Spring Jdbc Template

Overview

Spring provides various strategies for database access, one of which is JDBC.

While JDBC is a low-level API, it requires writing extensive boilerplate code.

To simplify this, Spring provides JdbcTemplate, a helper class that:

- Reduces repetitive JDBC code.
- Manages database connections automatically.
- Improves exception handling.

Implementing JDBC in Spring

A good design pattern involves:

- Creating a DAO class that implements an interface containing all CRUD methods.
- Injecting the DAO class into a service layer class to perform database operations.

For example:

```
public class BookServiceProductionImpl implements BookService {
	private BookDao dao;
	public BookServiceProductionImpl(BookDao dao) {
		this.dao = dao;
	}
	@Override
	public List<Book> getAllBooksByAuthor(String author) {
		return dao.findBooksByAuthor(author);
	}
```

Wiring JdbcTemplate in Spring XML

To integrate JdbcTemplate in Spring XML configuration, we modify the service bean:

```
<beans>
    <!-- Dao beans -->
    <bean id="bookDao" class="se.yrgo.spring.data.BookDaoJdbcImpl" />

    <bean id="bookService"
      class="se.yrgo.spring.services.BookServiceProductionImpl">
        <constructor-arg ref="bookDao">
        </constructor-arg>
    </bean>
</beans>
```

Using JdbcTemplate in DAO Implementation

1. Injecting JdbcTemplate:

JdbcTemplate is used to manage database operations efficiently. We declare it as a dependency and inject it:

```
private JdbcTemplate jdbcTemplate;


public BookDaoSpringJdbcImpl(JdbcTemplate jdbcTemplate) {
    this.jdbcTemplate = jdbcTemplate;
}
```

This class comes from org.springframework.jdbc.core.

CRUD Operations with JdbcTemplate

2. Inserting Data (Create Operation):

JdbcTemplate's update() method allows inserting a new book:

```
public void create(Book newBook) {
   jdbcTemplate.update(INSERT_BOOK_SQL, newBook.getIsbn(),
       newBook.getTitle(), newBook.getAuthor(), newBook.getPrice());
}

private static final String INSERT_BOOK_SQL = "insert into BOOK
(ISBN, TITLE, AUTHOR,PRICE) values (?, ?, ?, ?) ";
```

CRUD Operations with JdbcTemplate

3. Retrieving Data (Read Operation)

To fetch data, we use query() along with a custom RowMapper class:

```
@Override
public List<Book> allBooks() {
   return jdbcTemplate.query("SELECT * FROM Book", new
           BookMapper());
}

@Override
public List<Book> findBooksByAuthor(String author) {
        return jdbcTemplate.query("select * from Book where
                    author=?", new BookMapper(), author);
}
```

Using RowMapper for Result Mapping

Since Spring doesn't automatically convert ResultSet into objects, we define a RowMapper:

```
class BookMapper implements RowMapper<Book> {
     @Override
     public Book mapRow(ResultSet rs, int rowNumber) throws SQLException {
          String isbn = rs.getString("ISBN");
          String title = rs.getString("title");
          String author = rs.getString("author");
          double price = rs.getDouble("price");
          Book book = new Book(isbn, title, author, price);
          return book;
     }
}
```

This class implements RowMapper from org.springframework.core.

Deleting Data

4. Deleting a Book:

```
public void delete(Book redundantBook) {
   jdbcTemplate.update("Delete from Book where ISBN=?",
                       redundantBook.getIsbn());
}
```

Creating Tables on Startup

To ensure tables exist, we can use update() inside a constructor:

```
public BookDaoSpringJdbcImpl(JdbcTemplate jdbcTemplate) {
    this.jdbcTemplate=jdbcTemplate;
    try {
         jdbcTemplate.update(CREATE_TABLE_SQL);
    }catch(Exception e) {
          System.err.println("Table already exists");
    }
}

private static final String CREATE_TABLE_SQL = "create table BOOK(ISBN
VARCHAR(20), TITLE VARCHAR(50), AUTHOR VARCHAR(50), PRICE DOUBLE)";
```

Conclusion

- JdbcTemplate simplifies JDBC interactions by handling connections, exceptions, and query execution.
- XML configuration allows easy dependency injection of the DAO layer.
- RowMapper is used to map query results to Java objects.
- CRUD operations are significantly shorter compared to raw JDBC implementations.

Next Step

In the next section, we will explore Connection Pools in Spring for efficient database management