SpringOne Platform by Pivotal

JDBC, what is it good for?

Thomas Risberg

@trisberg

About Me

Thomas Risberg (@trisberg)

- Member of the Spring engineering team at Pivotal
- Contributing to Project riff and Spring Cloud Data Flow
- Joined the Spring Framework open source project in 2003 working on JDBC support

Where it all started ... back in 2002

```
41   * <p>The motivation and design of this class is discussed
42   * in detail in
43   * <a href="http://www.amazon.com/exec/obidos/tg/detail/-/1861007841/">Expert One-On-One J2EE Design and Development</a>
44   * by Rod Johnson (Wrox, 2002).
45   * <br>All SQL issued by this class is logged.
46   * <br>Because this class is parameterizable by the callback interfaces and the
47   * SQLExceptionTranslater interface, it isn't necessary to subclass it.
48   * @author  Rod Johnson
49   * @see com.interface21.dao
50   * @version $Id: JdbcTemplate.java,v 1.1 2003/02/11 08:10:22 johnsonr Exp $
51   * @since May 3, 2001
52   */
53  public class JdbcTemplate {
54
55          //-------------------------------------------------------------------------
56          // Instance data
57          //-------------------------------------------------------------------------
58          /**
59          * Create a Java 1.4-style logging category.
60          */
61          protected final Logger logger = Logger.getLogger(getClass().getName());
62
63          /**
64           * Used to obtain connections throughout
65           * the lifecycle of this object. This enables this class to
66           * close connections if necessary.
67           **/
68          private DataSource dataSource;
69
70          /**
71           * If this variable is false, we will throw exceptions on SQL warnings
72           */
73          private boolean ignoreWarnings = true;
74
75          /** Helper to translate SQL exceptions to DataAccessExceptions */
76          private SQLExceptionTranslater exceptionTranslater;
77
```

Spring solved JDBC try-catch nightmare and more

```
public class JdbcDao {
    private Logger logger = Logger.getLogger("SpringOne");

    public int getBeerCount() {
        Connection conn = null;
        Statement stmt = null;
        ResultSet rs = null;
        int count = 0;
        Properties properties = new Properties();
        try {
            properties.load(new FileInputStream("jdbc.properties"));
        } catch (IOException e) {
            throw new MyDataAccessException("I/O Error", e);
        }
        try {
            Class.forName(properties.getProperty("driverClassName"));
            conn = DriverManager.getConnection(properties.getProperty("url"), properties);
            stmt = conn.createStatement();
            rs = stmt.executeQuery("select count(*) from beers");
            if (rs.next()) {
                count = rs.getInt(1);
            }
        }
        catch (ClassNotFoundException e) {
            throw new MyDataAccessException("JDBC Error", e);
        }
        catch (SQLException se) {
            throw new MyDataAccessException("JDBC Error", se);
        }
        finally {
            if (rs != null) {
                try {
                    rs.close();
                }
                catch (SQLException ignore) {
                }
            }
            if (stmt != null) {
                try {
                    stmt.close();
                }
                catch (SQLException ignore) {
                }
            }
            if (conn != null) {
                try {
                    conn.close();
                }
                catch (SQLException ignore) {
                }
            }
        }
        return count;
    }
}
```

Using straight JDBC code

Spring

Using Spring

```
public class SpringDao {
    private Logger logger = Logger.getLogger("SpringOne");

    public int getBeerCount() {
        DriverManagerDataSource dataSource = new DriverManagerDataSource();
        int count = 0;
        Properties properties = new Properties();
        try {
            properties.load(new FileInputStream("jdbc.properties"));
        } catch (IOException e) {
            logger.severe(e.toString());
        }
        dataSource.setConnectionProperties(properties);
        dataSource.setDriverClassName(properties.getProperty("driverClassName"));
        dataSource.setUrl(properties.getProperty("url"));

        JdbcTemplate jdbcTemplate = new JdbcTemplate(dataSource);
        count = jdbcTemplate.queryForInt("select count(*) from beers");
        return count;
    }
}
```

Spring

From “Spring JDBC” presentation at OSCON 2005
http://docs.huihoo.com/spring/spring-jdbc-oscon-2005.pdf

Spring JDBC - who does what?

