01 Introdução aos Enterprise Java Beans

Transcrição

Nessa vídeo-aula começaremos a desenvolver uma aplicação Java com EJB. Já temos um ambiente de desenvolvimento pré-configurado baseado no Java SE 7 (http://www.oracle.com/technetwork/java/javase/downloads/index.html) e utilizaremos como IDE (explicado abaixo) o Eclipse IDE for Java EE Developers (http://www.eclipse.org/downloads/).

Além disso, usaremos o servidor de aplicação JBoss AS 7 da Red Hat (explicado abaixo). Você pode baixar ele diretamente no site nesse link: JBoss AS 7 (http://jbossas.jboss.org/downloads)

Nesse capítulo usaremos um projeto já preparado para não começar do zero e usar uma interface gráfica. Esse projeto *livraria* pode ser baixado aqui (https://s3.amazonaws.com/caelum-online-public/ejb/livraria.zip)

Durante do treinamento usaremos alguns arquivos de configuração, todos disponíveis aqui: resources.zip (https://s3.amazonaws.com/caelum-online-public/ejb/resources.zip)

Introdução ao EJB

Hoje em dia, a grande maioria das aplicações são desenvolvidas para executar na web. Ou seja, usamos um navegador para acessar o servidor através do protocolo HTTP. Para fazer isso funcionar basta termos um servidor como o Apache Tomcat, bastante utilizado em outros treinamentos no Alura. Com ele podemos executar uma aplicação feita com *JavaServer Faces* (JSF) ou outros *frameworks MVC* (*Model-View-Controller*).

A maioria das aplicações utilizam um banco de dados como o MySQL ou Oracle, entre várias outras opções do mercado. Nesse caso, a nossa aplicação deve se preocupar em gerenciar as conexões com o banco, o que normalmente é feito através de um *pools* de conexões. A escolha e configuração correta do *Pool* é de grande importância para qualquer aplicação e afeta diretamente o desempenho e escalabilidade.

Para *persistirmos* e acessarmos dados usando o paradigma orientado a objetos podemos usar *frameworks* de Mapeamento-Objetos-Relacional (MOR) como o Hibernate ou EclipseLink, que seguem a especificação JPA (Java Persistence API). A integração do *framework* deve ser feita da melhor maneira possível para evitar desperdício de recursos e mau uso do banco de dados.

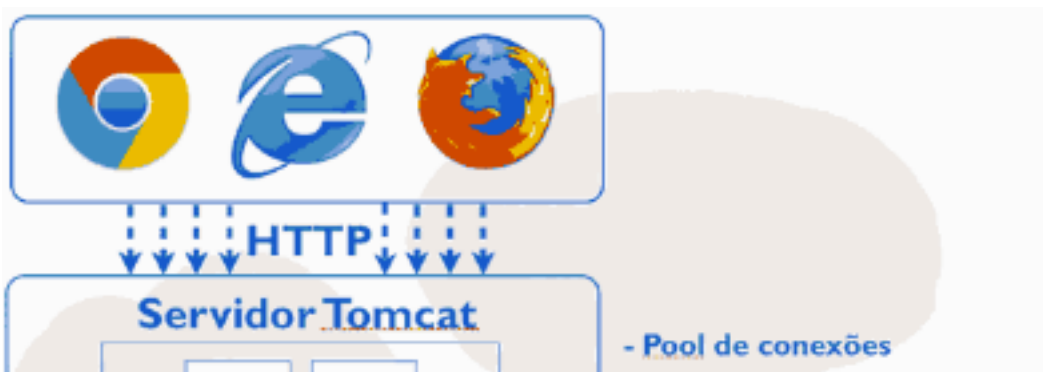
Alteração de dados no banco mesmo com JPA envolve transações que precisam ser gerenciadas, tarefa difícil de se fazer de maneira robusta. O mau gerenciamento das transações é um problema comum nas aplicações e pode causar problemas nas consistência dos dados.

