03

Integração com EJB

Transcrição

Até agora, trabalhamos com a especificações mais importantes do Java EE mas ainda não experimentamos rodar esse projeto em um servidor de aplicação. O que faremos agora é utilizar o **JBoss WildFly** para rodar nosso projeto. O que será que iremos precisar alterar no projeto? Vamos continuar usando JSF e o CDI, mas em um servidor de aplicação quem gerencia a JPA e as transações é um *container*, mais sofisticado que o CDI, chamado **EJB**. Então no nosso código, pra injetar o `EntityManager` vamos utilizar o *EJB container* que fará a injeção para nós. Contudo, não mexeremos em nada de JSF e CDI. Então vamos baixar o projeto preparado para rodar no *container* Java EE.

https://s3.amazonaws.com/caelum-online-public/jsf-cdi/stages/livraria-maven-wildfly-completo.zip (https://s3.amazonaws.com/caelum-online-public/jsf-cdi/stages/livraria-maven-wildfly-completo.zip)

Quando importamos o projeto pode demorar um pouco, porque o Maven vai baixar e compilar nosso projeto. Olhando nosso `pom.xml` vemos as dependências do projeto. A novidade é que temos uma dependência principal relacionada com Java EE, a `java-ee-api` .

```
<dependency>
    <groupId>javax</groupId>
    <artifactId>javaee-api</artifactId>
    <version>7.0</version>
</dependency>
```

Veja que seu escopo é `provided` . Isso significa que a dependência fará parte do projeto apenas na hora de compilar (e não no WAR), porque ela será fornecida pelo servidor de aplicação. JSF, CDI, JPA e toda especificação JavaEE será suprimida pelo nosso servidor de aplicação WildFly.

Baixando o JBoss Wildfly

Precisamos baixar o servidor de aplicação em http://www.wildfly.org/downloads (http://www.wildfly.org/downloads). A versão 10 é a que será usada, mas versões anteriores (8 e 9) a esta também devem ser compatíveis. Com o arquivo baixado, vamos descompactá-lo e em seguida associá-lo ao Eclipse.

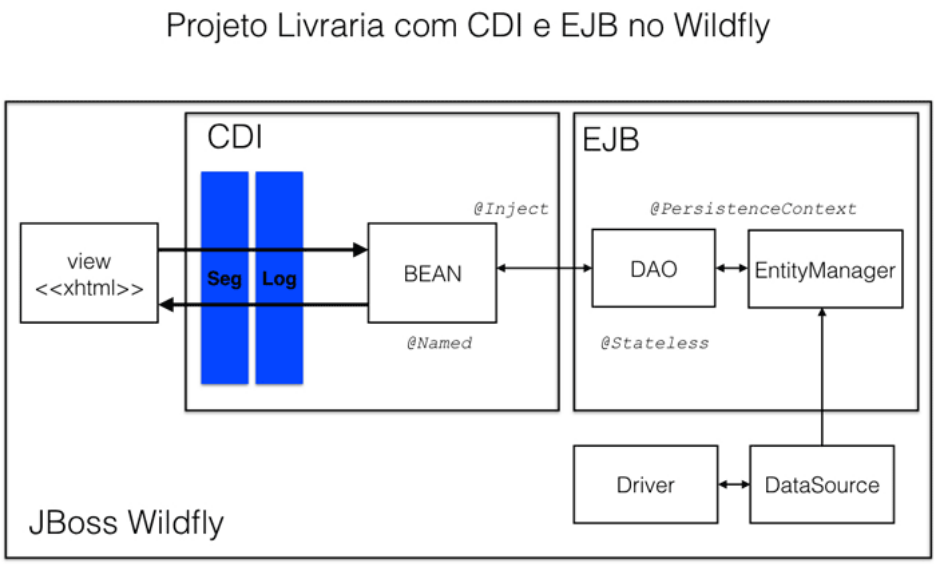
No Eclipse, a aba de servidores já há o Tomcat. Vamos clicar com o direito na área branca e selecionar a opção `New -> Server` . Dependendo se você já instalou o Wildfly ou não, pode ser que ele não faça parte da lista, sendo necessário clicar em `Show Downloadable Server Adapter` para instalar o JBoss AS Tools. A instalação do JBoss AS Tools pode demorar um pouco.

Com a opção do Wildfly aparecendo, vamos selecioná-la e vinculá-la ao diretório do servidor de aplicação que descompactamos.

Para adicionar o projeto no Wildfly, basta clicarmos com o botão direito , e ir em *Add and Remove...* , selecionar a `livraria-wildfly` , clicar em *Adicionar* e em seguida em *Finish*.

Entendendo o projeto

Agora que como usamos EJB no projeto, nossa camada de persistência mudou um pouco. Se olharmos o pocote `br.com.caelum.livraria.bean` , vemos que não precisamos mexer em nada. A única coisa que mudou é que agora não temos mais nosso pacote `br.com.caelum.livraria.tx` , pois assim como o Spring, agora nós temos um servidor mais sofisticado, que assume o gerenciamento da transação e assume essa responsabilidade.