	Spring	You
DataSource/Connection Configuration		✓
Connection Management	✓	
Specify the SQL statement and its parameters		✓
Manage statement execution and loop through results	✓	
Retrieve and handle data for each row		✓
Manage the transaction	✓	
Translate exceptions to RuntimeExceptions	✓	

Spring JDBC Support

Choosing an approach for JDBC database access

- **JdbcTemplate** is the classic Spring JDBC approach and the most popular
- **NamedParameterJdbcTemplate** wraps a **JdbcTemplate** to provide named parameters instead of the traditional JDBC "?" placeholders
- **SimpleJdbcInsert** and **SimpleJdbcCall** utilize database metadata to limit the amount of necessary configuration
- RDBMS Objects including **MappingSqlQuery**, **SqlUpdate** and **StoredProcedure** for creating reusable and thread-safe objects during initialization

Demo

Some JDBC Examples

https://github.com/trisberg/s1p2017-jdbc/tree/master/jdbc-demo

What is next for Spring JDBC?

A new **Spring Data JDBC** project

https://github.com/spring-projects/spring-data-jdbc

- Spring Data Repository implementation with JDBC
  - **This is NOT an ORM**
- CRUD operations
- Id generation
- NamingStrategy
- Events
  - BeforeDelete / AfterDelete
  - BeforeSave / AfterSave
  - AfterCreation

Demo

Spring Data JDBC Demo App

https://github.com/gregturn/spring-data-jdbc-demo

Using JDBC in new application architectures

- Microservices / Cloud Deployments
- Event Sourcing / CQRS
- Reactive, Non-Blocking APIs
- Serverless

JDBC in the Cloud

Connecting to RDBMS databases in cloud environments

- Cloud Foundry
- Kubernetes

Cloud Foundry for Spring Boot with JDBC

Running Spring Boot JDBC Apps

- **Java Buildpack** will auto-reconfigure your app injecting its own `DataSource` configuration unless:
  - You have multiple `DataSource` beans defined
  - You have multiple RDBMS services bound to you app
- **Java Buildpack** auto-reconfiguration will also inject a symbolic link to it's jdbc driver if you don't provide one in your app jar
- You can turn auto-reconfiguration off with an environment variable:
  - `JBP_CONFIG_SPRING_AUTO_RECONFIGURATION: '[enabled: false]'`
- You probably still want to set the active profile:
  - `SPRING_PROFILES_ACTIVE: cloud`

Deploying JDBC Spring Boot App on Cloud Foundry

```
cf create-service p-mysql 512mb mysql
cf push -f cloudfoundry/jdbc-demo_manifest.yml \
 -p target/spring-data-jdbc-demo-0.0.1-SNAPSHOT.jar
```

cloudfoundry/jdbc-demo_manifest.yml

```
applications:
- name: jdbc-demo
  disk_quota: 512M
  instances: 1
  memory: 512M
  routes:
  - route: jdbc-demo.local.pcfdev.io
  services:
  - mysql
  stack: cflinuxfs2
```

resources/application-cloud.properties

```
spring.datasource.platform=mysql
spring.datasource.initialization-mode=always
spring.datasource.driverClassName=org.mariadb.jdbc.Driver
```

resources/schema-mysql.sql

```
CREATE TABLE IF NOT EXISTS Employee ( id BIGINT AUTO_INCREMENT
PRIMARY KEY, firstname VARCHAR(100), lastname VARCHAR(100), role
VARCHAR(20))
CREATE TABLE IF NOT EXISTS Manager ( manager_id BIGINT
AUTO_INCREMENT PRIMARY KEY, NAME VARCHAR(100))
```

Deploying JDBC Spring Boot App on Kubernetes

```
kubectl apply -f kubernetes/mysql/
kubectl apply -f kubernetes/
```

kubernetes/jdbc-demo-config.yaml

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: jdbc-demo
  labels:
    app: jdbc-demo
data:
  application.yaml: |-
    spring:
      datasource:
        url: jdbc:mysql://${MYSQL_SERVICE_HOST}:${MYSQL_SERVICE_PORT}/mysql
        username: root
        password: ${MYSQL_ROOT_PASSWORD}
        driverClassName: org.mariadb.jdbc.Driver
        testOnBorrow: true
        validationQuery: "SELECT 1"
```

resources/application-kubernetes.properties

```
spring.datasource.platform=mysql
spring.datasource.initialization-mode=always
spring.datasource.driverClassName=org.mariadb.jdbc.Driver
```

resources/schema-mysql.sql