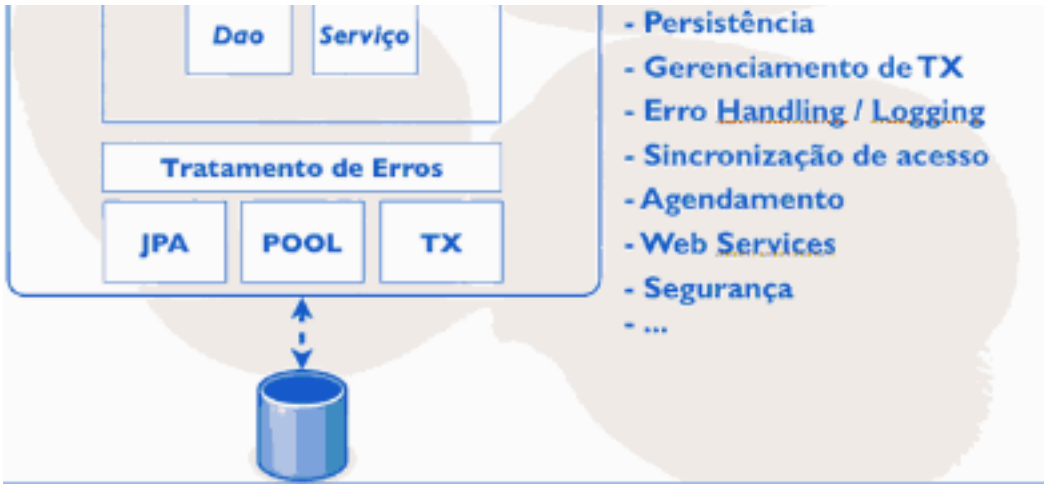
Ao trabalhar com todos esses recursos, erros ou exceções podem aparecer. Ou seja, sempre devemos ter uma estratégia para lidar com as possíveis exceções que a aplicação pode causar.

As classes da aplicação, como os *Data Access Objects* (DAOs) e as classes de serviços, serão utilizadas quando ocorrer uma requisição HTTP. A pergunta é: como podemos garantir que não há problemas de acesso e sincronização quando a quantidade de requisições crescer? Não é raro ver aplicações que começam a gerar problemas quando a demanda cresce.

Outra tarefa comum é executar tarefas periodicamente. Há aplicações que precisam agendar a execução de processos. Por exemplo, pode ser necessário enviar um email cada dia, ou verificar uma tabela no banco de dados a cada hora. O agendamento correto, no tempo exato é essencial para várias aplicações, e não é algo fácil de se implementar.

Durante o desenvolvimento de uma aplicação surgem várias outras preocupações, tais como o uso de Web Services ou mesmo a segurança da aplicação, ambos importantíssimos. Em geral, todas essas preocupações listadas são agnósticas às regras de negócio. É algo que faz parte do desenvolvimento, mas não deveriam ser a preocupação principal.





A tarefa de um servidor de aplicações é justamente livrar o desenvolvedor dessas preocupações e fornecer uma infra-estrutura pronta para que ele possa utilizar. Ou seja, não é a aplicação que vai gerenciar a transação ou se preocupar com o agendamento de tarefas. Vamos inverter o controle e deixar o servidor de aplicações fazer essa parte.

Por isso, essas preocupações também se chamam serviços do *container* ou serviços do servidor. Transação, persistência, etc. são serviços que o servidor de aplicações fornece.

Por exemplo, a classe DAO da minha aplicação deve ter acesso ao JPA sem se preocupar em como inicializa-lo. Para isso funcionar o *Enterprise Java Beans* (EJB) fornece o componente (Entity Bean), que é responsável pelo controle de transações de persistência de dados. O próprio DAO vai ser um EJB e assim poderá utilizar a JPA sem problemas.

Em outras palavras, é através dos EJBs que temos acesso aos serviços que o servidor oferece sem nos preocuparmos em como cada um deles foi inicializado. Então, para usar os EJBs, sempre precisamos de um servidor de aplicações.

Falta saber qual servidor de aplicação usaremos no treinamento. O Apache Tomcat não serve, pois não é um servidor de aplicações completo. Contudo há outras opções como RedHat JBoss AS, Oracle Glassfish, Apache Geronimo ou Oracle WebLogic, entre outros. Usaremos o JBoss AS, o servidor Java EE mais popular, *opensource* e totalmente gratuito.

Preparação do ambiente de desenvolvimento

1) Vamos então baixar o JBoss Application Server para começar a usar os EJBs. Para tal, acesse o site (http://jbossas.jboss.org/)http://jbossas.jboss.org/ (http://jbossas.jboss.org/);

2) Na página principal escolha a opção Downloads (http://jbossas.jboss.org/downloads), depois selecione a versão `JBoss AS 7.1.1.Final` no formato ZIP (http://download.jboss.org/jbossas/7.1/jboss-as-7.1.1.Final/jboss-as-7.1.1.Final.zip).

3) Além disso, vamos utilizar o Eclipse como IDE de desenvolvimento disponível em: www.eclipse.org/downloads/ (http://www.eclipse.org/downloads/)

Importante é baixar a versão Eclipse IDE for Java EE Developers que já vem com vários *plugins* para o desenvolvimento web. É preciso selecionar o Eclipse para o seu sistema operacional, já que o Eclipse depende dele.

No nosso caso já temos JBoss AS e Eclipse baixados, só falta extrair os arquivos.

