Índice

Assuntos: Res API com Spring Boot

Abertura Alunos:
Apresentação pitch
Wendy e Giovanna.

Relacionamento entre classes

Recurso tema

Então, o Spring é como um chef que ajuda os cozinheiros a preparar pratos incríveis (ou, nesse caso, sites e aplicativos). Agora, dentro desse mundo culinário do Spring, existe algo chamado "tema". Um tema seria como a decoração do prato, com cores, apresentação e estilo únicos.

Quando você está cozinhando com o Spring para fazer um prato (ou site), pode escolher um tema, assim como escolhe os ingredientes para sua receita. Por exemplo, você pode escolher um tema com cores vibrantes e elementos gráficos se quiser dar um toque festivo ao seu prato (ou site).

E o legal é que o Spring permite que você mude de tema facilmente, como se estivesse trocando a decoração de um restaurante. Se um dia você estiver entediado com o tema da festa e quiser algo mais relaxado, pode trocar para um tema com cores suaves e detalhes mais simples.

Isso facilita para os "cozinheiros" (ou programadores), porque podem criar pratos bonitos (ou sites bonitos) sem precisar mexer demais na receita (ou código).

Camadas

Camada Model
A camada model representa a lógica de negócios e o acesso aos dados. É responsável pelo gerenciamento e manipulação dos dados, assim como pelas regras de negócios da aplicação.

A camada de serviços geralmente contém operações e lógica de negócios que são utilizadas por controladores para processar solicitações do usuário. isso ajuda a manter uma separação clara entre as responsabilidades das camadas, facilitando a organização e a manutenção do código

Camada Interface
A interface refere-se a um ponto de interação entre diferentes sistemas, módulos ou componentes. Pode incluir métodos, operações ou interações visuais que permitem a comunicação e integração entre partes distintas de um sistema.

Camada Controller
O controller gerencia a lógica de interação entre a interface do usuário e os dados, facilitando a coordenação e manipulação de informações.

Annotations

Annotations são classes de uma biblioteca, elas têm métodos escondidos dentro de si, para facilitar a criação de código, usando palavras pré definidas.

Comandos

@NotNull

Annotation que não permite valor nulo, mas aceita espaço. Mais adequado para valores inteiros, float, boolean, etc

Documentação extra

@NotBlank

Annotation que não permite que um valor fique vazio, não aceita espaços como resposta. Mais adequado para String, char, etc.

@JsonIgnoreProperties (“*variável de relação de tabelas*”)

Ignora uma propriedade a seguir, usada para indicar que certas propriedades de um objeto Java devem ser ignoradas durante o processo de serialização (transformar um objeto Java em JSON) ou desserialização (transformar JSON de volta em um objeto Java).

```
33
34      @UpdateTimestamp
35      private LocalDateTime data;
36
37      @ManyToOne
38      @JsonIgnoreProperties("postagem")
39      private Tema tema;
40
41      public Long getId() {
42          return this.id;
43      }
44
```

[Documentação extra JSON]

fetch - fetchtype

```
@OneToMany(fetch = FetchType.LAZY, mappedBy = "tema", cascade = CascadeType.REMOVE)
```

Configuração de performance, melhora de performance.

FetchType.LAZY: A estratégia LAZY diz ao Hibernate para obter todas as entidades relacionadas somente quando for necessário, ou seja, ao carregarmos os dados de uma Entidade, ele não carregará os dados de todas as Entidades associadas até precisarmos destes dados. A estratégia LAZY é a estratégia padrão do Hibernate.

FetchType.EAGER: A estratégia EAGER diz ao Hibernate para obter todas as entidades relacionadas com a consulta inicial, ou seja, ao carregarmos os dados de uma Entidade, ele também carregará os dados de todas as Entidades associadas e armazenará na memória.

Exemplo: EAGER compraria a compra do mês o LAZY compraria apenas os itens necessários.

Explicação 2:

Fetch refere-se a como os dados são recuperados do banco de dados. Um fetch pode ser "lazy" (preguiçoso), onde os dados são carregados apenas quando necessários, ou "eager" (ansioso), onde todos os dados relacionados são carregados imediatamente. É como decidir se você vai buscar todos os itens de uma lista de compras apenas quando precisar deles ou se vai pegar todos de uma vez

[Documentação Extra Fetch]

CascadeType

Comando que especifica que tudo o que ocorre em relação a uma tabela, ocorrerá em outra tabela, ou seja, é uma interligação de tabelas que define um comando e onde ele ocorre.

Exemplo: Possuo o tema terror, e em outra tabela tenho 15 livros com o tema de terror, se eu apago o tema terror, eu automaticamente apago todos os livros dentro desse tema.

Uso é indicado para especificações e não base de informações.

Tipo	Descrição
Persist	Ele propaga a operação de persistir um objeto Pai para um objeto Filho, assim quando salvar a Entidade Cliente, também será salvo todas as Entidades Telefone associadas.
Merge	Ele propaga a operação de atualização de um objeto Pai para um objeto Filho, assim quando atualizadas as informações da Entidade Cliente, também será atualizado no banco de dados todas as informações das Entidades Telefone associadas.
Remove	Ele propaga a operação de remoção de um objeto Pai para um objeto Filho, assim quando remover a Entidade Cliente, também será removida todas as entidades Telefone associadas.
Refresh	Ele propaga a operação de recarregar de um objeto Pai para um objeto Filho, assim, quando houver atualização no banco de dados na Entidade Cliente, todas as entidades Telefone associadas serão recarregadas do banco de dados.
All	Corresponde a todas as operações acima (MERGE, PERSIST, REFRESH e REMOVE).
Detach	A operação de desanexação remove a entidade do contexto persistente. Quando usamos CascaseType.DETACH, a entidade filha também é removida do contexto persistente

