JAVA SPRING DATA

Prof. Júlio Machado

julio.machado@pucrs.br

SPRING DATA JPA

Repositórios

Spring Data

- A interface central para implementação do Padrão *Repository* é *Repository<T,ID>*
  - *T* é o tipo da entidade a ser gerenciada
  - *ID* é o tipo do identificador da entidade
  - É apenas uma interface de “marcação”, sem qualquer método, que funciona como base para as demais interfaces

Spring Data

- Fornece diferentes implementações de código-base para todas as operações de CRUD a partir de interfaces tais como:
  - *CrudRepository<T,ID>* - operações genéricas de CRUD que manipulam coleções *Iterable<T>*
    - https://docs.spring.io/spring-data/commons/docs/current/api/org/springframework/data/repository/CrudRepository.html
  - *ListCrudRepository<T,ID>* - operações genéricas de CRUD que manipulam retornam *List<T>*
    - https://docs.spring.io/spring-data/commons/docs/current/api/org/springframework/data/repository/ListCrudRepository.html
  - *PagingAndSortingRepository<T,ID>* - estende as operações de consulta para suportar paginação e ordenação
    - https://docs.spring.io/spring-data/commons/docs/current/api/org/springframework/data/repository/PagingAndSortingRepository.html

Spring Data

```
public interface CrudRepository<T, ID
extends Serializable> extends Repository<T,
ID> {
    <S extends T> S save(S entity);
    T findOne(ID primaryKey);
    Iterable<T> findAll();
    Long count();
    void delete(T entity);
    boolean exists(ID primaryKey);
    ...
}
```

Spring Data

```
public interface
PagingAndSortingRepository<T, ID>  {
  Iterable<T> findAll(Sort sort);
  Page<T> findAll(Pageable pageable);
}
```

Spring Data - Paginação

- XXXX
- Exemplo:

```
Page<User> users =
repository.findAll(PageRequest.of(1, 20));
```

Spring Data - Ordenação

- XXXX
- Exemplo:

```
Page<User> users =
repository.findAll(PageRequest.of(1, 20));
```

Spring Data - Customização

- Framework permite estender as interfaces de repositórios para criação automática de código.
- Customização de novas operações de CRUD se dá através de:
  - Query methods
  - Implementação customizada

Spring Data - Query Methods

- Framework permite estender as interfaces de repositórios para criação automática de código:
  - Através da derivação da query a partir da assinatura do método;
  - Através da definição manual da query (usualmente via linguagem JPQL).

Spring Data - Query Methods

- Derivação de query a partir da assinatura dos métodos:
  - Assinatura do método é dividida em duas partes: *subject* e *predicate*.
  - A primeira parte define o *subject* até o termo “By”.
    - Exemplo: find...By, exists...By, count...By, delete...By
    - Diferentes composições de termos pré-definidos podem aparecer até o termo “By”.
    - https://docs.spring.io/spring-data/commons/docs/current/reference/html/#appendix.query.method.subject
  - A segunda parte define o *predicate* iniciando após o termo “By”.
    - O *predicate* pode conter combinações lógicas de diferentes operadores.
    - https://docs.spring.io/spring-data/commons/docs/current/reference/html/#appendix.query.method.predicate

Spring Data - Query Methods

```
interface UserRepository extends
CrudRepository<User, Long> {
  long countByLastname(String lastname);
  long deleteByLastname(String lastname);
  List<User> removeByLastname(String lastname);
}
```

Spring Data - Query Methods

- Derivação de query a partir da assinatura dos métodos:
  - Expressões na assinatura do método devem referenciar o nome correto das propriedades das entidades gerenciadas.
  - É permitido “navegar” entre as propriedades.
  - Ex.:
    - Entidade *Person* possui uma propriedade *address* que referencia um objeto que possui uma propriedade *zipCode*.

```
List<Person> findByAddress_ZipCode(ZipCode zipCode);
```

Spring Data - Query Methods

- Derivação manual da query:
  - Método é anotado com a consulta definida através da anotação @Query.
  - Convém seguir a padronização das assinaturas dos query methods.

