

29%

ATIVIDADES

1 de 10

FÓRUM DO CURSO

VOLTAR PARA DASHBOARD

MODO NOTURNO

ABRIR CADERNO

F

115.2k xp

a

01

Integração do JPA com Pool e DataSource

PRÓXIMA ATIVIDADE

Transcrição

Durante o treinamento usaremos alguns arquivos de configuração. Se você não baixou ainda, todos os arquivos estão disponíveis aqui: [resources.zip](https://s3.amazonaws.com/caelum-online-public/ejb/resources.zip) (<https://s3.amazonaws.com/caelum-online-public/ejb/resources.zip>).

Revisão

No último capítulo vimos como funciona o ciclo de vida dos *Session Beans*. Aprendemos a configurar um método *callback* de criação através da anotação `@PostConstruct` , além de configurar o *pool* de objetos pelo servidor JBoss AS. Vimos que *Session Beans* são automaticamente *thread safe*, e todo acesso é sincronizado.

Nos exercícios, deixamos todos os DAOs como *Session Bean Stateless* para em seguida injetá-los nos *beans*.

Na classe Banco usamos um *Session Bean* especial, o *Singleton Bean*. Ele é especial pois garante automaticamente que existirá apenas uma instância deste bean.

Todos os *beans* foram injetados pelo *EJB Container* através da anotação `@Inject` .

Nesse capítulo vamos focar na persistência, ou seja, na integração do JPA com o EJB. O foco então é o uso do JPA dentro do *EJB Container*. Para aqueles que gostariam de aprender o JPA mais a fundo, aconselhamos assistir ao treinamento específico de JPA 2 na plataforma Alura.

Injetando o EntityManager

Até agora usamos a classe Banco para simular um banco de dados, mas chegou a hora de realmente persistir os dados. O primeiro passo é injetar o `EntityManager` ; interface principal da especificação JPA. Para tanto, onde usávamos Banco, passaremos a usar o `EntityManager` .

O `EntityManager` possui métodos de alto nível para trabalharmos com objetos. Para salvar o autor podemos usar o método `persist()` :





29%

ATIVIDADES

1 de 10

FÓRUM DO

CURSO

VOLTAR

PARA

DASHBOARD

MODO

NOTURNO

ABRIR

CADERNO

`manager.persist(autor);`

COPIAR CÓDIGO

Para listar todos os autores, basta executar uma *query*:

`manager.createQuery("select a from Autor a", Autor.class)`

COPIAR CÓDIGO

Por último, podemos procurar um autor pelo id:

`manager.find(Autor.class, autorId);`

COPIAR CÓDIGO

Pronto, a classe `AutorDao` já está compilando, agora só falta ajustar a anotação de injeção de dependência. Quando injetamos um `EntityManager` não podemos utilizar a anotação `@Inject`. Nesse caso, o *Contexts and Dependency Injection* (CDI), outra especificação com o foco na injeção de dependência, buscaria o `EntityManager`. No entanto não encontraria o objeto e causaria uma exceção. Como o *EJB Container* administrará o JPA, é preciso usar uma anotação específica do mundo EJB, nesse caso `@PersistenceContext`:

`@PersistenceContext`
`EntityManager manager;`

COPIAR CÓDIGO

Isso fará com que o *EJB Container* injete o `EntityManager`. Mas qual banco de dados vamos utilizar e qual é o endereço desse banco? Para tudo isso realmente funcionar, temos que definir algumas configurações sobre o banco de dados.

Configuração do banco de dados

O primeiro passo é copiar o arquivo `persistence.xml` que faz parte do JPA. Já preparamos o arquivo para você e está disponível dentro dos *resources*, basta copiar a pasta META-INF para a pasta src do projeto *livraria*.

O arquivo `persistence.xml` possui algumas configurações específicas do mundo JPA como, o nome da unidade da persistência, o provedor de persistência e as entidades do projeto - todas elas explicadas no treinamento JPA 2 da plataforma Alura.

Também há algumas propriedades sobre a conexão com o banco de dados, usuário, senha e o *driver connector* utilizadas. O problema é que não devemos





29%

ATIVIDADES

1 de 10

FÓRUM DO

CURSO

VOLTAR

PARA

DASHBOARD



MODO NOTURNO



ABRIR CADERN

configurar os dados da conexão dentro do `persistence.xml` . Quem é responsável por fornecer a conexão é o *EJB Container*! É um serviço que o servidor disponibilizará para a aplicação.

A única coisa que deve ser feita dentro do `persistence.xml` é configurar o endereço do serviço. Para isso, existe a configuração `<jta-data-source>` . Vamos deixar o endereço ainda com interrogações para entender como configura-lo primeiro.

Usando o datasource

Como já falamos antes, é responsabilidade do servidor fornecer a conexão com o banco de dados. Uma conexão é feita através de um *driver connector*, por isso precisamos registrar o *driver* do banco MySQL como módulo no JBoss AS.

Dentro da pasta `resources` nos *downloads* já temos o módulo preparado, que consiste de um arquivo XML e um JAR do *connector*. Esses dois arquivos devem ser copiados para a pasta `modules` do JBoss AS.