```
CREATE TABLE IF NOT EXISTS Employee ( id BIGINT AUTO_INCREMENT
PRIMARY KEY, firstname VARCHAR(100), lastname VARCHAR(100), role
VARCHAR(20))
CREATE TABLE IF NOT EXISTS Manager ( manager_id BIGINT AUTO_INCREMENT
PRIMARY KEY, NAME VARCHAR(100))
```

Event Sourcing / CQRS

- Separates reads from writes
- Uses different data models for Event Store and Read Storage
- Might use different type of data stores for Event Store and Read Storage
- JDBC could be a good for for one or both of them

JDBC and Async / Reactive

We are not there yet.

Async JDBC proposal from JavaOne 2017

- Developed by the JDBC Expert Group through the Java Community Process
- The API is available for download from OpenJDK at
  http://oracle.com/goto/java-async-db
- Send feedback to jdbc-spec-discuss@openjdk.java.net

Fake it using thread pools, an async layer and Futures to hide your blocking calls

Serverless JDBC

Running serverless functions accessing RDBMS data

- Spring Cloud Function provides abstraction layer that can be used on:
  - Project riff ✓
  - AWS Lambda ✓
  - Azure Functions
  - Open Whisk
  - Fn Project
  - ...
- Prefer using JDBC rather than JPA/Hibernate for faster cold start

The JDBC Function

```
public class JdbcWriter implements Function<Map<String, Object>, String> {

   @Autowired
    private JdbcTemplate jdbcTemplate;

    private SimpleJdbcInsert insert;

    @PostConstruct
    public void init() {
       this.insert = new SimpleJdbcInsert(jdbcTemplate)
                .withTableName("data")
                .usingColumns("name", "description")
                .usingGeneratedKeyColumns("id");
    }

   @Override
    public String apply(Map<String, Object> data) {
        logger.info("Received: " + data);
        Object name = data.get("name");
        Object description = data.get("description");
        logger.info("Inserting into data table: [" + name + ", " + description +"]");
        SqlParameterSource input = new MapSqlParameterSource("name", name).addValue("description", description);
        Number newId = insert.executeAndReturnKey(input);
        logger.info("NewId is: " + newId);
        return "{ \"newId\": " + newId + " }";
    }

}
```

Spring Cloud Function on AWS Lambda

```
AWSTemplateFormatVersion : '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: Spring Cloud Function with JDBC Writer
Parameters:
  DBPwd:
    NoEcho: true
    Description: The database account password
    Type: String
Resources:
  Employee:
    Type: AWS::Serverless::Function
    Properties:
      FunctionName: JdbcWriter
      Handler: org.springframework.cloud.function.adapter.aws.SpringBootStreamHandler
      Runtime: java8
      CodeUri: s3://trisberg-functions/jdbc-writer-0.0.1-SNAPSHOT-uber.jar
      Description: Demo JDBC Writer on AWS
      Timeout: 30
      MemorySize: 1024
      Environment:
        Variables:
          SPRING_DATASOURCE_URL: jdbc:mysql://springone.clsmkylda5na.us-east-1.rds.amazonaws.com:3306/test
          SPRING_DATASOURCE_USERNAME: master
          SPRING_DATASOURCE_PASSWORD: { "Ref" : "DBPwd" }
          SPRING_DATASOURCE_PLATFORM: mysql
      Role: arn:aws:iam::641162161031:role/lambda-execution-role
```

Spring Cloud Function on Project riff

```yaml
apiVersion: projectriff.io/v1
kind: Topic
metadata:
  name: data
---
apiVersion: projectriff.io/v1
kind: Function
metadata:
  name: jdbc-writer
spec:
  protocol: http
  input: data
  container:
    image: trisberg/jdbc-writer:0.0.1-SNAPSHOT
    env:
    - name: FUNCTION_URI
      value: file:///functions/function.jar?handler=example.JdbcWriter
    - name: SPRING_PROFILES_ACTIVE
      value: kubernetes
    - name: SPRING_DATASOURCE_DRIVER_CLASS_NAME
      value: 'org.mariadb.jdbc.Driver'
    - name: SPRING_DATASOURCE_URL
      value: 'jdbc:mysql://data-mysql:3306/mysql'
    - name: SPRING_DATASOURCE_USERNAME
      value: root
    - name: SPRING_DATASOURCE_PASSWORD
      valueFrom:
        secretKeyRef:
          name: data-mysql
          key: mysql-root-password
```

Demo

Spring Cloud Function JDBC Writer App

https://github.com/trisberg/s1p2017-jdbc/tree/master/jdbc-writer

Learn More. Stay Connected.

https://github.com/spring-projects/spring-framework/tree/master/spring-jdbc

https://github.com/spring-projects/spring-data-jdbc

https://github.com/spring-cloud/spring-cloud-function

https://github.com/projectriff/riff

SpringOne Platform by Pivotal

@s1p #springone