Database Control Sample

Table of contents

1. Control Programming: Simplifying Access to Resources

A Database control makes it easy to access a relational database from your Java code using SQL commands. The Database control handles the work of connecting to the database, so you don't have to understand JDBC to work with a database.

All Database controls are subclassed from the DatabaseControl interface. The interface defines methods that Database control instances can call from an application.

The methods in the Database control execute SQL commands against the database. You can send any SQL command to the database via the Database control, so that you can retrieve data and perform operations like inserts and updates.

A method on a Database control always has an associated SQL statement, which executes against the database when the method is called. The method's @SQL annotation describes the method's SQL statement. The method's SQL statement may include substitution parameters. These parameters are replaced at runtime with the values that were passed to the method. The names of the substitution parameters in the SQL statement must match those in the method signature, so that the Database control knows which parameter to replace with which value.

The following example Database control method illustrates using parameter substitution in Database control methods:

```
    @SQL(statement="INSERT INTO EMPLOYEE " +
                   "(id, fName, lName, title) " +
                   "VALUES ({emp.id}, {emp.fName}, {emp.lName},
{emp.title})")
    public void insertEmployee(Employee emp) throws SQLException;
```

In the example above, the SQL statement includes the substitution {emp.id}, {emp.fName}, ... These map to the various class variables of the "emp" parameter in the insertEmployee method. When the method is invoked, the values of any referenced parameters are substituted in the SQL statement before it is executed. Note that parameter substitution is case sensitive, so parameters mentioned in substitutions must exactly match the spelling and case of the parameters to the method.

The return type of the database operation is determined by the return type of the Java method. The Database control attempts to format the results in whatever type you have specified for the method to return. A method of a Database control can return a single value, a single row, or multiple rows.

When your method returns a single value, its return type must be compatible with the value returned from the query. The following example looks up an employee's title.

```
    @SQL(statement="SELECT title FROM EMPLOYEE WHERE id={id}")
```

```
    public String getEmployeeTitle(int id) throws SQLException;
```

In this example, the title field is of type VARCHAR, so the return value is declared as String.

When your method returns a single row with multiple fields, its return type can be a user-defined object or a java.util.HashMap object. When the return type is a user-defined object, it must contain members with names that match the names of the columns that will be returned by the query. Because database column names are case-insensitive, the matching names are case-insensitive. The class may also contain other members; members not matching any column names will not be set. The following example declares an Employee class with members corresponding to fields in the Employee table. The findEmployee method returns an object of type Employee:

```
    @SQL(statement="SELECT * FROM EMPLOYEE WHERE id={id}")
    public Employee findEmployee(int id) throws SQLException;

    public class Employee
    {
        public int id;
        public String fName;
        public String lName;
        public String title;
    }
```

When your method returns multiple rows from the database, its return type can be an array, a java.util.Iterator, or a java.sql.ResultSet.

When you want to return an Iterator object, you must specify the iteratorElementType element to the @SQL annotation to indicate the underlying type that the Iterator will contain. The following example returns an iterator of all employees sorted by their last names.

```
    @SQL(statement="SELECT * FROM EMPLOYEE ORDER BY lName",
iteratorElementType=Employee.class, maxRows=500)
    public Iterator getEmployeesSortedByLastName() throws SQLException;
```

You can limit the number of rows returned by setting the maxRows element of the @SQL annotation. This element can protect you from very large resultsets that may be returned by very general queries. The default value of maxRow is 1024.

The Database Control defines two annotations: SQL and ConnectionDataSource.

The SQL Annotation specifies the SQL statement and associated attributes that correspond to a method in a Database control.

```
    public @interface SQL
    {
        String statement()              default "";
        int maxRows()                   default MAXROWS_ALL;
        @AnnotationMemberTypes.Optional
        Class iteratorElementType()     default
UndefinedIteratorType.class;
    }
```

The "statement" element is required. It specifies the SQL (Structured Query Language) statement that will be executed when the associated Database control method is invoked. The statement may contain substitutions of the form {varName}, where paramName is a parameter of the Database control method (or a member accessible from the parameter).

The "iteratorElementType" is required if Database control method return type is java.util.Iterator. It specifies the underlying class of the Iterator that will be returned by the Database control method.

The "maxRows" element is optional. It sets the maximum number of records to be returned by the query. The maxRows element limits the size of all types of data sets, including Arrays, Iterator, ResultSet, etc. The default value of this element is 1024.

The ConnectionDataSource annotation specifies the data source that the Database control will use to obtain connection. At the time of writing, this has not been fully implemented yet. The Database control currently uses the embedded Derby database driver to obtain connection, and it is expecting the value provided for the "jndiName" element to be a database URL.

```
    public @interface ConnectionDataSource
    {
            String jndiName();   // no default ... value is required
    }
```

The Database control is currently included in the Beehive distribution as a sample control (\samples\controls-db). Building this control with the Ant script provided will create a `dbControl.jar` in \samples\controls-db\build\. You can start creating your own database control extension by simply including this JAR file in your classpath.

2. The EmployeeWS Sample

Notice that the sample EmployeeWS (../wsm/sample_AddressBook.html) uses this same technique (subclassing DatabaseControl) for database access. The EmployeeWS sample is a database control exposed as a web service. The exposed database control, EmployeeDBControl, is a subclass of DatabaseControl (archived in `dbControl.jar`).

```
public interface EmployeeDBControl extends DatabaseControl
{
    ...
}
```

The Java source for EmployeeDBControl is available at

```
BEEHIVE_HOME/samples/EmployeeWS/WEB-INF/src/org/apache/beehive/sample/EmployeeDBControl
```

3. Building the Database Control

To build `dbControl.jar` run the following Ant command.

On Windows:

```
ant -f %BEEHIVE_HOME%\controls-db\build.xml build
```

This produces the file `dbControl.jar` in the directory `<BeehiveRoot>/samples/controls-db/build`.