EJB Stateless

Se olharmos no nosso `AutorDao` vemos que agora temos a anotação `@Stateless` , que significa que esse `DAO` agora é um EJB, o que nos diz que agora ele pode receber um `EntityManager` aonde anotamos com `@PersistenceContext` . Se você quer aprender com mais detalhes sobre esse mundo do Java EE, você pode conferir aqui no treinamento de Java EE do Alura. (https://cursos.alura.com.br/course/projeto-javaee) E se você quiser saber mais detalhes sobre EJB, pode conferir aqui neste curso (https://cursos.alura.com.br/course/ejb) focado neste assunto.

Preparando o DataSource

Se tentarmos rodar agora, vamos ver que o nosso servidor irá quebrar. Vemos nos erros que ele já encontra nossos `DAO's` , e carrega eles corretamente, mas vemos que ele dá um erro em uma `livraria-ds` , acusando uma dependência não satisfeita. Mas o que será isso ? Bom, o que mudou no projeto foi a nossa camada de persistência, então vamos dar uma olhada no nosso `persistence.xml` . Observamos que ele não tem mais nenhuma configuração, não tem mais usuários, url's de banco de dados e nem mais nada disso. Mas podemos notar que aqui encontramos a nossa configuração de datasource, que é exatamente o `livraria-ds` .

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://java.sun.com/xml/ns/persis
    version="2.0">

    <persistence-unit name="livraria" transaction-type="JTA"><!-- aqui temos o datasource -->

        <jta-data-source>java:/livraria-ds</jta-data-source>
```

```
        <class>br.com.caelum.livraria.modelo.Usuario</class>
        <class>br.com.caelum.livraria.modelo.Livro</class>
        <class>br.com.caelum.livraria.modelo.Autor</class>
        <class>br.com.caelum.livraria.modelo.Venda</class>

        <properties>
            <property name="hibernate.hbm2ddl.auto" value="update" />
            <property name="hibernate.show_sql" value="true" />
            <property name="hibernate.dialect" value="org.hibernate.dialect.MySQL5InnoDBDialect"
        </properties>
    </persistence-unit>

</persistence>
```

Um dos recursos que o **Wildfly** nos oferece é um **datasource**. Um **datasource** é normalmente um provedor de conexões, um *pool* de conexões. Quando o Wildfly inicializa, ele inicializa esses provedores de conexões, e a nossa aplicação faz uma solicitação a esse datasource através do nome `livrariads`. Só que quando subimos a aplicação, ela solicitou um datasource que ainda não configuramos. Ou seja, temos essa dependência a configurar.

Configurando o modulo

Vamos acesar o JBoss Wildflay fora do Eclipse. Entrando em `wildfly -> modules -> system -> layers -> base -> com`, vamos criar uma nova pasta: **mysql**. A ideia é que cadastremos o driver dentro do JBoss, e esse driver é o módulo que o JBoss irá carregar. Dentro da pasta `mysql`, vamos criar outra: **main**, e dentro dela colocaremos o **.jar** do *mysql*. Para que o JBoss interprete essa pasta e esse arquivo como módulo, é necessário um arquivo de configuração `module.xml`, que disponibilizamos para você aqui (https://s3.amazonaws.com/caelum-online-public/jsf-cdi/arquivos/module.xml). O que esse arquivo faz é cadastrar o `.jar` como módulo do JBoss.

```
<?xml version="1.0" encoding="UTF-8"?>
<module xmlns="urn:jboss:module:1.1" name="com.mysql">
  <resources>
    <resource-root path="mysql-connector-java-5.1.38.jar"/>
  </resources>
  <dependencies>
    <module name="javax.api"/>
  </dependencies>
</module>
`
```

Segue também o link do JAR do driver utilizado que deve estar na pasta `com/mysql/main`:

http://central.maven.org/maven2/mysql/mysql-connector-java/5.1.38/mysql-connector-java-5.1.38.jar (http://central.maven.org/maven2/mysql/mysql-connector-java/5.1.38/mysql-connector-java-5.1.38.jar)

Configurar o datasource

Porém agora temos um módulo do JBoss, como se fosse uma biblioteca disponível para ele usar. O que queremos é um dizer para o JBoss que esse módulo é um `Driver`. Por isso, voltamos a pasta raiz do `Wildfly -> standalone -> configuration`, e vamos mexer no arquivo **standalone.xml**. Vamos procurar por um elemento `<drivers>`, e cadastrar o nosso novo `driver`. Devemos adicionar:

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

Após isso, só falta configurar o `datasource`, adicione o seguinte xml dentro do elemento **`<datasources>`**:

```
<datasource jndi-name="java:/livraria-ds" pool-name="livrariaDS" enabled="true" use-java-contex
    <connection-url>jdbc:mysql://localhost:3306/livrariadb</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>
    </security>
</datasource>
```

Com isso, o JBoss irá cadastrar o `driver` e disponibilizar o `datasource`. Rodando a aplicação, vemos que está tudo configurado, não recebemos a exceção, nos indica que o contexto foi registrado e tudo funciona!

Parabéns por terminar o curso, alguma dúvida, sempre utilize o nosso fórum!