Spring 6 Jdbc - Data Access with JdbcTemplate

Autor: Hong Le Nguyen last update : 11.2024

Code to example on Github : https://github.com/hong1234/spring6-jdbcTemplate

1 - Infrastructure beans configuration

DataSource enable

We use Spring Jdbc - JdbcTemplate to implement database access layer.

Let's assume that the data source is a MySQL database, and the configuration properties associated with the data sources are stored in the database.properties file.

To inject external configuration properties into env property using @PropertySource("classpath:file-name").

The important beans regarding database access via Jdbc
dataSource: type javax.sql.DataSource, represents database as data source
jdbcTemplate: type JdbcTemplate or NamedParameterJdbcTemplate, represents an Api for database access
dataSourceInitializer (option): type DataSourceInitializer, used for database initialization via SQL-file for example "data.sql"

```
@Configuration
@PropertySource("classpath:database.properties")
public class DataSourceConfig {

  @Autowired
  Environment env;

  @Bean
  DataSource dataSource() {
    DriverManagerDataSource dataSource = new DriverManagerDataSource();
    dataSource.setDriverClassName(env.getProperty("driver"));
        ...
    return dataSource;
  }

  @Bean
  public NamedParameterJdbcTemplate jdbcTemplate(DataSource dataSource) {
    return new NamedParameterJdbcTemplate(dataSource);
  }

  @Bean
  public DataSourceInitializer dataSourceInitializer(DataSource dataSource) {
    ResourceDatabasePopulator populator = new ResourceDatabasePopulator();
    populator.addScript(new ClassPathResource("data.sql"));  // database init per sql file
    DataSourceInitializer initializer = new DataSourceInitializer();
    initializer.setDataSource(dataSource);
    initializer.setDatabasePopulator(databasePopulator);
        ...
    return initializer;
  }
```

Transaction enable

bean transactionManager of type PlatformTransactionManager, represents Transaction Management

```
@Configuration
@EnableTransactionManagement
public class TransactionConfig {
  @Autowired
  DataSource dataSource;

  @Bean
  public PlatformTransactionManager transactionManager(DataSource dataSource) {
    return new DataSourceTransactionManager(dataSource);
  }
}
```

Jdbc configuration in Spring Boot 3

To use JDBC in your Spring Boot application, we add the spring-boot-starter-jdbc dependency. Spring Boot auto-configures the datasource, jdbcTemplate, transactionManager beans.

To override the defaults, you need to provide your own datasource declaration, either JavaConfig or in the application.properties file. For example fine-tuning DataSource bean using *application.properties* file.

```
# MySQL Database as DataSource
spring.datasource.url=jdbc:mysql://localhost:3306/db_name
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
spring.datasource.username=root
spring.datasource.password=password
# Database Init per "schema.sql" or "data.sql" file
spring.sql.init.mode=always
```

2 - Repository implement using JdbcTemplate

The NamedParameterJdbcTemplate class (a JdbcTemplate wrapper)

provides named parameters (:parameterName) instead of the traditional JDBC "?" placeholders, exposes different methods for.

Query operation (SELECT)

> *List<T> query(String sql, SqlParameterSource paramSource, RowMapper<T> rowMapper)*
>
> Description: Query given SQL to create a prepared statement from SQL and a list of arguments to bind to the query, mapping each row to a Java object via a RowMapper.

Insert operation (INSERT)

> *int update(String sql, SqlParameterSource paramSource, KeyHolder generatedKeyHolder)*
>
> Description: Issue an update via a prepared statement, binding the given arguments, returning generated keys.

Update or Delete operation (UPDATE/DELETE)

> *int update(String sql, SqlParameterSource paramSource)*
>
> Description: Issue an update via a prepared statement, binding the given arguments.

An example (Jdbc-)Repository for domain class Person

The JdbcTemplatePersonDAO class becomes jdbcTemplatePersonDao bean through @Repository annotation.
The jdbcTemplate bean is injected using constructor into repository bean.
Binding the given arguments to the sql using parameters object of type SqlParameterSource.
The PersonMapper class maps ResultSet/Row to the Person object.

```
public class PersonMapper implements RowMapper<Person> {
  public Person mapRow(ResultSet rs, int i) throws SQLException {
    Person person = new Person();
    person.setId(rs.getInt("id"));
    person.setFirstName(rs.getString("first_name"));
        ...
    return person;
  }

@Repository
public class JdbcTemplatePersonDAO implements PersonDAO {

  private final String SQL_QUERY_PERSON = "select * from people where id = :id";

  private final String SQL_INSERT_PERSON =
        "insert into people(first_name, last_name, age, gender, created_on) values(:fname, :lname,
        :age, :gender, :createdOn)";

  private final NamedParameterJdbcTemplate jdbcTemplate;

  public JdbcTemplateDAO(NamedParameterJdbcTemplate jdbcTemplate) {
    this.jdbcTemplate = jdbcTemplate;
  }

  public Person getPersonById(Integer id) {
    SqlParameterSource parameters = new MapSqlParameterSource();
    parameters.addValue("id", id);
    return jdbcTemplate.queryForObject(
                SQL_QUERY_PERSON, parameters, new PersonMapper());
  }

  public Person createPerson(Person person) {
    SqlParameterSource parameters = new MapSqlParameterSource();
    parameters.addValue("fname", person.getFirstName());
        ...
    KeyHolder generatedKeyHolder = new GeneratedKeyHolder();
    jdbcTemplate.update(SQL_INSERT_PERSON, parameters, generatedKeyHolder);
    Number key = generatedKeyHolder.getKey();
    return getPersonById(key.intValue());
  }
```

3 - Transaction consistency enable

The transaction manager

A transaction is a group of SQL operations.

Transactions must be managed by the transaction manager bean. The transaction manager bean's configuration is the only thing that has to be changed when the environment changes. In Spring JDBC environment, a local JDBC configuration declaring a basic datasource to be used and a bean of type org.springframework.jdbc.datasource.DataSourceTransactionManager.

The @Transactional annotation

is used to mark a method or a class as transactional, meaning that any database operations performed within the marked method or class will be executed within a transaction.

If the transaction is successful, the changes will be committed to the database. If an error occurs and the transaction is rolled back, the changes will not be persisted in the database.

And here is an example of how the @Transactional annotation can be used on a Service class or on a method:

```
@Service
@Transactional
public class PersonManager {
   private final PersonDAO  personDao;
   public void addPersons() {
      // perform database operations here
   }

@Service
public class PersonManager {
   ...
   @Transactional
   public void multiOperations() {
      // perform database operations here
   }
```

The similar can be done with Repository class or method

```
@Repository
@Transactional
public class JdbcTemplatePersonDAO implements PersonDAO {
   ...
   // @Transactional
   public void updatePerson(Person person) {
      // perform database update here
   }
}
```