07

Para saber mais: Conhecendo a Criteria API do Hibernate

Desde a versão 3, o Hibernate possui uma API própria para montar as pesquisas: a Criteria. A aceitação do mercado foi bem grande e por isso, na versão 2 da JPA, ela virou especificação. Porém as diferenças entre a versão do Hibernate e a versão da JPA são enormes, apenas a ideia é a mesma. Vamos traduzir a query que escrevemos (usando Critéria do JPA) para buscar os produtos com vários filtros para Critéria do Hibernate:

No método `getProdutos`, o primeiro passo será conseguir uma instância de `Session` (que é equivalente do `EntityManager` do JPA). Conseguimos isso chamando o método `unwrap` do `EntityManager`:

```
Session session = entityManager.unwrap(Session.class);
```

Com *Criteria* ao invés de criar um objeto `Query`, criamos um objeto do tipo `Criteria`, passando em qual tabela ele buscará os resultados:

```
Criteria criteria = session.createCriteria(Produto.class);
```

Este objeto criado já é suficiente para devolver todos os produtos, basta chamar o método `list`:

```
public List<Produto> listarTodos(){
    Criteria c = session.createCriteria(Produto.class);
    return c.list();
}
```

O que queremos é adicionar no nosso método, os filtros de busca. Para isso podemos usar a classe `Restrictions`:

```
if (!nome.isEmpty()) {
    criteria.add(Restrictions.like("nome", "%" + nome + "%"));
}

if (lojaId != null) {
    criteria.add(Restrictions.like("loja.id", lojaId));
}
```

E por último, a categoria. Onde precisamos fazer um `join`:

```
if (categoriaId != null) {
    criteria.setFetchMode("categorias", FetchMode.JOIN)
            .createAlias("categorias", "c")
            .add(Restrictions.like("c.id", categoriaId));
}
```

Já que no nosso projeto a JPA é gerenciada pelo Spring, ele pede que ao chamarmos o método `unwrap` que exista já um `EntityManager` ativo. Para isso, podemos anotar o método com `@Transactional` dizendo que dentro de todo esse método deverá ter uma transação ativa e portanto um `EntityManager`.

Uma alternativa veremos no capítulo seguinte, com o padrão `OpenEntityManagerInView`, pra criarmos um `EntityManager` a cada requisição.

Usando a anotação `@Transactional`, o método completo deverá ficar parecido com a implementação abaixo:

```
@Transactional
public List<Produto> getProdutos(String nome, Integer categoriaId, Integer lojaId) {
    Session session = em.unwrap(Session.class);
    Criteria criteria = session.createCriteria(Produto.class);

    if (!nome.isEmpty()) {
        criteria.add(Restrictions.like("nome", "%" + nome + "%"));
    }

    if (lojaId != null) {
        criteria.add(Restrictions.like("loja.id", lojaId));
    }

    if (categoriaId != null) {
        criteria.setFetchMode("categorias", FetchMode.JOIN)
            .createAlias("categorias", "c")
            .add(Restrictions.like("c.id", categoriaId));
    }

    return (List<Produto>) criteria.list();
}
```