02

O comportamento Lazy e OpenEntityManagerInView

Transcrição

Editando produtos

Em algum momento, provavelmente, acontecerá de precisarmos alterar os preços dos produtos de nossa loja, por exemplo, no caso de ocorrer alguma promoção. E, para isso, nossa loja permite modificar os dados de produtos na página de edição/cadastro.

Na classe `ProdutoController` temos uma *Action* (chamada de `update`) que busca o produto que possui um certo `id` e o disponibiliza à `view` através do objeto `Model` .

```
@RequestMapping(value="/{id}/form", method=RequestMethod.GET)
public String update(@PathVariable Integer id, Model model) {
    Produto produto = produtoDao.getProduto(id);

    model.addAttribute("produto", produto);
    return form(produto);
}
```

Depois disso, somos redirecionados para `produto/form.jsp` onde realizaremos todas as edições. Ok?

Comportamento Lazy

Então, vamos escolher um produto qualquer e clicar no botão `Editar` . O problema é que ao fazer isso, recebemos uma exceção do tipo `LazyInitialializationException` . Precisamos corrigir esse problema, mas antes vamos entender o que aconteceu!

Se olharmos no log da exceção, podemos ver a seguinte mensagem:

```
failed to lazily initialize a collection of role: br.com.caelum.model.Produto.categorias, could not initialize proxy - no Session
```

Percebemos que alguma coisa há de errado com o atributo `categorias` do model `Produto` . Parece que ele não conseguiu buscar os dados! Se olharmos a classe vemos que o relacionamento entre produto e categoria é *many-to-many*. Já sabemos que relacionamentos `@*ToMany` são processados de forma *lazy*. Ou seja, quando buscarmos por um produto ele não estará acompanhado com os dados das categorias. Elas serão "populadas" quando executarmos algum método da lista de categorias. Nesse momento a JPA irá disparar uma `query` em busca desse relacionamento.

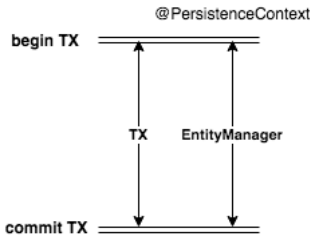
Porém, no projeto: Quem é o responsável por criar e gerenciar o *EntityManager*? Vamos olhar a classe:

```
@Repository
public class ProdutoDao {

    @PersistenceContext
    private EntityManager em;
```

```
    // código omitido

}
```

##Ciclo da vida do EntityManager

Recebemos o `EntityManager` através de Injeção de Dependência realizada pelo container do Spring. Por padrão, a `EntityManager` é criada sempre em cada transação que for executada e é encerrada ao final desta ação. Podemos ver um exemplo na figura abaixo:



Já vimos que ao executarmos a *Action* somos redirecionados para a página de formulário, no trecho de edição do produto, onde realizamos um `c:forEach` para mostrar as categorias.

```
<c:forEach items="${categoriaProduto.id}" var="categoriaProduto" varStatus="statusProduto">
...
</c:forEach>
```

Na primeira interação do `forEach` o método `iterator`, da lista de categorias do produto, é chamado. Porém, os relacionamentos `@*toMany` são `Lazy` por *default*, como já vimos. A JPA irá disparar uma query em busca dos dados das categorias daquele produto. O problema é que no momento em que ele faz isso, já não há mais um `EntityManager` ativo pra ele usar o que provoca o disparo dessa exceção.

Precisamos então, de um `EntityManager` que dure mais do que apenas uma transação, que fique aberto até o final da requisição, até ele renderizar o JSP!

##O padrão Open EntityManagerInView

Podemos utilizar um componente que intercepte a requisição e que abra o *EntityManager* no início e o feche no final. Nesse caso, qual componente da especificação de Servlets poderemos usar? Um *filtro*, pois ele permite executar algo antes do *request* e depois da resposta! E isso, é ideal para o cenário que precisamos começar, ou seja, o EntityManager iniciado antes do request e fechado depois da resposta, após mostrar o JSP na tela.

Um rascunho desse filtro seria:

```
public class EntityManagerFilter implements Filter {

    // código omitido

    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) {

        EntityManagerFactory emf = // obtem EntityManagerFactory
        EntityManager em = emf.createEntityManager();
```

```
            request.setAttribute("em", em);

            chain.doFilter(request, response);

            em.close();
        }
    }
```

Essa técnica do EntityManager ficar aberto até o final do *request* é, na verdade, um padrão! Chamado de *OpenEntityManagerInView*.

Atenção: Precisamos interceptar realmente **TODAS** as requisições? Quando realizamos uma requisição em uma página, recebemos um HTML de resposta que é exibido pelo navegador. Durante o processo de renderização, são realizadas no servidor, várias requisições em busca de arquivos estáticos utilizados na página. O problema de utilizar filtros, portanto, é que cada uma dessas requisições passará pelo filtro do SpringMVC, abrindo desnecessariamente um `EntityManager` para cada uma das requisições feitas.