11

Usando o Extended Entity Manager

Transcrição

Resolvemos o problema da Exception `LazyInitializationException` usando o `join fetch` , que consegue trazer o livro que não era carregado.

```
public Livro buscarPorId(Integer id) {
        String jpql = "select l from Livro l order by l.autores"
                        + "where l.id = :id";
        return manager.createQuery(jpql, Livro.class)
                .setFirstResult("id", id);
                .getSingleResult();

}
```

Vamos deixar o nosso código como estava antes, e ocorria a exception:

```
public Livro buscarPorId(Integer id) {
        return manager.find(Livro.class, id);
}
```

Vamos entender porque ocorria a `LazyInitializationException` . Nós fazemos o carregamento do Livro na seguinte linha:

```
<f:viewAction action="#{livroDetalheBean.carregaDetalhe()}" />
```

Então, uma conexão com o banco de dados é aberta e o `Livro` é carregado, porém os relacionamentos da **JPA** que terminam com `@..ToMany` não são carregados. Quando a página continua a ser carregada, chegamos no ponto de exibir os autores pela tag `<ui:repeat value="#{livroDetalheBean.livro.autores}" ..>` . Neste ponto, o `Hibernate` possui uma funcionalidade chamada `Lazy Initialization` que tem por propósito carregar os relacionamentos que ele não trouxe na primeira vez. O que acontece é que, ao tentar buscar esses relacionamentos a conexão com o banco de dados já foi fechada e por isso obtemos uma `LazyInitializationException` .

Além de buscar as entidades antes, como fizemos anteriormente, existem outras formas de fazer o `Lazy Initialization` funcionar. E uma das formas é mantendo a conexão aberta para que o usuário possa carregar tudo, mesmo sendo `Lazy` . A questão que fica é, quanto tempo devemos deixar a conexão aberta?

Se pensarmos um pouco, vamos perceber que tudo que precisa ser carregado, servirá para que o usuário visualize a página completa, mas depois que o usuário já tem as informações na página, não precisaremos mais de uma conexão. Ou seja, o tempo de um único ***request***. Esse é o tempo que queremos deixar a conexão aberta.

Para esse caso, precisamos fazer uso de um recurso que temos na especificação do **JPA**, que estende o contexto por mais tempo do que o normal, também conhecido como ***Extended Entity Manager***.

Para ativar o contexto estendido do *Entity Manager*, vamos abrir nosso `LivroDao`, e dentro da annotation `@PersistenceContext` vamos colocar um atributo chamado `type` com o valor `PersistenceContextType.EXTENDED`. Pela especificação do **JavaEE**, para usar o `EXTENDED`, o nosso `LivroDao` precisará ser um **EJB** do tipo *Statefull*. Essa é uma dependência da especificação.

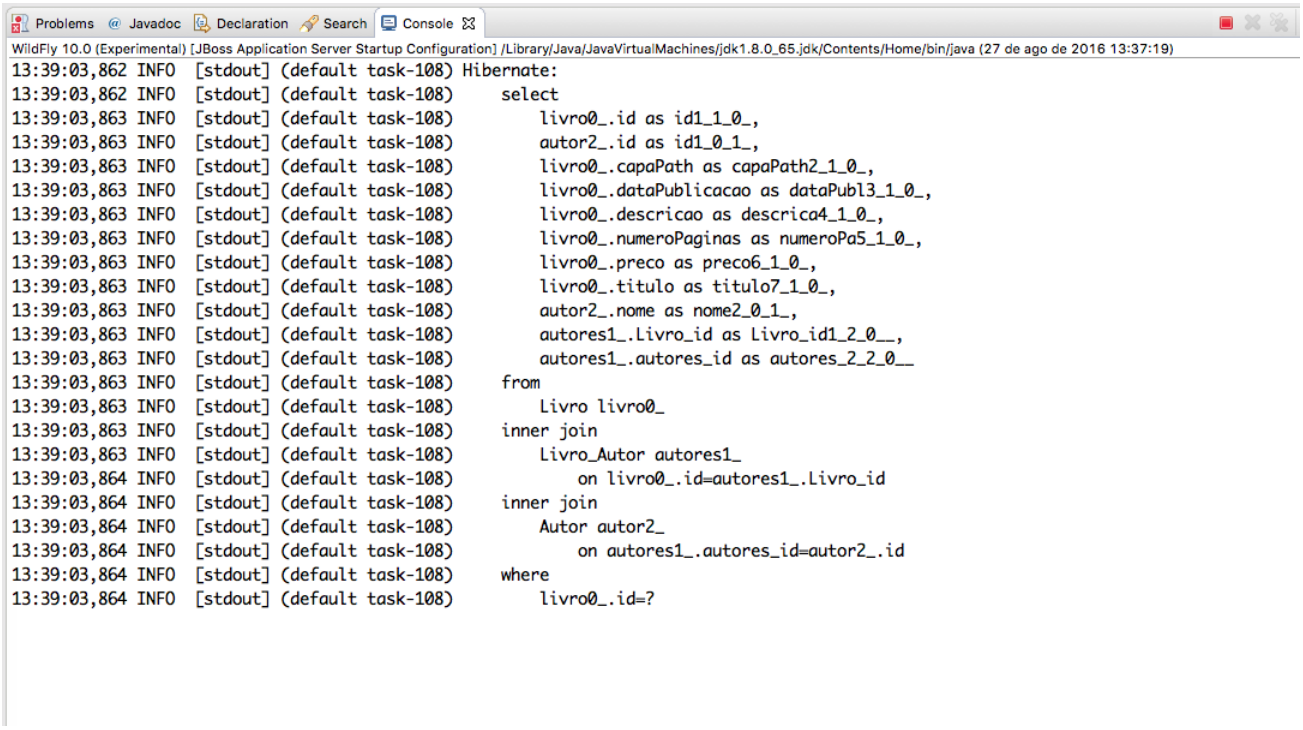
Assim, nosso `LivroDao` ficará assim:

```
@Stateful
public class LivroDao {

    @PersistenceContext(type=PersistenceContextType.EXTENDED)
    private EntityManager manager;

    // demais métodos abaixo
}
```

