Back To Course

JDBC Simplification with JdbcTemplate Lab ☆

Spring Framework Essentials

48 of 48 lessons completed

100%

Agenda 3m

MODULE 1
Spring Essentials Overview

MODULE 2
Java Configuration

MODULE 3
More on Java Configuration

MODULE 4
Component Scanning

MODULE 5
Inside the Spring Container

MODULE 6
Introducing Aspect Oriented Programming

MODULE 7
Testing Spring Applications

MODULE 8
JDBC Simplification with Jdbc Template

- **Problems with traditional JDBC and into JdbcTemplate** 7m
- **JdbcTemplate Basic Usage** 5m
- **Working with ResultSets using Callbacks** 8m
- **Exception Handling** 5m
- **JDBC Simplification with JdbcTemplate Lab** 45m
- **Demo - JDBC Simplification with JdbcTemplate Lab** 22m

MODULE 9
Transaction Management with Spring

JDBC Simplification with JdbcTemplate Lab

Purpose

In this lab you will gain experience with Spring's JDBC simplification. You will use a `JdbcTemplate` to execute SQL statements with JDBC.

Learning Outcomes

What you will learn:

1. How to retrieve data with JDBC
2. How to insert or update data with JDBC

Specific subjects you will gain experience with:

1. The `JdbcTemplate` class
2. The `RowMapper` interface
3. The `ResultSetExtractor` interface

You will be using the *26-jdbc* project.

Estimated time to complete: 45 minutes.

Prerequisites

The prerequisites are included as part of the *Introduction to the Spring Professional Learning Path* course *Lab Setup* lesson.

If you already completed it, you should be ready to do this lab. If not, assuming you already have JDK 11 or 17 installed and Java IDE, you will need to do following:

1. Download the **lab codebase zip file**.
2. Once you have downloaded the file, unzip it under a directory of your choice. The unzipped directory `core-spring-labfiles/lab` contains the lab projects.
3. Run `./mvnw clean install` (if you plan on using Maven as your build tool).

Use Case

The existing JDBC based repository codebase uses low-level DataSource object directly for performing various database operations, which results in cor and duplicating boiler-plate code.

You are also responsible for mapping the data read from database into the domain objects yourself. This could be a tedious programming task.

In this lab, you are going to refactor this codebase to use Spring-provided `JdbcTemplate` class, which will result in simpler and easy to read code.

Quick Instructions

If you are already knowledgeable with the lesson concepts, you may consider jumping right to the code, and execute the lab in form of embedded TODO comments. Instructions on how to view them are at the *Using TODO Tasks* article.

If you aren't sure, try the TODO instructions first and refer to the lab instructions by TODO number if you need more help.

Instructions

Refactor a Repository to Use `JdbcTemplate`

The first repository to refactor will be the `JdbcRewardRepository`. This repository is the easiest to refactor and will serve to illustrate some of the key fea available because of Spring's simplification.

Use `JdbcTemplate` in a Test to Verify Insertion

Before making any changes to `JdbcRewardRepository`, let's first ensure the existing functionality works by implementing a test.

TODO-01: Use the JdbcTemplate to query for the number of rows

- Open `JdbcRewardRepositoryTests` in the `rewards.internal.reward` package and find the `getRewardCount()` method.

  In this method, use the `jdbcTemplate` included in the test fixture to query for the number of rows in the `T_REWARD table` and return it.

TODO-02: Use the JdbcTemplate to query for a map of all values

- In the same class, find the `verifyRewardInserted(RewardConfirmation, Dining)` method.

  In this method, use the `jdbcTemplate` to query for a map of all values of a row in the `T_REWARD` table based on the `confirmationNumber` of the `RewardConfirmation`. The column name to use for the `confirmationNumber` in the where clause is `CONFIRMATION_NUMBER`.
- Finally run the test class. When you have successful test, move on to the next step.

Refactor `JdbcRewardRepository` to Use `JdbcTemplate`

We are now going to refactor an existing Repository class, which currently uses DataSource object directly with lots of boiler plate code, so it can use the `JdbcTemplate`.

TODO-03: Refactor nextConfirmationNumber() and confirmReward(...) methods to use JdbcTemplate

- Find the `JdbcRewardRepository` in the `rewards.internal.reward` package.

  Open the class and add a private field to it of type `JdbcTemplate`. In the constructor, instantiate the `JdbcTemplate` and assign it to the field you just created.
- Refactor the `nextConfirmationNumber()` method to use the `JdbcTemplate`. This refactoring is a good candidate for using the `queryForObject(String, Class<T>, Object...)` method.

  The `Object...` represents a variable argument list allowing you to append an arbitrary number of arguments to a method invocation, including no arguments at all.
- Next refactor the `confirmReward(AccountContribution, Dining)` method to use the `JdbcTemplate`. This refactoring is a good candidate for us `update(String, Object...)` method.
- Once you have completed these changes, run the test again (`JdbcRewardRepositoryTests`) to ensure these changes work as expected.
- When you have successful test, move on to the next step.

Use a RowMapper to Create Domain Objects

In many cases, you'll want to return domain objects from calls to the database. To do this you'll need to tell the `JdbcTemplate` how to map a single `Resul` row to an object. In this step, you'll refactor `JdbcRestaurantRepository` using a `RowMapper` to create a `Restaurant` object.

