Lidando com Entidades

Site: Moodle institucional da UTFPR
Curso: CETEJ34 - Sistemas De Banco De Dados - JAVA_XIX (2020_01)
Livro: Lidando com Entidades

Impresso por: Alexandre Tiago Ximenes
Data: sábado, 22 ago 2020, 06:50

Sumário

1. Lidando com Entidades

O conceito de entidades é bem simples de ser compreendido e nasceu nos bancos de dados relacionais. Todas as bases de dados são formadas por tabelas e estas, por linhas e colunas, onde cada linha é considerada uma entidade.

Em uma aplicação Java, preparada para trabalhar com um framework ORM, é preciso realizar o mapeamento objeto relacional. Este mapeamento é a forma como o framework vai referenciar uma tabela no banco de dados com a classe que a representa na aplicação.

E cada coluna da tabela vai ter uma referência na classe que será um atributo (variável de instância). Assim, cada classe da aplicação que referencia uma tabela no banco de dados é chamada de classe de entidade. Isto porque, um a classe vai ser a instância de um objeto que será persistido na tabela como uma linha, ou seja, uma entidade.

Os mapeamentos podem ser feitos de duas formas, uma delas, por meio de arquivo XML.Com o lançamento da versão 5 do Java anos a trás, a linguagem passou a oferecer o uso de anotações e elas foram adicionadas como opção de mapeamento pelos frameworks ORM e também pela especificação JPA.

Por exemplo, o Hibernate tem suas próprias anotações de mapeamento, se elas forem usadas, apenas aplicações que trabalham com o Hibernate farão uso deste mapeamento.

Já, se o mapeamento for realizado com as anotações oferecidas pela especificação JPA, qualquer framework ORM poderá fazer uso deste mapeamento. Sendo assim, é sempre mais apropriado usar o sistema de mapeamento da especificação.

1.1. Classe AbstractPersistable

Neste capítulo serão abordados dois exemplos de classes de entidades mapeadas via anotações da especificação JPA.

Há dois motivos pelos quais preciso abordar as classesde entidades, o primeiro é que precisamos delas para trabalhar com os repositórios do Spring Data.

O segundo é referente a uma classe abstrata, fornecida pelo SpringData JPA, que minimiza a quantidade de código necessária para a implementação de cada classe de entidade.

Normalmente, este recurso é desenvolvido à parte, em projetos profissionais, o que o Spring fez foi oferecer uma implementação já pronta com o objetivo de facilitar as coisas para o programador.

A classe em questão é a **AbstractPersistable** e, seu uso é por meio de herança. Trabalhando com **AbstractPersistable,** uma classe de entidade vai herdar alguns métodos como **getId() e setId(),** os quais têm o atributo **id** já mapeado:

```
@Id
@GeneratedValue
private PK id;

public PK getId() {
   return id;

protected void setid(final PK id} {
   this.id = id;
}
```

Outro método interessante é o **isNew() ,** o qual verifica se o val r do atributo id é nulo ou não, caso seja, o retorno será verdadeiro, caso contrário será falso.

Este método é útil para saber, se uma entidade já foi persistida ou não. Lembre-se que toda entidade persistida já contém um identificador.

```
@Transient
public boolean isNew() {
   return null == getId();
}
```

O **toString()** também está presente em **AbstractPersistable,** assim como, os métodos **equals()** e **hashCode()**. Estes três métodos levam em consideração apenas o atributo id da entidade.

Se em algum momento for necessário ter uma implementação diferente em algum desses métodos citados, basta sobrescrevê-los na classe de entidade que herda estas características.

O pacote onde se encontra a classe **AbstractPersistable** é o **org.springframework.data.jpa.domain**.

1.2. Classes de Entidades

Agora que já foram apresentadas as informações necessárias sobre a classe **AbstractPersistable,** será demonstrado como mapear duas entidades, as quais são base de exemplos para os repositórios que serão abordados durante este curso.

