/programandojava

Tecnologia, programação e afins

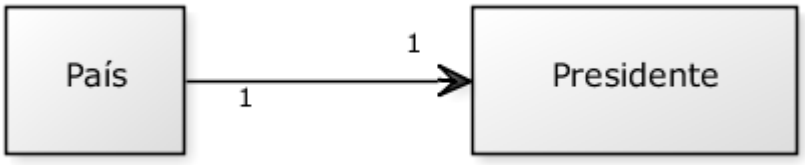
JPA 2 + Hibernate – Relacionamentos

Filed under: Frameworks by Felipe — 2 Comentários
maio de 2012
i
3 Votes

Mais um post sobre JPA 2.0 + Hibernate, neste post iremos ver os relacionamentos entre as classes.

Os relacionamentos entre as entidades devem ser expressos na modelagem através de vínculos entre as classes. Podemos definir quatro tipos de relacionamentos entre as classes, tais como:

- One to One (Um para um)



Na imagem acima temos uma relação um para um, pois um presidente só pode comandar um país e um país só pode ter um presidente.

Por ter um relacionamento entre essas entidades, deve-se usar a anotação **@OneToOne** na classe Pais, com isso o hibernate irá gerar um join column.

Por padrão o hibernate concatena um "_" com o nome da chave primária da tabela de relacionamento alvo, podemos personalizar isso usando a anotação **@JoinColum**

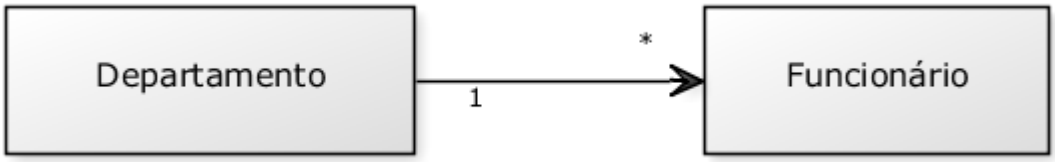
Veja as classes:

```java
// imports omitidos

@Entity
@Table(name = "PAIS")
public class Pais implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nome;

    @OneToOne
    @JoinColumn(name = "presid_id")
    private Presidente presidente;

    // getters e setters omitidos
}
```

```java
// imports omitidos

@Entity
@Table(name = "PRESIDENTE")
public class Presidente implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nome;

    // getters e setters omitidos
}
```

Caso não tivessemos utilizado da anotação **@JoinColum** a coluna da tabela referenciada teria o nome de presidente_id.

- One to Many (Um para muitos)



Vendo o relacionamento acima podemos notar que o mesmo é um para muitos, pois um departamento possui muitos funcionários e um funcionário possui apenas um departamento.

Por possuir o relacionamento um para muitos, devemos expressar este vinculo através da anotação **@OneToMany** na classe Departamento e devemos usar uma coleção do objeto alvo.

No banco de dados além das tabelas correspondentes as classes Departamento e Funcionario, um join table é criada para fazer o relacionamento de dados dos produtos com os registros das vendas. Por padrão, o hibernate concatena com "_" o nome das duas entidades. No caso, a tabela relacionada teria o nome de Departamento_Funcionario, esta tabela possuíra uma coluna chamada Departamento_id e outra chamada Funcionario_id.

Podemos personalizar o nome das colunas e o nome da tabela utilizando a anotação **@JoinTable**

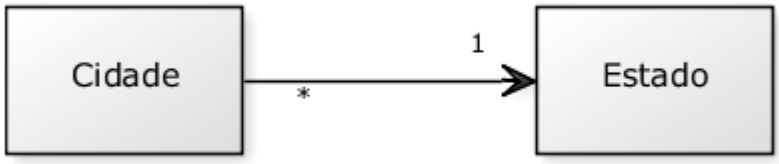
Veja as classes:

```
1  // Imports omitidos
2
3  @Entity
4  @Table(name = "DEPARTAMENTO")
5  public class Departamento implements Serializable {
6      private static final long serialVersionUID = 1L;
7
8      @Id @GeneratedValue(strategy=GenerationType.IDENTITY)
9      private Long id;
10     private String nome;
11
12     @OneToMany
13     @JoinTable(name="DPTO_FUNC",
14             joinColumns=@JoinColumn(name="dpto_id"),
15             inverseJoinColumns=@JoinColumn(name="func_id"))
16     private Collection<Funcionario> funcionarios;
17
18     // getters e setters omitidos
19 }
```

```
1  // Imports omitidos
2
3  @Entity
4  @Table(name = "FUNCIONARIO")
5  public class Funcionario implements Serializable {
6      private static final long serialVersionUID = 1L;
7
8      @Id
9      @GeneratedValue(strategy = GenerationType.IDENTITY)
10     private Long id;
11     private String nome;
12
13 }
```

Neste caso o hibernate criará uma tabela de relacionamento adicional chamada **DPTO_FUNC** contendo as colunas dpto_id (referenciando ao id da tabela departamento) e func_id (referenciando o id da tabela funcionario).

- Many to One (Muitos para um)



Podemos descrever este relacionamento vendo a imagem acima, uma cidade pertence a apenas um estado e um estado possui muitas cidades.

