Building Applications Using

Spring JDBC

— Nandakumar Purohit

Spring JDBC

Agenda

- Overview
- JdbcTemplate
- RowMapper

Spring JDBC

What is Spring JDBC?

Core	The core functionality of JDBC. Some of the important classes under this package include **JdbcTemplate, SimpleJdbcInsert, SimpleJdbcCall and NamedParameterJdbcTemplate**.
Datasource	The utility classes to access a datasource. It also has various datasource implementations for **testing JDBC code** outside the Java EE container.
Object	DB access in an object-oriented manner. It allows **executing queries** and **returning the results** as a business object. It also **maps** the query results between the columns and properties of business objects.
Support	support classes for classes under core and object packages. E.g. provides the **SQLException translation** functionality.

Spring JDBC

The Problem

- Complexity
- Design
- Portability
- Business Focus

Spring JDBC

Agenda

```
public Car getById(String id) {
    Connection con = null;
    PreparedStatement stmt = null;
    ResultSet rs = null;

    try {
        String sql = "select * from CAR where ID = ?";
        con = DriverManager.getConnection("localhost:3306/cars");
        stmt = con.prepareStatement(sql);
        stmt.setString(1, id);
        rs = stmt.executeQuery();
        if(rs.next()) {
            Car car = new Car();
            car.setMake(rs.getString(1));
            return car;
}
else {
            return null;
        }
    } catch (SQLException e) { e.printStackTrace();}
    finally {
        try {
            if(rs != null && !rs.isClosed()) {
                rs.close();
            }
        } catch (Exception e) {}
    }
    return null;
}
```

Spring JDBC

The Solution

- Configuration
- Focus
- Testing
- Dependency Injection

Spring JDBC

How Simplified?

JDBC

```
public Car getById(String id) {
    Connection con = null;
    PreparedStatement stmt = null;
    ResultSet rs = null;

    try {
        String sql = "select * from CAR where ID = ?";
        con = DriverManager.getConnection("localhost:3306/cars");
        stmt = con.prepareStatement(sql);
        stmt.setString(1, id);
        rs = stmt.executeQuery();
        if(rs.next()) {
            Car car = new Car();
            car.setMake(rs.getString(1));
            return car;
}
else {

            return null;
        }
    } catch (SQLException e) { e.printStackTrace();}
    finally {
        try {
            if(rs != null && !rs.isClosed()) {
                rs.close();
            }
        } catch (Exception e) {}
    }
    return null;
}
```

Spring JDBC

```
public Car findCar(String id) {

    return jdbcTemplate.queryForObject(sql, Car.class, id);

}
```

Spring JDBC

JdbcTemplate

The JDBCTemplate functionalities

- Creation and closing of connections
- Executing statements and stored procedure calls
- Iterating over the ResultSet and returning results

Simple Syntax:

```
int result = jdbcTemplate.queryForObject(
      "SELECT COUNT(*) FROM EMPLOYEE", Integer.class);
```

and also here's a simple INSERT:

```
public int addEmplyee(int id) {
      return jdbcTemplate.update("INSERT INTO EMPLOYEE VALUES (?, ?, ?, ?)",
5, "Bill", "Gates", "USA");
   }
```

Spring JDBC

JdbcTemplate

Queries with Named Parameters

```
SqlParameterSource namedParameters = new MapSqlParameterSource().addValue("id", 1);
new NamedParameterJdbcTemplate(dataSource).queryForObject(
          "SELECT FIRST_NAME FROM EMPLOYEE WHERE ID = :id", namedParameters,
String.class);

public Integer countEmployees() {
       Employee employee = new Employee();
       employee.setFirstName("James");

       String SELECT_BY_ID = "SELECT COUNT(*) FROM EMPLOYEE WHERE FIRST_NAME
= :firstName";

       SqlParameterSource namedParameters = new
BeanPropertySqlParameterSource(employee);
       return namedParameterJdbcTemplate.queryForObject(SELECT_BY_ID, namedParameters,
Integer.class);
   }
```

Spring JDBC

RowMapper

- Like ResultSetExtractor, we can use **RowMapper** interface to fetch the records from the database using **query()** method of **JdbcTemplate** class.
- In the execute of we need to pass the instance of **RowMapper** now.

```
public List<Person> getAllPersons() {
        return jdbcTemplate.query(SQL_GET_ALL, new PersonMapper());
   }
```

- RowMapper interface allows to map a row of the **relations** with the instance of **user-defined** class.
- It **iterates** the **ResultSet internally** and adds it into the **collection**.
- **Avoid boiler-plate** code unlike ResultSetExtractor.

Spring JDBC

CRUD

- JdbcTemplate
- SimpleJdbcInsert

Spring JDBC

Create Test

```
@Test(timeout=3000)
   public void testCreateRide() {
        RestTemplate restTemplate = new RestTemplate();
        Ride ride = new Ride();
        ride.setName(“Hillside Trail");
        ride.setDuration(35);
        restTemplate.put(“http://localhost:8089/
ride_tracker/
   ride", ride);
   }
```

Spring JDBC

cRud

- JdbcTemplate
- RowMapper

Spring JDBC

Persistence Mechanisms

Supports:	Serialization	JDBC	ORM	ODB	EJB	JDO	JPA
Java Objects	**Yes**	No	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**
Advanced OO Concepts	**Yes**	No	**Yes**	**Yes**	No	**Yes**	**Yes**
Transactional Integrity	No	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**
Concurrency	No	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**
Large Data Sets	No	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**
Existing Schema	No	**Yes**	**Yes**	No	**Yes**	**Yes**	**Yes**
Rx & Non-Rx Stores	No	No	No	No	**Yes**	**Yes**	No
Queries	No	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**
Strict standards / portability	**Yes**	No	No	No	**Yes**	**Yes**	**Yes**
Simplicity	**Yes**	**Yes**	**Yes**	**Yes**	No	**Yes**	**Yes**