A primeira classe de entidade - **Listagem 3.1** - é a Contato, a qual possui o mapeamento para uma tabela no banco de dados chamada **Contatos.**

LISTAGEM 3 .1: CLASSE DE ENTIDADE CONTATO

```
package com.curso.entity;

import java.util.Date;
import javax.persistence.*;
import org.springframework.data.jpa.domain.AbstractPersistable;

@Entity
@Table(name = "CONTATOS")
public class Contato extends AbstractPersistable<Long> {

   @Column(name = "nome", length = 64, nullable = false)
   private String nome;

   @Column(name = "idade")
   private Integer idade;

   @Column(name = "data_cadastro")
   @Temporal(TemporalType.DATE)
   private Date dtCadastro;

   @OneToOne(cascade = CascadeType.ALL)
   @JoinColumn(name = "endereco_id", nullable = false)
   private Endereco endereco;

   @Override
   public void setId(Long id) {
      super.setId(id);
   }

// demais métodos setters e getters foram omitidos nesta listagem.

}
```

Antes de analisar o código da classeContato, é bom ressaltar que todas as anotações referentes ao mapeamento são do pacote **javax.persistence,** referente a especificação JPA.

Observe também que a entidade herda as características de **AbstractPersistable.** Por este motivo, não é necessário ter na entidade um atributo **id,** nem os métodos **toString() , equals(), hashCode()** e **getId{).**

Já o método **setId{)** foi sobrescrito, o que é opcional, por ele ser originalmente, na superclasse, um método protegido. O que torna seu acesso restrito apenas à classe filha.

Outro ponto importante é informar, por meio de um *Generics,* na declaração de **AbstractPersistable,** qual o tipo de dado que representa o **id** da entidade. Neste caso, foi definido como um **Long**.

Como dito anteriormente, se você tem um conhecimento, mesmo que básico, sobre JPA ou Hibernate, vai entender perfeitamente este m apeamento. Entretanto, vou fazer uma rápida análise sobre o que há nesta classe:

- **@Entity** - esta anotação marca a classe como uma entidade gerenciada pelo framework ORM;
- **@Table** - tem algumas finalidades como: configurar o nome da tabela que a classe está mapeando *e* a adição de índices e *consrtaints;*
- **@Column** - sua função é indicar qual coluna na tabela está sendo mapeada. Caso o nome da coluna na tabela seja idêntico ao do atributo, a anotação não é necessária. Porém, é possível também adicionar informações complementares nesta anotação como: definir o tamanho da coluna, estabelecer se aceita valores nulos, entre outras opções;
- **@Temporal** - é uma anotação complementar para definir que um atributo é do tipo temporal (data e ou hora). Na tabela, a coluna também deve ser do tipo temporal (Date; DateTime; Timestamp);
- **@0neToOne** - esta anotação tem como finalidade mapear o relacionamento 1 - 1 entre entidades;
- **@JoinColumn** - indica qual o nome da chave estrangeira no relacionamento 1 - 1.

Se Contato tem um relacionamento 1 - 1**,** chegou o momento de conhecer o outro lado deste relacionamento, que é a entidade **Endereco**.

LISTAGEM 3.2: CLASSE DE ENTIDADE ENDERECO

```java
package com.curso.entity;
import javax.persistence.*;
import org.springframework.data.jpa.domain.AbstractPersistable;

@Entity
@Table(name = "ENDERECOS")
public class Endereco extends AbstractPersistable<Long> {

   public enum TipoEndereco{
      RESIDENCIAL,  COMERCIAL
   }

   @Column(name = "tipo", length = 32, nullable = false)
   @Enumerated(EnumType.STRING)
   private TipoEndereco tipoEndereco;

   @Column(name = "logradouro", length = 64, nullable = false)
   private String logradouro;

   @Column(name = "cidade", length = 64, nullable = false)
   private String cidade;

   @Column(name = "estado", length = 2, nullable = false)
   private String estado;

   @Override
   public void setId(Long id) {
      super.setId(id);
   }

// demais métodos setters e getters foram omitidos nesta listagem.

}
```

Na classe Endereco, apresentada na **Listagem3.2,** há uma nova anotação em relação às abordadas na **Listagem3.1.** Sendo assim, vamos aprender um pouco mais sobre ela.

- **@Enumerated** - responsável pelo mapeamento de um atributo do tipo enum. Como parâmetro, esta anotação recebe o tipo de dado referente à coluna na tabela.

Agora já temos as duas classes de entidades devidamente mapeadas. Elas serão utilizadas como base para o estudo das operações de CRUD nos repositórios do Spring Data JPA.