

Divisão de responsabilidades



46%

ATIVIDADES 7 DE 9

FÓRUM DO CURSO

VOLTAR PARA DASHBOARD





Vamos testar alguns tipos de gerenciamento de transações dentro de um servidor JEE?

1) Apenas com objetivos didáticos, abra a classe
AutorDao e anote-a com
@TransactionManagement(TransactionManagementType.CONTAINER)
para definirmos explicitamente que quem
controla nossas transações é o *container*.

@Stateless

```
@TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement(TransactionManagement
```

2) Ainda na classe AutorDao vamos deixar explicito o padrão de configuração de transações para cada método. Faça isso anotando o método salva() com

@TransactionAttribute(TransactionAttributeType.REQUIRED):

```
116.5k xp
```

3

```
@TransactionAttribute(TransactionAt public void salva(Autor autor) {
....
}
COPIAR CÓDIGO
```





FÓRUM DO CURSO

VOLTAR PARA DASHBOARD





116.5k xp

3

ATENÇÃO: Vale ressaltar que ambas configurações feitas são opcionais, visto que são os padrões adotados pelo *container*. Então, ao republicar e testar a aplicação, tudo deve continuar funcionando normalmente.

3) Podemos entender melhor o controle de transações implementando a divisão de responsabilidades proposta no vídeo do capítulo. Comece substituindo o TransactionAttributeType.REQUIRED por TransactionAttributeType.MANDATORY), assim, o container irá verificar se há uma transação aberta, caso contrário, lançará uma exception do tipo

EJBTransactionRequiredException . Teste isso reiniciando o servidor após a alteração e tentando adicionar um novo autor (olhe no console do eclipse).

- 4) Agora, vamos colocar a responsabilidade de criar a transação em um outro EJB que não seja o DAO, visto que os DAOs não são o melhor lugar para isso. Crie uma classe, no pacote br.com.caelum.livraria.bean, chamada AutorService que ficará entre o AutorBean e AutorDao. Transforme essa classe em um EJB Stateless e injete o EJB AutorDao nela.
- 5) Crie em AutorService um método adiciona() que recebe um **Autor** e delega para o DAO a tarefa de salvar, e um outro método chamado todosAutores() que também fará apenas a delegação para o DAO.

@Stateless





FÓRUM DO CURSO

VOLTAR PARA DASHBOARD





116.5k xp

3

```
public class AutorService {
    @Inject
    AutorDao dao;

    public void adiciona(Autor
autor) {
        this.dao.salva(autor);
    }

    public List<Autor>
todosAutores() {
        return
this.dao.todosAutores();
    }
}
```

6) Precisamos agora refatorar a classe
AutorBean para utilizar o novo EJB de serviços
AutorService . Sendo assim, ao invés de
injetarmos AutorDao , passaremos a injetar
AutorService . Não podemos esquecer de
refatorar os métodos que usavam o atributo
dao para passar a usar o atributo service :





FÓRUM DO CURSO

VOLTAR PARA DASHBOARD





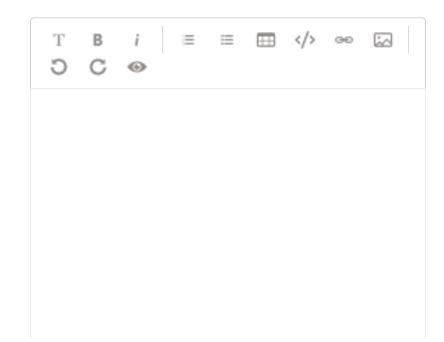
```
this.service.adiciona(autor);
          this.autor = new
Autor();
    }

    public List<Autor>
getAutores() {
          // return
this.dao.todosAutores();

    return
this.service.todosAutores();
}
}
```

7) Faça o *Full Publish* da aplicação, adicione um novo autor e veja se a *exception* do teste anterior será novamente lançada. O que você acha que aconteceu?

Responda



116.5k xp



EJB: Aula 4 -	Atividade 7	Divisão o	de respon	sabilidades	
---------------	-------------	-----------	-----------	-------------	----------





FÓRUM DO CURSO

VOLTAR PARA DASHBOARD







3