Spring JdbcTemplate Querying examples

By mkyong | March 20, 2010 | Updated : June 23, 2011 | Viewed : 1,015,370 | +2,897 pv/w

Here are few examples to show you how to use JdbcTemplate `query()` methods to query or extract data from database.

1. Querying for Single Row

Here's two ways to query or extract a single row record from database, and convert it into a model class.

1.1 Custom RowMapper

In general, It's always recommended to implement the RowMapper interface to create a custom RowMapper to suit your needs.

```
package com.mkyong.customer.model;

import java.sql.ResultSet;
import java.sql.SQLException;

import org.springframework.jdbc.core.RowMapper;

public class CustomerRowMapper implements RowMapper
{
    public Object mapRow(ResultSet rs, int rowNum) throws SQLException {
        Customer customer = new Customer();
        customer.setCustId(rs.getInt("CUST_ID"));
        customer.setName(rs.getString("NAME"));
        customer.setAge(rs.getInt("AGE"));
        return customer;
    }

}
```

Pass it to `queryForObject()` method, the returned result will call your custom `mapRow()` method to match the value into the properly.

```
public Customer findByCustomerId(int custId){

    String sql = "SELECT * FROM CUSTOMER WHERE CUST_ID = ?";

    Customer customer = (Customer)getJdbcTemplate().queryForObject(
            sql, new Object[] { custId }, new CustomerRowMapper());

    return customer;
}
```

1.2 BeanPropertyRowMapper

In Spring 2.5, comes with a handy RowMapper implementation called 'BeanPropertyRowMapper', which can maps a row's column value to a property by matching their names. Just make sure both the property and column has the same name, e.g property 'custId' will match to column name 'CUSTID' or with underscores 'CUST_ID'.

```
public Customer findByCustomerId2(int custId){

    String sql = "SELECT * FROM CUSTOMER WHERE CUST_ID = ?";

    Customer customer = (Customer)getJdbcTemplate().queryForObject(
            sql, new Object[] { custId },
            new BeanPropertyRowMapper(Customer.class));

    return customer;
}
```

2. Querying for Multiple Rows

Now, query or extract multiple rows from database, and convert it into a List.

2.1 Map it manually

In mutiple return rows, RowMapper is not supported in `queryForList()` method, you need to map it manually.

```
public List<Customer> findAll(){

    String sql = "SELECT * FROM CUSTOMER";

    List<Customer> customers = new ArrayList<Customer>();

    List<Map> rows = getJdbcTemplate().queryForList(sql);
    for (Map row : rows) {
        Customer customer = new Customer();
        customer.setCustId((Long)(row.get("CUST_ID")));
        customer.setName((String)row.get("NAME"));
        customer.setAge((Integer)row.get("AGE"));
        customers.add(customer);
    }

    return customers;
}
```

2.2 BeanPropertyRowMapper

The simplest solution is using the BeanPropertyRowMapper class.

```
public List<Customer> findAll(){

    String sql = "SELECT * FROM CUSTOMER";

    List<Customer> customers  = getJdbcTemplate().query(sql,
            new BeanPropertyRowMapper(Customer.class));

    return customers;
}
```

3. Querying for a Single Value

In this example, it shows how to query or extract a single column value from database.

3.1 Single column name

It shows how to query a single column name as String.

```
public String findCustomerNameById(int custId){

    String sql = "SELECT NAME FROM CUSTOMER WHERE CUST_ID = ?";

    String name = (String)getJdbcTemplate().queryForObject(
            sql, new Object[] { custId }, String.class);

    return name;

}
```

3.2 Total number of rows

It shows how to query a total number of rows from database.

```
public int findTotalCustomer(){

    String sql = "SELECT COUNT(*) FROM CUSTOMER";

    int total = getJdbcTemplate().queryForInt(sql);

    return total;
}
```

Run it

```
package com.mkyong.common;

import java.util.ArrayList;
import java.util.List;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.mkyong.customer.dao.CustomerDAO;
import com.mkyong.customer.model.Customer;

public class JdbcTemplateApp
{
    public static void main( String[] args )
    {
         ApplicationContext context =
            new ClassPathXmlApplicationContext("Spring-Customer.xml");

         CustomerDAO customerDAO = (CustomerDAO) context.getBean("customerDAO");

         Customer customerA = customerDAO.findByCustomerId(1);
         System.out.println("Customer A : " + customerA);

         Customer customerB = customerDAO.findByCustomerId2(1);
         System.out.println("Customer B : " + customerB);

         List<Customer> customerAs = customerDAO.findAll();
         for(Customer cust: customerAs){
             System.out.println("Customer As : " + customerAs);
         }

         List<Customer> customerBs = customerDAO.findAll2();
         for(Customer cust: customerBs){
             System.out.println("Customer Bs : " + customerBs);
         }

         String customerName = customerDAO.findCustomerNameById(1);
         System.out.println("Customer Name : " + customerName);

         int total = customerDAO.findTotalCustomer();
         System.out.println("Total : " + total);

    }
}
```

Conclusion

The JdbcTemplate class, comes with many useful overloaded query methods. It's advise to refer to the existing query method before you create own customize query method, because Spring may done it for you already.

Download Source Code

Download it – Spring-JdbcTemplate-Querying-Example.zip (15 KB)

jdbc spring