**Query Execution:**

Discuss the usage of JdbcTemplate for executing SQL queries in a Spring application. Provide examples of querying for a single row (queryForObject()), querying for multiple rows (query()), and executing an update (update()).

Using JdbcTemplate in a Spring application simplifies the execution of SQL queries and updates. It provides several methods to interact with the database, such as queryForObject(), query(), and update(). Let's explore each of these methods with examples.

**1. Querying for a Single Row (queryForObject())**

The queryForObject() method is used when you expect exactly one row to be returned from the database query.

**Example:**

Assume you have a users table with columns id, username, and email. Here's how you can use queryForObject() to fetch a single user's information based on their id:

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.jdbc.core.JdbcTemplate;

import org.springframework.jdbc.core.RowMapper;

import org.springframework.stereotype.Repository;

@Repository

public class UserDao {

@Autowired

private JdbcTemplate jdbcTemplate;

public User getUserById(int userId) {

String sql = "SELECT id, username, email FROM users WHERE id = ?";

// RowMapper to map ResultSet rows to User objects

RowMapper<User> rowMapper = (rs, rowNum) -> {

User user = new User();

user.setId(rs.getInt("id"));

user.setUsername(rs.getString("username"));

user.setEmail(rs.getString("email"));

return user;

};

// Execute queryForObject

return jdbcTemplate.queryForObject(sql, new Object[]{userId}, rowMapper);

}

}

In this example:

* User is a simple POJO (Plain Old Java Object) representing the structure of a user.
* RowMapper is used to map each row from the ResultSet to a User object.
* queryForObject() executes the SQL query with parameters (userId) and returns a single User object.

### 2. Querying for Multiple Rows (query())

The query() method is used when you expect multiple rows to be returned from the database query.

#### Example:

Let's extend our UserDao to fetch all users from the users table:

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.jdbc.core.JdbcTemplate;

import org.springframework.jdbc.core.RowMapper;

import org.springframework.stereotype.Repository;

import java.util.List;

@Repository

public class UserDao {

@Autowired

private JdbcTemplate jdbcTemplate;

public List<User> getAllUsers() {

String sql = "SELECT id, username, email FROM users";

// RowMapper to map ResultSet rows to User objects

RowMapper<User> rowMapper = (rs, rowNum) -> {

User user = new User();

user.setId(rs.getInt("id"));

user.setUsername(rs.getString("username"));

user.setEmail(rs.getString("email"));

return user;

};

// Execute query

return jdbcTemplate.query(sql, rowMapper);

}

}

In this example:

* getAllUsers() method retrieves all users from the users table.
* query() method is used without parameters because the query retrieves all rows.
* It returns a list of User objects mapped using the RowMapper.

### 3. Executing an Update (update())

The update() method is used for executing INSERT, UPDATE, DELETE, or any SQL statement that doesn't return data.

#### Example:

Let's add an insertUser() method to our UserDao to insert a new user into the users table:

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.jdbc.core.JdbcTemplate;

import org.springframework.stereotype.Repository;

@Repository

public class UserDao {

@Autowired

private JdbcTemplate jdbcTemplate;

public void insertUser(User user) {

String sql = "INSERT INTO users (username, email) VALUES (?, ?)";

// Execute update

jdbcTemplate.update(sql, user.getUsername(), user.getEmail());

}

}

In this example:

* insertUser() method inserts a new user into the users table using update() method.
* The SQL query contains placeholders (?) which are replaced by parameters (user.getUsername() and user.getEmail()).