Spring Data - Query Methods

```
interface UserRepository extends
CrudRepository<User, Long> {

  @Query("select u from User u where u.firstname =
:firstname")
  List<User> findByFirstname(String firstname);

  @Query("select u from User u where u.firstname =
:name or u.lastname = :name")
  List<User> findByFirstnameOrLastname(String name);
}
```

Spring Data - Query Methods

- Framework permite a criação de interfaces básicas seletivas que não fornecem todas operações de CRUD.
  - Criar nova interface que estende *Repository*;
  - Anotar a interface com @*NoRepositoryBean* para impedir que o framework crie uma implementação concreta de interface;
  - Copiar os métodos das interfaces que se deseja.

Spring Data - Query Methods

```
@NoRepositoryBean
interface MyBaseRepository<T, ID> extends
Repository<T, ID> {
  Optional<T> findById(ID id);
  <S extends T> S save(S entity);
}

interface UserRepository extends
MyBaseRepository<User, Long> {
  User findByEmailAddress(EmailAddress
emailAddress);
}
```

Spring Data - Query Methods

- CUIDADO!
- Consultas podem retornar valores *null* e o framework oferece anotações para controlar o comportamento:
  - @NonNullApi – utilizado em pacotes para declarar o comportamento padrão para parâmetros e retornos de métodos para não aceitar nem produzir *null*.
  - @NonNull – utilizado em parâmetros ou retorno de métodos que não devem ser *null*.
  - @Nullable – utilizado em parâmetros ou retorno de métodos que podem ser *null*.

Spring Data - Query Methods

```
@org.springframework.lang.NonNullApi
package com.acme;
import org.springframework.lang.Nullable;

interface UserRepository extends Repository<User, Long>
{
  User getByEmailAddress(EmailAddress emailAddress);

  @Nullable
  User findByEmailAddress(@Nullable EmailAddress
emailAdress);

  Optional<User>
findOptionalByEmailAddress(EmailAddress emailAddress);
}
```

Spring Data - Query Methods

- No exemplo do slide anterior:
  - Pacote desabilita *null* para parâmetros e retorno dos métodos.
  - Método *getByEmailAddress* gera exceções *EmptyResultDataAccessException* se a consulta não retornou dados e *IllegalArgumentException* se o parâmetro recebido for *null*.
  - Método *findByEmailAddress* returna *null* se a consulta não retornou dados e aceita *null* como parâmetro de entrada.
  - Método *findOptionalByEmailAddress* retorna objeto *Optional.empty()* quando a consulta não produz dados e lança exceção *IllegalArgumentException* quando o parâmetro for *null*.

Spring Data - Customização

- Framework permite integrar repositórios desenvolvidos diretamente sobre diferentes APIs, tal como JPA, JDBC, etc.
  - Definir um fragmento de interface com as operações desejadas.
  - Implementar a interface em uma classe concreta cujo nome deve possuir o sufixo “Impl”.
  - Definir uma interface compondo as interfaces *Repository* do framework e os fragmentos de interfaces customizadas.

Spring Data - Customização

```
interface ContactRepository {
  List<Contact> findRelatives(Contact contact);
}

public class ContactRepositoryImpl implements
ContactRepository {
  …
  public List<Contact> findRelatives(Contact contact) {
    return entityManager.createQuery("SELECT u FROM
User u WHERE u.lastname = :lastname")
    .setParameter("lastname", contact.getLastname())
    .getResultList();
  }
}

public interface UserRepository extends
CrudRepository<User, Long>, ContactRepository {}
```

Spring Data JPA

- Projeto Spring Data JPA adiciona elementos de automatização do Spring Data sobre acesso a dados via API do JPA (Jakarta Persistence API)
- O framework Spring Data JPA cria automaticamente as consultas utilizando a API JPA em função da interface escolhida para o repositório e a assinatura dos métodos definidos pelo programador ou consultas explícitas definidas pelo programador