOYEE

      <dynamic prepend="WHERE ">
         <isNotNull property="id">
            id = #id#
         </isNotNull>
      </dynamic>

   </select>
</sqlMap>
```

The above SELECT statement would work in two ways –

- If you pass an ID, then it returns records corresponding to that ID Otherwise, it returns all the records.

IbatisReadDy.java File

This file has application level logic to read conditional records from the Employee table –

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

import java.io.*;
import java.sql.SQLException;
import java.util.*;

public class IbatisReadDy{
   public static void main(String[] args) throws IOException,SQLException{

      Reader rd=Resources.getResourceAsReader("SqlMapConfig.xml");
      SqlMapClient smc=SqlMapClientBuilder.buildSqlMapClient(rd);

      /* This would read all records from the Employee table.*/
      System.out.println("Going to read records.....");
      Employee rec = new Employee();
      rec.setId(1);

      List <Employee> ems = (List<Employee>)
         smc.queryForList("Employee.findByID", rec);
      Employee em = null;

      for (Employee e : ems) {
         System.out.print("  " + e.getId());
         System.out.print("  " + e.getFirstName());
         System.out.print("  " + e.getLastName());
         System.out.print("  " + e.getSalary());
         em = e;
         System.out.println("");
      }
      System.out.println("Records Read Successfully ");
   }
}
```

Compilation and Run

Here are the steps to compile and run the above mentioned software. Make sure you have set PATH and CLASSPATH appropriately before proceeding for compilation and execution.

- Create Employee.xml as shown above.
- Create Employee.java as shown above and compile it.
- Create IbatisReadDy.java as shown above and compile it.
- Execute IbatisReadDy binary to run the program.

You would get the following result, and a record would be read from the EMPLOYEE table.

```
Going to read records.....
   1  Zara  Ali  5000
Record Reads Successfully
```

Try the above example by passing **null** as *smc.queryForList " Employee. findByID " , null*.

iBATIS OGNL Expressions

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

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

The if Statement

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

```
<select >
   SELECT * FROM BLOG
   WHERE state = 'ACTIVE.

   <if test="title != null">
      AND title like #{title}
   </if>

</select>
```

This statement provides an optional text search type of functionality. If you pass in no title, then all active Blogs are returned. But if you do pass in a title, it will look for a title with the given **like** condition.

You can include multiple **if** conditions as follows −

```
<select >
   SELECT * FROM BLOG
   WHERE state = 'ACTIVE.

   <if test="title != null">
      AND title like #{title}
   </if>

   <if test="author != null">
      AND author like #{author}
   </if>

</select>
```

The choose, when, and otherwise Statements

iBATIS offers a **choose** element which is similar to Java's switch statement. It helps choose only one case among many options.

The following example would search only by title if one is provided, then only by author if one is provided. If neither is provided, it returns only featured blogs −

```
<select >
   SELECT * FROM BLOG
   WHERE state = 'ACTIVE.

   <choose>
      <when test="title != null">
         AND title like #{title}
      </when>

      <when test="author != null and author.name != null">
         AND author like #{author}
      </when>

      <otherwise>
         AND featured = 1
      </otherwise>
   </choose>

</select>
```

The where Statement

Take a look at our previous examples to see what happens if none of the conditions are met. You would end up with an SQL that looks like this −

```
SELECT * FROM BLOG
WHERE
```

This would fail, but iBATIS has a simple solution with one simple change, everything works fine −

```
<select >
   SELECT * FROM BLOG

   <where>
      <if test="state != null">
         state = #{state}
      </if>

      <if test="title != null">
         AND title like #{title}
      </if>

      <if test="author != null>
         AND author like #{author}
      </if>
   </where>

</select>
```

The **where** element inserts a *WHERE* only when the containing tags return any content. Furthermore, if that content begins with *AND* or *OR,* it knows to strip it off.

The foreach Statement

The foreach element allows you to specify a collection and 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 −

```
<select >
   SELECT *
   FROM POST P
   WHERE ID in

   <foreach item="item" index="index" collection="list"
      open="(" separator="," close=")">
      #{item}
   </foreach>

</select>
```

Processing math: 100%