Como existe um relacionamento entre cidade e estado, devemos expressar este vinculo através da anotação **@ManyToOne** na classe Cidade.

No banco de dados a tabela cidade possuíra um join column que estará diretamente vinculada a tabela estado. Por padrão, o nome que o hibernate gera este join column é a concatenação com "_" do nome da entidade alvo de relacionamento e o nome da chave primaria também da entidade alvo. No exemplo de cidade e estado, o nome da coluna seria Estado_id, podemos alterar este modo de criação do hibernate utilizando a anotação **@JoinColumn**.

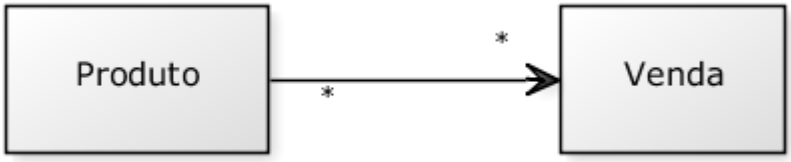
Veja as classes com os relacionamentos e mapeamentos:

```
1  // Imports omitidos
2
3  @Entity
4  @Table(name = "CIDADE")
5  public class Cidade implements Serializable {
6      private static final long serialVersionUID = 1L;
7
8      @Id
9      @GeneratedValue(strategy = GenerationType.IDENTITY)
10     private Long id;
11     private String nome;
12
13     @ManyToOne
14     @JoinColumn(name = "EST_ID")
15     private Estado estado;
16
17     // getters e setters omitidos
18 }
```

```java
// Imports omitidos.

@Entity
@Table(name = "ESTADO")
public class Estado implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nome;

    // getters e setters omitidos
}
```

Neste caso a join column gerada terá o nome de **EST_ID** e não mais de Estado_id.

- Many to Many (Muitos para muitos)



Podemos ver claramente o relacionamento muitos para muitos na imagem acima, pois um produto pode estar em muitas vendas e uma venda pode possuir muitos produtos.

Por possuir o relacionamento muitos para muitos, devemos expressar este vinculo através da anotação **@ManyToMany** na classe Produto e devemos utilizar uma coleção do objeto alvo.

No banco de dados além das tabelas correspondentes as classes Produto e Venda, um join table é criada para fazer o relacionamento de dados dos produtos com os registros das vendas. Por padrão, o hibernate concatena com "_" o nome das duas entidades. No caso, a tabela relacionada teria o nome de Produto_Venda, esta tabela possuíria uma coluna chamada Produto_id e outra chamada Venda_id.

Podemos personalizar o nome das colunas e o nome da tabela utilizando a anotação **@JoinTable**

Veja as classes:

```java
// Imports omitidos

@Entity
@Table(name = "PRODUTO")
public class Produto implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nome;
    private Double valor;

    @ManyToMany
    @JoinTable(name = "prod_vend",
            joinColumns = @JoinColumn(name = "prod_id"),
            inverseJoinColumns = @JoinColumn(name = "vend_id"))
    private Collection<Venda> vendas;

    // getters e setters omitidos
}
```

```java
// Imports omitidos

@Entity
@Table(name = "VENDA")
public class Venda implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private Double valor;

    // getters e setters omitidos
}
```

Neste caso, a join table gerada terá o nome de **prod_vend** e possuirá as colunas **prod_id** e **vend_id**

Relacionamentos Bidirecioanais

Quando temos um relacionamento entre entidades e esta é definida em apenas uma classe podemos acessar a outra entidade a partir da primeira, veja o exemplo do relacionamento entre país e presidente:

```java
// imports omitidos

@Entity
@Table(name = "PAIS")
public class Pais implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nome;

    @OneToOne
    @JoinColumn(name = "presid_id")
    private Presidente presidente;

    // getters e setters omitidos
}
```

Para acessarmos o presidente a partir de um país faríamos assim:

```java
Pais p = entityManager.find(Pais.class, 1L);
Presidente pr = p.getPresidente();
```

Para podermos acessar um País a partir de um Presidente devemos mapear este mesmo relacionamento na classe Presidente, porém usamos o atributo **mappedBy** da anotação **@OneToOne** para que o JPA não considere que seja dois relacionamentos distintos. O valor que o atributo mappedBy espera é o nome do atributo que expressa o mesmo relacionamento na outra entidade.

Veja:

```java
// imports omitidos

@Entity
@Table(name = "PAIS")
public class Pais implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nome;

    @OneToOne
    @JoinColumn(name = "presid_id")
    private Presidente presidente;

    // getters e setters omitidos<br />
}
```

```java
// imports omitidos

@Entity
@Table(name = "PRESIDENTE")
public class Presidente implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nome

    @OneToOne(mappedBy="presidente")
    private Pais pais;

    // getters e setters omitidos
}
```

Espero que vocês tenham gostado e caso tenham alguma dúvida podem perguntar nos comentários.

Até a próxima !!
Tags:hibernate, ManyToMany, ManyToOne, OneToMany, OneToOne

Comments RSS (Really Simple Syndication) feed

2 Comments:

- Jackson
  agosto de 2014 às 12:30
  Muito bem explicado. Obrigado!

  Responder
- Arnaldo Vicente
  abril de 2018 às 11:40
  Bem transmitido. Abs

  Responder