LDS TECH

THE CHURCH OF JESUS CHRIST OF LATTER-DAY SAINTS

Database Development Part 2

Bruce Campbell

BUILD UPON THE TRADITION OF SERVICE!

Notes

- This is a training NOT a presentation
- Please ask questions
- Prerequisites
  - Database Development 1
    - Introduction to Spring
    - Introduction to Spring MVC
    - A basic understanding of SQL
    - LDS Tech IDE Installed

Outline

- Spring JDBC
  - Review
    - JdbcTemplate, RowMapper
  - Compared to vanilla JDBC
  - Exception Translation
  - Named Parameter Options
  - Row mappers Options
  - Lab - Parameter Map & Row Mapper
  - MappingSqlQuery & SQLUpdate
  - Demo & Conclusion

Spring JDBC Review

Spring JDBC Review

- Spring JDBC is good
  - Does lots of the tedious work
  - Cleans up resources automatically
- See Also Reference docs and javadoc API
  http://www.springsource.org/spring-core#documentation

Spring JDBC Review

- Overview
  - Query the database with
    - JdbcTemplate
    - NamedParameterJdbcTemplate
    - SimpleJdbcInsert
  - Map the results into Java objects with
    - RowMapper

Spring JDBC Review

- namedParameterJdbcTemplate.query(...)

```
sql = "select * from EXAMPLE where DATA = :data";
MapSqlParameterSource params = new MapSqlParameterSource();
params.addValue("data", data);
List<Example> examples =
        namedParameterJdbcTemplate.query(
                sql, params, new ExampleRowMapper());
```

Spring JDBC Review

- RowMapper

```
private static class ExampleRowMapper
                                    implements RowMapper<Example> {
    @Override
    public Example mapRow(ResultSet rs, int rowNum)
                                            throws SQLException {
        Example example = new Example();
        example.setId(rs.getLong("ID"));
        example.setName(rs.getString("EXAMPLE_NAME"));
        example.setData(rs.getString("DATA"));
        return example;
    }
}
```

Compare Vanilla JDBC to Spring JDBC

Without Spring JDBC

```
27    private static List<Example> getExamples() {
28        Statement statement = null;
29        ResultSet results = null;
30        try {
31            connection = Session.getConnection();
32            String sql = "select * from EXAMPLE order by DATA";
33            results = statement.executeQuery(sql);
34            List<Example> examples = new ArrayList<>();
35            while (results.next()) {
36                Example example = new Example();
37                example.setId(results.getLong("ID"));
38                example.setName(results.getString("EXAMPLE_NAME"));
39                example.setData(results.getString("DATA"));
40                examples.add(example);
41            }
42            return examples;
43        } catch (SQLException e) {
44            throw new RuntimeException(e);
45        } finally {
46            if (results != null) {
47                try {
48                    results.close();
49                } catch (SQLException ignore) { }
50            }
51            if (statement != null) {
52                try {
53                    statement.close();
54                } catch (SQLException ignore) { }
55            }
56        }
57    }
```

With Spring JDBC

```
public List<Example> getAllExamples() {
     return jdbcTemplate.query("select * from EXAMPLE order by DATA", new ExampleRowMapper());
}

private static class ExampleRowMapper implements RowMapper<Example> {
    @Override
    public Example mapRow(ResultSet results, int rowNum) throws SQLException {
        Example example = new Example();
        example.setId(results.getLong("ID"));
        example.setName(results.getString("EXAMPLE_NAME"));
        example.setData(results.getString("DATA"));
        return example;
    }
}
```

Spring JDBC Exception Mapping

SQLException

- JDBC has exception fixation disease
- Almost everything is a "SQLException"
  - which is difficult to recover from[1]
  - is not very friendly to diagnose
  - messages are database specific
  - is a checked exception
    - require try/catch blocks which are ugly and distracting
  - it's not likely you'll be able to recover anyway

Spring JDBC Exception Mapping

- Spring translates SQLException to DataAccessException subclasses which are
  - independent of database access strategy
  - more informative
    - by default the Oracle error codes are passed through[1]

Spring JDBC Exception Mapping

- Instead of getting SQLException for every exception that occurs, you get for example...
  - DataIntegrityViolationException
    - DuplicateKeyException
  - PermissionDeniedDataAccessException
  - BadSqlGrammarException
  - TypeMismatchDataAccessException
  - RecoverableDataAccessException[1]

Named Parameter Options

Named Parameters

- There are several ways to provide parameter values to the query(), queryFor*(), update(), and execute() methods of namedParameterJdbcTemplate
  - Map<String, ?>
  - MapSqlParameterSource
  - BeanPropertySqlParameterSource