4) Enquanto estão sendo extraídos os arquivos, vamos aproveitar o tempo e verificar a versão da *Java Virtual Machine* (JVM) instalada em nosso computador. Abrimos um terminal e ao executar o comando `java -version` aparece corretamente a verão 1.7 da JVM da Oracle.

Instalação do Server Adapter

Hora de iniciar o Eclipse... Ao abrir, o Eclipse pergunta qual pasta ou *workspace* queremos usar para os nossos projetos, mas também nos sugere uma pasta padrão. É essa que vamos utilizar. Portanto só é preciso confirmar.

Vamos fechar a tela inicial do Eclipse. O próximo passo é configurar o JBoss AS como servidor dentro do Eclipse. Para isso existe a aba Servers, onde configuraremos o JBoss. Já baixamos e extraímos o JBoss antes, então vamos criar um novo *Server* dentro do Eclipse.

Aqui nós temos um problema. É que na lista padrão de servidores disponíveis não aparece a versão do JBoss AS que baixamos. Só existe JBoss até a versão 5.

Para podermos configurar o JBoss 7 é preciso baixar um novo *server adapter* pelo link acima. O Eclipse atualiza a tela com os *server adapters* disponíveis e logo veremos o JBoss AS Tools. É isso que estamos procurando.

Basta selecionar e confirmar na próxima tela para concluir a instalação. A instalação pode demorar um pouco, pois vai baixar

vários arquivos. Ao final dela, vamos reiniciar o Eclipse para aplicar as alterações.

Agora, vamos repetir o processo novamente, selecionar na aba *Servers* o link para criar um novo servidor. Agora aparece na lista de servidores disponíveis o JBoss AS 7.1.

A configuração é simples, basta encontrar e selecionar a pasta do servidor JBoss, ou seja, aquela pasta que extraímos antes.

No nosso caso, também vou escolher a versão 1.7 da JVM que roda o JBoss. Pronto, só falta finalizar. Na aba *Servers* aparece o JBoss com alguns arquivos de configuração. Para iniciar clique no botão verde no lado esquerdo do Eclipse.

Ao inicializar aparece na view console do Eclipse a saída do JBoss. Dando uma olhada no console, vemos que bem no final nos foi informado a interface web do JBoss.

Vamos testar no navegador digitando localhost:9990 (http://localhost:9990).

A tela do JBoss AS indica que conseguimos iniciar o servidor!

Primeira aplicação com EJB

Vamos começar a usar os EJBs, mas para não começar do zero, preparamos um projeto que está disponível nos downloads. Vamos importar esse projeto no Eclipse.

No menu File -> Import escolhemos General -> Existing Project Into Workspace e depois Archive file para selecionar o arquivo livraria.zip (https://s3.amazonaws.com/caelum-online-public/ejb/livraria.zip).

Esse projeto nada mais é do que um *Dynamic Web Project*. Nele já foram criadas algumas classes e a interface web, mas não há nada especifico do EJB ainda. Criamos este projeto apenas por fins didáticos.

Ao importá-lo, verifique se todas as classes estão compilando. Veja que no projeto existe um problema, pois as biblioteca do JBoss ainda não fazem parte do projeto web.

Vamos configurar isso: botão direito no projeto livraria, depois escolha Java Build Path. Na aba Libraries aperte o botão Add Library, escolha Server Runtime e o JBoss 7. Através dessa configuração as bibliotecas no JBoss fazem parte do classpath.

Falta ainda associar o projeto com JBoss. Na aba Servers, botão direito Add and Remove..., e escolha o projeto livraria. Ao confirmar o JBoss carrega a aplicação. Podemos ver no console a saída: Deployed "livraria.war" .

Apresentação do projeto

A aplicação livraria usa JSF e Primefaces para definir a interface. Há um outro treinamento no Alura que ensina essas duas tecnologias caso esteja com dúvidas ou queira aprender mais.

Vamos testar a aplicação acessando no navegador:

http://localhost:8080/livraria/login.xhtml (http://localhost:8080/livraria/login.xhtml)

Há uma página de login: o login é *admin* a senha é *pass* . Após efetuado o login somos redirecionados para a página principal de aplicação. Trata-se de um cadastro de livros e autores, com abas para cada funcionalidade além do logout.

Uma coisa que podemos observar na página é que os acentos estão errados. Pode haver vários motivos por este problema, mas no nosso caso basta redefinir a codificação do projeto. No Eclipse, nas propriedades do projeto, basta selecionar UTF-8 no item *Resource*.

Vamos reiniciar o servidor JBoss e recarregar a aplicação para publicar a mudança. Após login, podemos ver que os acentos estão certos.