TODO-04: Refactor `findByMerchantNumber(..)` method to use `JdbcTemplate`

- Before making any changes, run the `JdbcRestaurantRepositoryTests` test to ensure that the existing implementation functions correctly. When y have successful test, move on to the next step.
- Find the `JdbcRestaurantRepository` in the `rewards.internal.restaurant` package. Open this class and modify it so that it has a `JdbcTemplat` Refactor the constructor to instantiate `JdbcTemplate` object from the given `DataSource` object.
- Create a `RowMapper` object, which you will pass as an argument to the `jdbcTemplate.queryForObject(..)` method.

  You can create a `RowMapper` object in three different ways:

  - Create it as a Lambda expression
  - Create it from an anonymous inner class
  - Write a private inner class and create an object from it

  If you want to write a private inner class, it might look like Figure 1.

  Figure 1: `RestaurantRowMapper` class and method declaration.

```
private class RestaurantRowMapper implements RowMapper<Restaurant> {
    public Restaurant mapRow(ResultSet rs, int rowNum) throws SQLException {
        return mapRestaurant(rs);
    }
}
```

- The implementation of the `mapRow(ResultSet, int)` method of the `RowMapper` object should delegate to the provided `mapRestaurant(ResultSe` method for actually creating a `Restaurant` object.
- Refactor the `findByMerchantNumber(String)` method to use `queryForObject(String, RowMapper<T>, Object...)` method of the `JdbcTemp`
- Run the `JdbcRestaurantRepositoryTests` test again. When you have successful test, move on to the next step.

Refactor the JdbcAccountRepository

In this repository, there are two different methods that need to be refactored: `updateBeneficiaries(Account)` and `findByCreditCard(String)`.

Only do this section if you have enough time left. You will need 10-15 mins.

Refactor a SQL UPDATE

TODO-05: Instantiate `JdbcTemplate`

- Before making any changes, run the `JdbcAccountRepositoryTests` test to ensure the existing implementation functions properly. When you have successful test, move on.
- Find the `JdbcAccountRepository` in the `rewards.internal.account` package. Modify it so that it has a field of type `JdbcTemplate`. Refactor the constructor to instantiate `JdbcTemplate` object from the given `DataSource` object.

TODO-06: Refactor `updateBeneficiaries(..)` to use `JdbcTemplate`

- Refactor the `updateBeneficiaries(Account)` method to use the `JdbcTemplate`. This refactoring is very similar to the one that you did earlier for t `JdbcRewardRepository`.
- When you are done, rerun the `JdbcAccountRepositoryTests` test. When you have successful test, you are good.

Use a `ResultSetExtractor` to Traverse a `ResultSet` for Creating `Account` Objects

This is an optional step. Do this step if you do have some extra time.

**TODO-07 (Optional) **: Refactor `findByCreditCard(..)` method to use `JdbcTemplate` and `ResultSetExtractor`

Sometimes when doing complex joins in a query you'll need to have access to an entire result set instead of just a single row of a result set to build a dom object. To do this you'll need to tell the `JdbcTemplate` that you would like to have a full control over `ResultSet` extraction.

- In this step you'll refactor `findByCreditCard(String)` using a `ResultSetExtractor` to create an `Account` object.

  You can create a `ResultSetExtractor` object in three different ways:

  - Create it as a Lambda expression
  - Create it from an anonymous inner class
  - Write a private inner class and create an object from it

  The implementation of the `extractData(ResultSet)` method of the `ResultSetExtractor` object should delegate to the provided `mapAccount(ResultSet)` method for actually creating an `Account` object.
- Refactor the `findByCreditCard(String)` method to use the `query(String, ResultSetExtractor<T>, Object...)` method of the `JdbcTempla`
- Run the `JdbcAccountRepositoryTests` test once again. When you have successful test, you've completed the lab!
- Note that all three repositories still have a `DataSource` field. Now that you are using the constructor to instantiate the `JdbcTemplate`, you do not ne `DataSource` field anymore. For completeness sake, you can remove the `DataSource` fields if you like.

Inject JdbcTemplate to Repository classes directly (Optional)

Since the repository classes do not use `DataSource` object directly instead use the `JdbcTemplate`, they could be refactored to have the `JdbcTemplate` o to be injected through their constructors.

TODO-08 (Optional): Inject JdbcTemplate directly JdbcRewardRepository class

- Refactor the constructor to get the JdbcTemplate injected directly (instead of DataSource getting injected)
- Refactor RewardsConfig accordingly
- Refactor JdbcRewardRepositoryTests accordingly
- Run JdbcRewardRepositoryTests and verity it passes

TODO-09 (Optional): Inject JdbcTemplate directly JdbcRestaurantRepository class

- Refactor the constructor to get the JdbcTemplate injected directly (instead of DataSource getting injected)
- Refactor RewardsConfig accordingly
- Refactor JdbcRestaurantRepositoryTests accordingly
- Run Jdbc JdbcRestaurantRepositoryTests and verity it passes

TODO-10 (Optional): Inject JdbcTemplate directly JdbcAccountRepository class

- Refactor the constructor to get the JdbcTemplate injected directly (instead of DataSource getting injected)
- Refactor RewardsConfig accordingly
- Refactor JdbcAccountRepositoryTests accordingly
- Run JdbcAccountRepositoryTests and verity it passes

Summary

In this lab, you have refactored several JDBC-based repository codebase to leverage the simplicity of Spring-provided `JdbcTemplate` class for performin database operations.

You also have used `RowMapper` for creating domain objects from the data read from the database.

Summary | Instructors

In this lab you gained experience the JdbcTemplate when working with Databases in Spring and how to configure the completed reference domain.

Give Feedback

Help us improve by sharing your thoughts.

Give Feedback