УДК 004

Написание динамических запросов с помощью Spring Data JPA

Еровлева Регина Викторовна
Приамурский государственный университет имени Шолом-Алейхема
Студент

Еровлев Павел Андреевич
Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация
В данной статье описывается процесс использования динамический запросов с помощью Spring Data JPA. Разобраны примеры написания запросов.
Ключевые слова: Spring Boot, Spring Data, JPA

Writing dynamic queries with Spring Data JPA

Erovleva Regina Viktorovna
Sholom-Aleichem Priamursky State University
student

Erovlev Pavel Andreevich
Sholom-Aleichem Priamursky State University
Student

Abstract
This article walks you through the process of using dynamic queries with Spring Data JPA. The examples of writing requests are analyzed.
Keywords: Spring Boot, Spring Data, JPA

Spring Data - предоставляет знакомую и последовательную модель программирования на основе Spring для доступа к данным, сохраняя при этом особые черты базового хранилища данных.

Это упрощает использование технологий доступа к данным, реляционных и нереляционных баз данных, фреймворков с сокращением карт и облачных сервисов данных. Это зонтичный проект, который содержит множество подпроектов, специфичных для данной базы данных. Проекты разрабатываются в сотрудничестве со многими компаниями и разработчиками, стоящими за этими захватывающими технологиями.

С помощью Spring Data очень легко создать простое приложение CRUD по созданию собственных DAO. Spring Boot упрощает задачу, так что даже не нужно создавать файл «EntityManager» самостоятельно.

П.В. Прохоров, Н.В. Разговоров рассмотрели в своей работе современные подходы и технологии в разработке серверных приложений на примере онлайн-магазина с использованием Spring [1]. В своей статье В.С. Жданова описала подход в реализации серверной части клиент-серверного мобильного приложения для просмотра расписания [2]. В своей статье А.И.Егунова и др. рассматривают проблемы повышения эффективности образовательной деятельности вуза и использование научной интеллектуальной собственности в преподавательской деятельности. Так же предложили реализацию информационного хранилища и поисковой системы в виде J2EE-приложения с использованием фреймворка Spring [3]. Так же А.Д. Нарижный и Н.Е. Губенко провели сравнительный анализ технологий, которые имеют схожую функциональность и которые предназначены для одних и тех же задач. Это технологии стеков Spring и JavaEE, которые предназначены для разработки Enterprise-приложений [4]. В.Л. Волушкова рассмотрела в своей работе технологии программирования на примере языка Java [5].

Одна из возможностей более динамичного контроля над запросами - это использование «Example API». Можно использовать этот API для создания примера объекта для дальнейшего использования в запросе.

Например, допустим, есть следующая сущность(рис.1).

```
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
@Entity
@Table(name = "marvel_character")
public class MarvelCharacter {
    @Id
    @Column(name = "hero_name")
    private String heroName;
    @Column(name = "first_name")
    private String firstName;
    @Column(name = "last_name")
    private String lastName;
}
```

Рисунок 1 – Пример сущностей

Кроме того, определяем службу для необязательной отправки имени и фамилии. Когда эти параметры являются «null», то запрос не должен фильтровать по этим полям (рис.2).

```
public List<MarvelCharacter> findByName(String firstName, String lastName) {

}
```

Рисунок 2 – Метод поиска по атрибутам.

Используя простые запросы, можно сделать несколько методов

- findByFirstName(String firstName)
- findByLastName(String lastName);
- findByFirstAndLastName(String firstName, String lastName).

Данные запросы, плохо масштабируется, так как для каждого нового параметра количество запросов удваивается. Используя пример API, можно использовать «findAll(Example)» метод, доступный в «JpaRepository» (рис.3).

```
MarvelCharacter example = MarvelCharacter
    .builder()
    .firstName(firstName)
    .lastName(lastName)
    .build();
return repository.findAll(Example.of(example));
```

Рисунок 3 – Масштабируемый метод

Преимущество этого API в том, что он намного лучше масштабируется, поскольку «null» значения отбрасываются. Это означает, что если «lastName» будет «null», то запрос проигнорируется.

Кроме того, можно изменить способ сопоставления примеров, указав «ExampleMatcher» параметр. Например, предположим, что необходимо отфильтровать «firstName» параметр без учета регистра и отфильтровать имена, которые содержат заданное значение, а не являются точным совпадением .