Falta testar uma vez a interface. Vamos cadastrar um autor apenas digitando seu nome... pronto. E depois verificar a existência do autor no combobox do cadastro de livros. Apareceu ... vamos cadastrar também um novo livro. Foi inserido com sucesso.

É importante mencionar que todos os dados ficam em memória, não há persistência por enquanto. Ou seja, ao reiniciar o servidor perderemos os dados inseridos anteriormente. Veremos como integrar o JPA com EJB mais para frente.

Vamos dar uma olhada no código fonte. Há 4 pacotes no projeto. O *modelo* possui as classes do domínio como `Livro`, `Autor` e `Usuario`. Para cada modelo existe um DAO para persistir os dados, mas como não usamos um banco de dados ainda, simulamos através da classe `Banco` um banco de dados em memória. Nessa classe também se encontram os livros e autores já cadastrados. Cada DAO, por sua vez, usa a classe `Banco`.

O primeiro Session Bean

Como falamos, ao usar EJB, teremos acesso aos serviços do servidor de aplicação, como transação, persistência com JPA ou tratamento de erro. Para transformar a classe *AutorDao* em um EJB basta uma configuração simples. Só precisamos anotá-la com *@Stateless*:

```
@Stateless
public class AutorDao{
```

COPIAR CÓDIGO

Vamos republicar a aplicação e analisar o console para realmente ver que a anotação causou uma mudança. Selecione a aplicação na aba *Servers*, usando *Full publish*.

Repare na saída algumas informações sobre a classe `AutorDao`. Ao subir, o servidor - para ser mais correto, o *EJB Container* - achou aquela anotação `@Stateless` e registrou esse EJB dentro de um registro disponível no servidor. Aquele registro se chama JNDI e o que estamos vendo na saída é o endereço do EJB nesse

registro JNDI. O servidor usa por baixo dos panos esse registro JNDI para organizar os componentes que ele administra.

Voltando ao Eclipse, vamos também configurar os outros DAOs como EJB. Abra a classe `LivroDao` e use novamente a anotação `@Stateless`. Faça o mesmo para a classe `UsuarioDao`.

```
@Stateless
public class LivroDao{
```

COPIAR CÓDIGO

```
@Stateless
public class UsuarioDao{
```

COPIAR CÓDIGO

Ao subir o servidor devem aparecer no console os endereços desses EJBs também.

Injeção de dependências

Na nossa aplicação, os DAOs são utilizados através das classes que ficam dentro do pacote *bean*. Abra a classe `AutorBean`. Essa classe é utilizada através da interface JSF, ela é chamada pelos componentes JSF definidos no arquivo `autor.xhtml`.

Para saber mais sobre o JSF também assista ao treinamento disponível no Alura.

Na classe `AutorBean`, podemos ver que estamos usando a classe

`AutorDao` para gravar e listar autores. Repare também que estamos instanciando a classe `AutorDao`:

```
public class AutorBean {

  private AutorDao dao = new
AutorDao();//criação do DAO

  //outros métodos e atributos omitidos
}
```

COPIAR CÓDIGO

É justamente essa a linha que precisa ser alterada. Ao usar EJB, não podemos mais instanciar o `AutorDao` na mão. Estamos assumindo o controle ao criar o DAO naquela linha. Nesse caso não estamos usando o `AutorDao` como um EJB.

O DAO está sendo administrado pelo EJB Container. Portanto, quem cria o DAO é o EJB Container e não a minha classe. Consequentemente precisamos pedir ao EJB Container passar aquela instancia que ele está administrando. Felizmente, isso é fácil de fazer, basta usar a anotação `@Inject`:

```
public class AutorBean {

  @Inject
  private AutorDao dao; //sem new

  //outros métodos e atributos omitidos
}
```

Pronto, o EJB será injetado! Essa parte da inversão de controle também é chamado *Injeção de dependências*. O DAO é uma dependência que será injetada pelo container.

COPIAR CÓDIGO

Faremos a mesma coisa na classe `LivroBean`. Mas além do `AutorDao` ela usa também o `LivroDao`. Vamos tirar o "new" e usar a anotação `@Inject` em cada atributo:

```
public class LivroBean {

  @Inject
  private AutorDao autorDao; //sem new

  @Inject
  private LivroDao livroDao; //sem new

//outros métodos e atributos omitidos
}
```

COPIAR CÓDIGO

Está tudo pronto para testar. Vamos publicar as alterações e acessar a aplicação pela interface gráfica. Nada mudou ao carregar a página e efetuar o login, ou seja, a aplicação continua funcionando. Já estamos usando EJB! Ainda é pouco vantajoso o uso do EJB, mas, como já falamos, temos agora acesso aos vários serviços do servidor. Veremos nos próximos capítulos como aproveitar a infra-estrutura pronta do servidor usando EJBs.