Возможности JPA Repository

- CRUD-операции
- Derived Query Methods
- Сортировка
- Пагинация
- Объединение пагинации и сортировки
- Кастомные запросы

В этом уроке мы подробнее обсудим `JpaRepository` — он устроен интереснее, чем может показаться на первый взгляд. С одной стороны, он предоставляет множество полезных встроенных методов, а с другой — поддерживает автоматическую генерацию кастомных методов для извлечения данных. Далее мы разберем эти возможности.

CRUD-операции

Сюда входит базовый набор методов для создания, обновления, удаления и выборки данных:

- Метод `S save(S entity)` сохраняет данные в базу:

```
var user = new User("John Doe");
userRepository.save(user);
```

- Метод `Optional<T> findById(ID id)` извлекает сущность по `id`:

```
var maybeUser = userRepository.findById(id);
```

- Метод `List<T> findAll()` возвращает список всех сущностей, что полезно для справочников и других небольших таблиц:

```
var users = userRepository.findAll();
```

- Метод `long count()` возвращает количество сущностей, то есть записей в таблице·

```
var count = userRepository.count();
```

- Метод `void deleteById(ID id)` удаляет сущность (запись в базе) по id
- Метод `void delete(T entity)` удаляет переданную сущность из базы данных:

```
userRepository.deleteById(id);
userRepository.delete(user);
```

Derived Query Methods

Одна из типовых задач — это выборка по определенному полю или набору полей. Spring Data JPA автоматически генерирует методы, выполняющие подобные выборки. Для этого надо добавить определение нужного метода или методов:

```
@Repository
public interface UserRepository extends JpaRepository<User, Long> {
    Optional<User> findByEmail(String email);
    List<User> findAllByEmail(String email);

    // Поддерживаются OR и AND
    List<User> findAllByEmailOrUsername(String email, String username);
    List<User> findAllByEmailAndUsername(String email, String username);
}
```

Кроме точного сопоставления параметров, этот механизм умеет генерировать код для множества других условий. Большая часть из них транслируется в SQL достаточно очевидным образом:

```
@Repository
public interface UserRepository extends JpaRepository<User, Long> {
    // name != value
    List<User> findAllByNameIsNot(String name);
    // name IS NULL
    List<User> findAllByNameIsNull();
    // name IS NOT NULL
    List<User> findAllByNameIsNotNull();
    // active = 'true'
    List<User> findByActiveTrue();
```

```
    // name LIKE 'value%'
    List<User> findByNameStartingWith(String prefix);
    // name LIKE '%value'
    List<User> findByNameEndingWith(String suffix);

    List<User> findByAgeLessThan(Integer age);
    List<User> findByAgeLessThanEqual(Integer age);

    List<User> findByAgeBetween(Integer startAge, Integer endAge);

    List<User> findByAgeIn(Collection<Integer> ages);

    List<User> findByNameOrderByName(String name);
    List<User> findByNameOrderByNameDesc(String name);
}
```

Сортировка

Сортировка данных выполняется с помощью комбинации методов и вложенных классов `Sort`:

```
import org.springframework.data.domain.Sort;

// Добавляет ORDER
var users = userRepository.findAll(Sort.by(Sort.Order.asc("name")));
```

Сортировка по возрастанию выполняется с помощью `Sort.Order.asc()`, по убыванию — с помощью `Sort.Order.desc()`.

Пагинация

Пагинация — это выборка только определенного среза данных с помощью конструкции `LIMIT OFFSET`. Это основной способ выборки наборов данных, потому что полные наборы обычно слишком большие — извлекать их целиком неудобно:

```
import org.springframework.data.domain.PageRequest;

// Добавляет LIMIT 5 OFFSET 0
```

```
// 0 — это страница, 5 — это количество элементов
// Возвращает Page<User>
var usersPage = userRepository.findAll(PageRequest.of(0, 5));
```

Обычно текущая страница приходит как параметр запроса. По умолчанию страница равна единице. Это не совпадает с тем, как работает `PageRequest` — он отображается напрямую на `OFFSET`, где базовое значение равно `0`. Поэтому для правильной работы пагинации нужно выполнить две задачи:

- Установить `1` в качестве значения параметра запроса `page` по умолчанию
- При формировании `PageRequest` вычитать единицы из `page`

```
public Page<User> index(@RequestParam(defaultValue = "1") int page) {
    var usersPage = userRepository.findAll(PageRequest.of(page - 1, 5));
    return usersPage;
}
```

Объединение пагинации и сортировки

Если нужно объединить пагинацию и сортировку, можно задавать сортировку через `PageRequest`. В итоге код будет выглядеть так:

```
var sort = Sort.by(Sort.Order.asc("name"));
var pageRequest = PageRequest.of(0, 5, sort);
var usersPage = userRepository.findAll(pageRequest);
```

Кастомные запросы

Выше мы перечислили множество разных вариантов. Несмотря на это, в некоторых ситуациях все таки придется написать SQL-код. Чтобы это сделать, нужно добавить определение метода с аннотациями `@Param` и `@Query`:

```
package io.hexlet.spring.repository;

import java.util.List;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
```

```java
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;

import io.hexlet.spring.model.User;

@Repository
public interface UserRepository extends JpaRepository<User, Long> {
    @Query("SELECT e FROM User e WHERE e.name LIKE %:name%")
    List<User> findAllByNameContaining(@Param("name") String name);
}
```

Советуем писать кастомные запросы только тогда, когда не остается другого выбора.

Дополнительные материалы

1. Создание запросов из имени метода

Далее →