Named Parameters

- Map<String, ?>
  - The key of the map entry should match the parameter name in the query
  - The value of the map gets sub'd into the query
  - The underlying PreparedStatement will guess the corresponding SQL type based on the value type
  - Pass the map as an argument to the namedParameterJdbcTemplate method

Named Parameters

- MapSqlParameterSource
  - Can be constructed
    - blank
    - using a Map
    - using the first parameter name and value
  - addValue() method to add parameter values
    - can chain the calls
    - can specify the sql type

```
public Example getExample(Long id) {
    String sql = "select * from example where id = :id";
    MapSqlParameterSource paramMap = new MapSqlParameterSource("id", id);
    return namedParameterJdbcTemplate.queryForObject(sql, paramMap, new ExampleRowMapper());
}
```

Named Parameters

- BeanPropertySqlParameterSource
  - obtains parameter values from the properties of a given JavaBean object - names must match

```
public List<Example> getExamplesWithMatchingData(Example example) {
    String sql = "select * from example where data = :data";
    return namedParameterJdbcTemplate.query(sql,
            new BeanPropertySqlParameterSource(example),
            new ExampleRowMapper());

}
```

Row Mappers

Row Mappers

- Spring provides several RowMapper implementations to convert ResultSet rows into Java objects
  - ParameterizedBeanPropertyRowMapper
  - ParameterizedSingleColumnRowMapper
  - ColumnMapRowMapper

Row Mappers

- ParameterizedBeanPropertyRowMapper
  - Column values mapped by matching the column name from the metadata of the result set to public setters on the object
    - FIRST_NAME -> setFirstName()
    - NAME -> setName
  - Type mapping is provided for fields in the target class for many common types, e.g.: String, boolean, Boolean, byte, Byte, short, Short, int, Integer, long, Long, float, Float, double, Double, BigDecimal, java.util.Date, etc.
  - Designed for convenience not performance

```
public List<Example> getAllExamples() {
    return jdbcTemplate.query("select id, example_name as name, data from example",
            ParameterizedBeanPropertyRowMapper.newInstance(Example.class));
}
```

- ParameterizedSingleColumnRowMapper
  - converts a single column into a single result value per row, of the specified type
  - must be selecting only one column
  - usage is the same as ParameterizedBeanPropertyRowMapper

- **ColumnMapRowMapper**
  - creates a java.util.Map for each row, representing all columns as key-value pairs: one entry for each column, with the column name as key

```
public List<Map<String, Object>> getExamples() {
    return jdbcTemplate.query("select * from example", new ColumnMapRowMapper());
}
```

Lab

- Parameter Map & Row Mapper Lab
- https://tech.lds.org/wiki/Database_Development_2#Lab_1
- Summary
  - download & unzip the project template
  - import into LDS Tech IDE
  - use a **BeanPropertySqlParameterSource**
  - use a **ParameterizedBeanPropertyRowMapper**
  - verify
- Note: project uses an in-memory database

Solution
Parameter Map & Row Mapper Lab

More Object Oriented Approaches

MappingSqlQuery and SqlUpdate

MappingSqlQuery & SqlUpdate

- MappingSqlQuery & SqlUpdate
  - allow modeling JDBC operations as Java Objects
  - **MappingSqlQuery** for modeling select statements
  - **SqlUpdate** for modeling insert and update statements

MappingSqlQuery & SqlUpdate

- create a class that extends one of them
  - implement mapRow(..) if modeling a select
  - inject a data source
  - specify the SQL string
  - declare parameter placeholders and types
  - specify columns populated by generated keys
  - specify required number of rows affected
  - compile the query

MappingSqlQuery & SqlUpdate

- Use
  - Inject the query object into your DAO
  - Call the appropriate method ...
    - execute(..) methods to retrieve multiple rows
    - or findObject(..) to retrieve one row
    - update(..) or updateByNamedParam(..) to perform an insert or update
    - pass a GeneratedKeyHolder to retrieve the key(s)

MappingSqlQuery & SqlUpdate

- Other Benefits
  - Can isolate database specific code into the query objects
    - the query
    - column names in the row mapper
    - query parameter names
  - Returns a null when no rows are found unlike jdbcTemplate.queryForObject(..) which throws EmptyResultDataAccessException

Demo
MappingSqlQuery & SqlUpdate

- Things to consider
  - CLOB loading is slow in some cases, google it
  - java.sql.Date is different than java.util.Date specifically toString and TimeZone related stuff
    - Be careful
    - Be consistent
    - Consider JodaTime

Conclusion

- Spring JDBC
  - eliminates much of the boilerplate code
  - cleans up resources automatically
  - simplifies database interaction
  - so use it

Questions?