Internamente o JBoss AS organiza seus módulos em pacotes, por isso devemos navegar para a pasta `modules/com` . Dentro da pasta `com` criaremos uma nova pasta `mysql` e dentro dela uma pasta `main` . Dentro da pasta `main` colocaremos o arquivo XML e o JAR (hierarquia final das pastas: `jboss/modules/com/mysql/main`).

Ao iniciar o JBoss AS, ele já carregará o novo módulo. Agora falta dizer ao JBoss AS que esse módulo representa um *driver connector*. Isso é feito no arquivo de configurações `standalone.xml` .

Vamos abrir o XML dentro de um editor de texto qualquer e procurar pelo elemento `<drivers>` . Dentro desse elemento vamos copiar a configuração do *driver* que já está disponível na pasta `resources`.

```
<driver name="com.mysql" module="com.mysql">
  <xa-datasource-class>
    com.mysql.jdbc.jdbc2.optional.MysqlXADataSource
  </xa-datasource-class>
</driver>
```

COPIAR CÓDIGO

A configuração do *driver* refere-se ao módulo definido anteriormente e fornece um nome para esse *driver*, além de especificar o nome da classe.

Por último, falta configurar o componente que no JavaEE chamamos de *DataSource*. Em uma aplicação mais robusta, é boa prática utilizar um *pool* de conexões. Cabe ao *pool* gerenciar e verificar as conexões disponíveis. Como





29%

ATIVIDADES

1 de 10

FÓRUM DO

CURSO

VOLTAR

PARA

DASHBOARD



MODO NOTURNO



ABRIR CADERNO

existem várias implementações de *pool* no mercado, o JavaEE define um padrão que se chama *DataSource*. Podemos dizer de maneira simplificada que um *DataSource* é a interface do *pool* de conexões.

Podemos ver no arquivo XML que até já existe um `datasource` dentro do JBoss AS. Nele podemos ver o min e max de conexões definidos, além do nome do *driver* responsável e os dados sobre o usuário e senha do banco.

Agora só precisamos definir o nosso próprio `datasource` . Isso também já está preparado dentro da pasta `resources`. Basta copiar e colar a definição do `datasource` para o arquivo `standalone.xml` .

```
<datasource jndi-name="java:/livrariaDS" pool-name="livrariaDS"
            enabled="true" use-java-context="true">
    <connection-url>jdbc:mysql://localhost:3306/livraria</connection-url>
    <driver>com.mysql</driver>
    <pool>
        <min-pool-size>10</min-pool-size>
        <max-pool-size>100</max-pool-size>
        <prefill>true</prefill>
    </pool>
    <security>
        <user-name>root</user-name>
        <password></password>
    </security>
</datasource>
```

COPIAR CÓDIGO


Repare que aquelas configurações do `persistence.xml` estão dentro do `datasource` agora. O servidor JBoss AS então criará o *pool* de conexões disponibilizando-o para as aplicações. A única coisa que as aplicações precisam saber é o endereço do serviço. Em nosso caso o endereço é `java:/livrariaDS` .

Vamos copiar e colar este endereço no `persistence.xml` . Pronto, a única informação que a aplicação precisa saber agora é que está acessando um `datasource` que se chama `livrariaDS` . Os detalhes da configuração estão totalmente desacoplados da aplicação.


Preparação do banco de dados

Vamos reiniciar o servidor e ficaremos atentos à saída no console para perceber possíveis problemas de configuração.

Para nossa surpresa o JBoss AS jogou uma exceção. A mensagem *Unkown Database* indica que o MySQL não conhece o banco `livraria` . Esquecemos de



115.2k xp





29%

ATIVIDADES
1 de 10

FÓRUM DO
CURSO

VOLTAR
PARA
DASHBOARD

MODOS
NOTURNO

ABRIR
CADERNO

preparar o MySQL.

Para resolver o problema vamos abrir um terminal e abrir uma conexão com o MySQL. Em nosso caso basta digitar:

```
mysql -u root
```

COPIAR CÓDIGO

Uma vez estabelecida a conexão do terminal com MySQL podemos criar e testar o banco:

```
create database livraria;  
use livraria;  
show tables;
```

COPIAR CÓDIGO

Como acabamos de criar o banco, ainda não há nenhuma tabela. Voltando ao Eclipse, vamos novamente iniciar o JBoss AS.

Dessa vez o servidor iniciou sem problemas. Até podemos observar no console que as *tables* foram criadas no banco.

Com o terminal ainda aberto testaremos rapidamente se as tabelas realmente existem. Basta repetir o comando `show tables` . O terminal mostrará as tabelas corretamente.


Testando a persistência

Chegou a hora de testar a aplicação pela interface web.

No navegador, após o *login*, podemos ver que o *combobox* dos autores está vazio. Isso faz sentido pois não cadastramos ainda nenhum autor. Vamos verificar o cadastro de autores e inserir alguns autores.

Agora, no *combobox* aparecem corretamente os autores que indicam a execução sem problemas. Vamos verificar o console, nele aparece o SQL gerado pelo JPA.

A próxima tarefa é alterar o `UsuarioDao` , que ainda usa a classe antiga. Mas isso ficará para os exercícios.



115.2k xp

alura