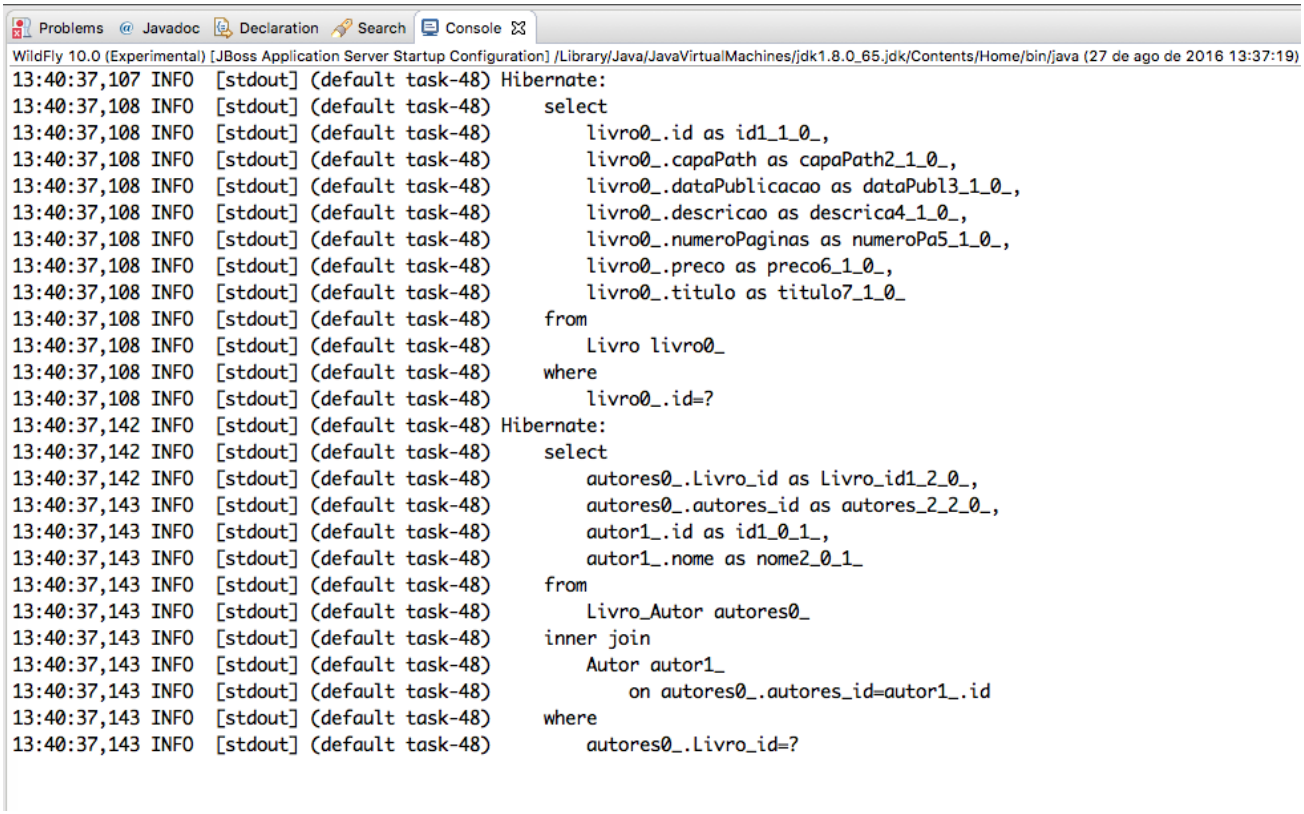
Essa annotation `@Stateful` torna nossa classe um *Enterprise Java Bean*. Os *EJB's* eram o centro das especificações **JavaEE** até a versão 6.0, quando chegou o **CDI** para ser essa camada intermediária que liga as partes do nosso sistema com o servidor.

Desde então o pessoal da **Oracle** vem tirando o poder concentrado nos **EJB's** e dividindo em pequenas outras especificações como a própria **JPA**, **CDI**, **JMS**, **JTA** e etc.

Dessa forma, estamos resolvendo o problema da conexão ser fechada, já que o incrementamos o contexto do *Entity Manager*. Essa é outra forma que temos de resolver o `LazyInitializationException`. Mas junto disso ganhamos mais uma coisa. Por exemplo, da forma como estamos fazendo, verifique no console a quantidade de selects que o **Hibernate** está fazendo para nós, e percebemos que temos apenas um *select* já com o *join* dos autores:



Já quando voltamos o código buscando apenas o `Livro` e deixando que o `LazyInitialization` procure pelos autores, o **Hibernate** realiza dois *selects*



Assim, planejar suas *queries* sempre tem um ganho a mais, nesse caso, menos consultas até o banco de dados, tornando a aplicação mais performática. Por isso é bem importante saber usar os recursos que temos disponíveis nas especificações, sabendo que **JSF** e **JPA**, apesar de ser relativamente fácil começar a programar nessas especificações, é importante saber o que está sendo feito. Não saber o correto funcionamento de uma especificação e quando usar certos recursos ou não, pode ser motivo de grandes problemas futuros.

Utilize os recursos do **JavaEE** com consciência e obtenha o máximo de produtividade no desenvolvimento do seu sistema.