18

Inserindo os Dados no Banco de Dados

Transcrição

Vamos agora voltar para o nosso `AdminLivrosBean` e nele temos o método salvar que persiste os nossos dados no banco de dados, para isso precisamos do LivroDao. Nós temos aqui a opção de inicializar o LivroDao:

```
private LivroDao dao = new LivroDao();
```

Mas desta forma nós estamos dizendo que vamos assumir a responsabilidade de instanciar todos os atributos. Vale lembrar que dentro do `LivroDao` temos o atributo `manager` do tipo `EntityManager` que é injetado pelo servidor. Devemos pedir para que o nosso `LivroDao` seja também injetado e assim o `EntityManager` será criado pelo servidor também. Quem é responsável por isso é o CDI, que cuida do contexto de injeção de dependências. Com o uso do CDI o nosso código ficará dessa forma:

```
@Inject
private LivroDao dao;
```

E o nosso método salvar da `AdminLivrosBean` :

```
public void salvar(Livro livro){
    dao.salvar(livro);
}
```

Nesse ponto, se você tentou subir a aplicação e cadastrou um *Livro*, recebeu o erro a seguir: `TransactionRequiredException` . Isso ocorre, porque estamos tentando alterar o estado do banco de dados. E se tentarmos salvar (ou alterar/remover) o nosso livro, vamos receber essa exception. Para tudo que altera o estado do banco, o servidor espera uma transação e nós não temos. Precisamos pedir essa transação para o JTA, o pedido é feito pela annotation `@Transactional` (*import* do pacote `javax.transaction.Transactional`) logo acima do método salvar.

```
@Transactional
public void salvar(Livro livro){
    // o código continua intacto aqui
}
```

Se subirmos nosso servidor agora, nós conseguiremos efetuar o cadastro, faça um select no banco para verificar se o nosso livro foi inserido corretamente. Observe também o *console* do Eclipse, se você encontra um comando `insert` na tabela *Livro*.

Um ponto interessante é que não anotamos a classe `LivroDao` e mesmo assim ele foi injetado. Isso acontece porque o **CDI** consegue descobrir quais são os objetos que podem ser injetados e onde devem ser injetados. Mas como ele descobre quais beans injetar e onde injetar? Vamos abrir o nosso arquivo `beans.xml` localizado em `src/main/webapp/WEB-INF/` , e vemos que não está defido a forma de busca, isso aconteceu devido a criação feita pelo

Forge. Isso não nos gerou problema, mas para evitar problemas futuros vamos corrigir logo. Deixando nosso arquivo conforme abaixo:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<beans xmlns="http://xmlns.jcp.org/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    bean-discovery-mode="all" version="1.1"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
        http://xmlns.jcp.org/xml/ns/javaee/beans_1_1.xsd"/>
```

Nesse arquivo, o que mudamos de importante foi o atributo `bean-discovery-mode` que não havia sido adicionado, e informamos o valor `all`, para que o CDI procure por toda aplicação, e não apenas as classes anotadas com `@Named`. E o outro ponto foi o `namespace` novo do JavaEE 7, que é `http://xmlns.jcp.org/xml/ns/javaee`, onde também informamos que queremos usar a versão **1.1** do CDI.