JDBC Control Tutorial

Table of contents

1. Overview

1.1. The Problem with JDBC: Complexity

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

The methods that you add to a JDBC Control execute SQL commands against the database. You can send any SQL command to the database via the JDBC Control, so that you can retrieve data, perform operations like inserts and updates, and even make structural changes to the database.

All JDBC controls are subclassed from the JdbcControl interface. The interface defines methods that JDBC control instances can call from an application. See the Tutorial for more detailed information about using the JdbcControl.

2. Tutorial

2.1. JDBC Control Tutorial

The code fragements used in the mini-tutorial are from the jdbcControlSample - the full source can be found in the samples directory of the JdbcControl's source tree.

2.2. Extending the JDBC-Control Interface

The JdbcControl is an extensible control. Before a JdbcControl can be used in an application, a subinterface of the org.apache.beehive.controls.system.jdbc.JdbcControl interface must be created.

```
/**
 * JdbcControl implementation for the JdbcControl sample app.
 */
@org.apache.beehive.controls.api.bean.ControlExtension
@JdbcControl.ConnectionDataSource(jndiName="java:comp/env/jdbc/JdbcControlSampleDB")
public interface SimpleDBControl extends JdbcControl {
                .
                .
                .
}
```

In the sample above several Java 1.5 annotations are used. The @ControlExtension

annotation is required and tells the Beehive control framework that this control extends an extensible control (in this case the JdbcControl).

2.3. Connecting to a Database Instance

The next step is to tell the JdbcControl how to connect to a database instance. This is done using class level Java annotations, there are two annotations which can be used:

- JdbcControl.ConnectionDriver
- JdbcControl.ConnectionDataSource

(i) See the JDBC Control Annotation Reference for additional information about these annotations.

2.4. Making JDBC Calls to a Database Instance

Now that the control knows how to connect to the database instance, the next step is to create methods in the control which access the database. Let's assume we want to access a table in the database which looks like:

```
CREATE TABLE products (name VARCHAR(64), description VARCHAR(128), quantity
INT)
```

Here's what the control might look like:

```
/**
 * JdbcControl implementation for the JdbcControl sample app.
 */
@org.apache.beehive.controls.api.bean.ControlExtension
@JdbcControl.ConnectionDataSource(jndiName="java:comp/env/jdbc/JdbcControlSampleDB")
public interface SimpleDBControl
    extends JdbcControl {

    static final long serialVersionUID = 1L;

    public static class Product {

        private String _name;
        private String _description;
        private int _quantity;

        public int getQuantity() { return _quantity; }
        public void setQuantity(int i) { _quantity = i; }

        public String getName() { return _name; }
        public void setName(String n) { _name = n; }
```

```
        public String getDescription() { return _description; }
        public void setDescription(String n) { _description = n; }
    }

    /**
     * Get the name column from the products table.
     * @return An array of strings.
     */
    @JdbcControl.SQL(statement="SELECT name FROM products")
    public String[] getProductNames() throws SQLException;

    /**
     * Get the rest of the columns associated with a product name.
     * @param productName Name of product to lookup.
     * @return An instance of Product.
     */
    @JdbcControl.SQL(statement="SELECT * FROM products WHERE
name={productName}")
    public Product getProductDetails(String productName) throws
SQLException;
}
```

The SimpleJdbcControl can be accessed from an application as follows:

```
public class Foo {

    // the @Control annotation causes the control to be intialized when
this class is loaded.
    @Control
    public SimpleDBControl jdbcCtrl;

    public void doFoo() {
         String[] productNames = jdbcCtrl.getProductNames();
         Product productInfo = jdbcCtrl.getProductDetails(productNames[3]);
    }
}
```

Note the use of the @SQL method annotation in SimpleDBControl.java, see the JdbcControl Annotation Reference for additional information about the SQL annotation.

2.5. SQL Parameter Substitution

It is also possible to substitute method parameter values into the statement member of the @SQL annotation:

```
//
// simple query with param substitution
//
@SQL(statement="SELECT * FROM USERS WHERE userid={someUserId}")
public ResultSet getSomeUser(int someUserId) throws SQLException;
```

```
//
// query with sql substitution
//
@SQL(statement="SELECT * FROM USERS WHERE {sql: where}")
public ResultSet getJustOneUser(String where) throws SQLException;
```

For the first method, the value of the parameter 'someUserId' gets substituted into the SQL statement at runtime when the getSomeUser() method is invoked. For the second method, the substitution gets added to the SQL statement as the literal value of the 'where' parameter.