В этом случае можно написать такой сопоставитель (рис.4).

```
ExampleMatcher matcher = ExampleMatcher
    .matchingAll()
    .withMatcher("firstName", contains().ignoreCase());
MarvelCharacter example = MarvelCharacter
    .builder()
    .firstName(firstName)
    .lastName(lastName)
    .build();
return repository.findAll(Example.of(example, matcher));
```

Рисунок 4 – Сопоставитель

В этом примере «contains()» метод представляет собой статический импорт «ExampleMatcher.GenericPropertyMatchers.contains()», что позволяет писать его более кратко.

Этот пример API можно использовать во многих случаях, но он все еще ограничен и не может использоваться для всех сценариев. Другая возможность, которая позволяет делать практически все что угодно, - это использование «Specification API».

API спецификации - это абстракция поверх API критериев JPA, что означает, что можно делать все, что нужно сделать с этими критериями.

Если взять пример, который использовали ранее, и преобразовать его в спецификации, можно написать следующие спецификации (рис.5)

```
public static Specification<MarvelCharacter> firstNameContains(String expression) {
    return (root, query, builder) -> builder.like(root.get("firstName"), contains(expression));
}

public static Specification<MarvelCharacter> lastNameContains(String expression) {
    return (root, query, builder) -> builder.like(root.get("lastName"), contains(expression));
}

private static String contains(String expression) {
    return MessageFormat.format("%{0}%", expression);
}
```

Рисунок 5 – Эффективные спецификации

В этом случае есть две спецификации «firstNameContains()» и «lastNameContains()». Поскольку Specification интерфейс содержит только один метод, можно записать их как лямбда-выражения.

Этот метод интерфейса передает три аргумента, которые можно использовать:

1. Первый параметр (root) позволяет выбрать поле для фильтрации. Это можно сделать с помощью «root.get("name")» или, если нужно присоединиться, то можно использовать «root.join("myField").get("name")».
2. Второй параметр (query) используется не так часто, но содержит информацию о типе выполняемого запроса.
3. Последний параметр является «CriteriaBuilder», что позволяет точно определить, какой тип запроса необходимо построить (LIKE, IS NULL, CONTAINS, AND, OR, =, ...).

Кроме того, необходимо изменить репозиторий, чтобы он расширялся от «JpaSpecificationExecutor» (рис.6).

```
public interface MarvelCharacterRepository extends JpaRepository<MarvelCharacter, String>, JpaSpecificationExecutor<MarvelCharacter> {
}
```

Рисунок 6 – Изменение репозитория

Как только это будет сделано, то реализуем сервисный метод(рис.7).

```
Specification<MarvelCharacter> specification = Specification

    .where(firstName == null ? null : firstNameContains(firstName))

    .and(lastName == null ? null : lastNameContains(lastName));
```

Рисунок 7 – Сервисный метод

Хорошая вещь с API Specification является то, что можно правильно связать эти спецификации с использованием «and()» и «or()» метода. Кроме того, «nul» значения отфильтровываются, поэтому, если вернуть «null» вместо фактической спецификации, то сможем правильно отфильтровать их в зависимости от входных значений. Это означает, что если в «firstName» есть «null», то не будет происходить фильтрация по «firstNameContains()».

Такой подход позволяет также писать более сложные спецификации (рис.8).

```
public static Specification<MarvelCharacter> lastNameIn(String... values) {
    return (root, query, builder) -> builder.or(Arrays
        .stream(values)
        .map(value -> builder.equal(root.get("lastName"), value))
        .toArray(Predicate[]::new));
}
```

Рисунок 8 – Новая спецификация

В данной статье были рассмотрены примеры написания динамичных запросов с помощью API спецификаций.

Библиографический список

4. Прохоров П.В., Разговоров Н.В. Современные подходы в backend разработке на примере онлайн-магазина // Прикладная математика и фундаментальная информатика. 2020. №2. С. 23-28.
5. Жданова В.С. Серверный модуль системы уведомления об изменениях в расписании занятий // Молодость. Интеллект. Инициатива. 2017. С. 21-22.
6. Егунова А.И., Аббакумов А.А., Воропаева М.А., Вечканова Ю.С. Система управления хранилищем электронных образовательных ресурсов // Образовательные технологии и общество. 2019. №3. С. 145-154.
7. Нарижный А.Д., Губенко Н.Е. Сравнительный анализ стеков технологий spring и javaee (jakartaee) для разработки enterprise приложений // Информатика, управляющие системы, математическое и компьютерное моделирование. 2020. №3. С. 549-462.
8. Волушкова В.Л. Архитектурные решения java для доступа к данным // Теоретические основы программирования. Учебное пособие. 2019. 137с.