Documentação Extra CascadeType

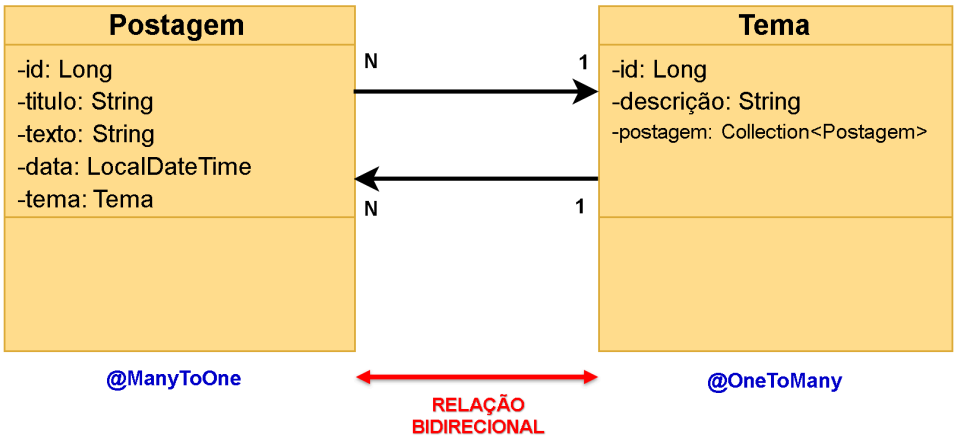
Mapeamento Objeto-Relacional (ORM)

É o processo de conversão das Classes Java em Tabelas (Entidades) no Banco de Dados da aplicação e vice-versa.

O JPA seriam comandos dentro do Spring para usar o SQL sem utilizar o SQL, com comandos pré definidos ele facilita a manipulação de informações.

O Framework responsável por implementar o ORM no Spring é o Hibernate.

A partir do Relacionamento entre as Classes, o Hibernate construirá o Relacionamento entre as Tabelas no Banco de dados Relacional.



Unilateral

Mapeamos somente uma das entidades envolvidas no relacionamento. Nestes relacionamentos, temos os conceitos de Entidade Fonte e Entidade Alvo. A diferença básica é que a Entidade Fonte possui o Mapeamento do Relacionamento.

Bilateral
As duas entidades são mapeadas e o relacionamento acontece em ambos os sentidos entre as entidades, que se comportam como se existissem dois Relacionamentos Unidirecionais, um para cada entidade envolvida.

Nos Relacionamentos Bidirecionais temos o conceito de **Entidade Proprietária** e **Entidade Inversa.**

Entidade Proprietária: A tabela desta Entidade será a Proprietária da Chave Estrangeira - Foreign Key.

Entidade Inversa: O atributo deve ser anotado e configurado com o comando mappedBy.

Em uma relação unilateral só uma classe vai receber a annotation, se for bilateral temos as annotations em ambas classes.

Se uma classe tem o Many to One a outra tem One to Many para terem um fluxo de informações.

Em comparação com as postagens(ManyToOne) e os temas(OneToMany), muitas postagens tem apenas um tema.

@ManyToOne: Indica uma relação onde muitos elementos de uma classe se relacionam com um único elemento de outra classe. Por exemplo, muitos comentários podem estar associados a um único post.

Imagine um time de futebol (Many) com vários jogadores, mas todos esses jogadores pertencem a um único treinador (One).

@OneToMany: Indica uma relação onde um elemento de uma classe se relaciona com muitos elementos de outra classe. Por exemplo, um post pode ter muitos comentários. usando um exemplo diferente pra ver se fica mais fácil de explicar.]

Agora, pense do ponto de vista do treinador (One). Ele pode ter vários times (Many) sob sua liderança, cada um com muitos jogadores. Resumindo, @ManyToOne é muitos (jogadores) para um (treinador), e @OneToMany é um (treinador) para muitos (times).

Dica: Definir a relação no DER e no MER.

Documentação extra Many to Many

Documentação extra One to Many

*Mensagem Hibernate

Para verificar se os comandos foram aceitos e criados na sua DB o Spring tem uma mensagem :

```
2023-02-20T15:07:41.788-03:00  INFO 1300 --- [  restartedMain] SQL dialect
Hibernate: alter table tb_postagens add column tema_id bigint
Hibernate: create table tb_temas (id bigint not null auto_increment, descricao varchar(255) not null
Hibernate: alter table tb_postagens add constraint FKc8fvybwie3ndsogids89ipisk foreign key (tema_id)
2023-02-20T15:07:43.980-03:00  INFO 1300 --- [  restartedMain] o.h.e.t.j.p.i.JtaPlatformInitiator
```

Links e afins

Cookbook, Spring relacionamento entre classes, recurso tema, MD13:

https://github.com/conteudoGeneration/cookbook_java_fullstack/blob/main/04_spring/13.md

Cookbook, Spring README geral:

https://github.com/conteudoGeneration/cookbook_java_fullstack/blob/main/04_spring/README.md

Cookbook, código Fonte do blog pessoal CB 13:

https://github.com/conteudoGeneration/backend_blogpessoal_v3/tree/10_Classe_Tema_Relacionamentos

Cookbook, código Fonte do blog pessoal CB 14:

https://github.com/conteudoGeneration/backend_blogpessoal_v3/tree/11_Classe_TemaController

Cookbook, Spring relacionamento entre classes, recurso tema, MD14:

https://github.com/conteudoGeneration/cookbook_java_fullstack/blob/0683f4f809ee6ec88ce79a3ae3c4ce53c276553c/04_spring/14.md

Dúvidas