01

Integração com o Maven

Transcrição

Começando daqui? Você pode fazer o DOWNLOAD (https://s3.amazonaws.com/caelum-online-public/jsf-cdi/stages/capitulo-5.zip) completo do projeto do capítulo anterior e continuar seus estudos a partir deste capítulo.

Integração com o Maven

Nesse momento, você consegue imaginar o projeto fora do Eclipse? Sem ele ficamos perdidos. Estamos bastante acoplados com essa ferramenta, porém muitas vezes queremos rodar em outro lugar ou fazer deploy do nosso projeto. Além disso, perceba que estamos copiando todas as bibliotecas (`.jar`) para dentro da nossa pasta **lib**. Caso haja atualização de alguma delas, como por exemplo do Hibernate que estamos usando, teríamos que baixar os novos `.jar` , deletar os antigos e colocar dentro da pasta **lib** novamente. Isso dá bastante trabalho e nos faz perder muita produtividade.

O Maven vem para ajudar no gerenciamento do projeto, não é perfeito, porém é muito utilizado no mercado. Para aprender melhor a como fazer essa integração e a integração com outros projetos, nós temos um curso de Maven (https://cursos.alura.com.br/course/maven) aqui no Alura.

Criando um novo projeto

Vamos criar um novo projeto, que terá a integração com o Maven. Para isso, iremos em *New -> Project -> Maven Project*. Selecionamos **Create a simple project (skip archetype selection)**, pois iremos criar tudo do zero.

Algumas configurações na próxima página e depois *Finish*:

- **`Group Id`** : Esse é o nosso pacote, que no caso é `br.com.caelum` ;
- **`Artifact Id`** : Esse é o nome do projeto. Já temos `livraria` no `workspace` , então vamos colocar `livraria-maven` ;
- **`Version`** : `0.0.1-SNAPSHOT` ;
- **`Packaging`** : `war` ;
- **`Name`** : Livraria Alura;

Agora, iremos copiar todos os pacotes do projeto `livraria` **(apenas os pacotes, sem a pasta `META-INF`)** para dentro da pasta **`src/main/java`** do novo projeto. Nada compila ainda, mas não se preocupe. Agora sim iremos copiar a pasta `META-INF` , para dentro de **src/main/resources**. Por último, vamos copiar tudo que está **dentro** da pasta `WebContent` e colar na pasta **src/main/webapp**.

pom.xml

O `pom.xml` é o arquivo XML do Maven. Vamos lembrar que queremos que o Maven gerencie nossos `.jar` e acabe com o trabalho manual. Para isso, precisamos informar ao Maven quais dependências ele irá gerenciar. Dentro do XML, nós informamos isso ao Maven através da tag `<dependencies></dependencies>` . Porém fica difícil saber a versão, o nome da dependência, e por isso existe um site específico para nos ajudar, o mvnrepository.com (http://mvnrepository.com). Nele podemos procurar por dependências dentro do repositório do Maven na internet.

Por exemplo, podemos procurar a nossa implementação do **CDI**, que tem como `.jar` o arquivo `weld-servlet-2.3.3.Final.jar`. No mvnrepository.com (http://mvnrepository.com), pesquisamos por **weld servlet**. No link, ele te mostra todas as versões disponíveis do `.jar` no repositório do Maven. Clicamos na versão desejada (**2.3.3.Final**) e copiamos o XML de configuração do Maven:

```
<dependency>
    <groupId>org.jboss.weld.servlet</groupId>
    <artifactId>weld-servlet</artifactId>
    <version>2.3.3.Final</version>
</dependency>
```

Adicionada a dependência, o Maven começa a baixar o `.jar` do seu repositório e o guarda dentro de uma pasta local. Isso pode demorar um pouco. Você pode ver quais dependências estão sendo gerenciadas pelo Maven em `Java Resources -> Libraries -> Maven Dependencies`. Já separamos aqui os `.jar` do **mvnrepository** necessários (Se você já conhece o Maven, pode baixar o `pom.xml` completo aqui (https://s3.amazonaws.com/caelum-online-public/jsf-cdi/arquivos/pom.xml)):

- weld-servlet (http://mvnrepository.com/artifact/org.jboss.weld.servlet/weld-servlet/2.3.3.Final)
- validation-api (http://mvnrepository.com/artifact/javax.validation/validation-api/1.1.0.Final)
- primefaces (http://mvnrepository.com/artifact/org.primefaces/primefaces/5.3)
- mysql-connector (http://mvnrepository.com/artifact/mysql/mysql-connector-java/5.1.38)
- hibernate (http://mvnrepository.com/artifact/org.hibernate/hibernate-entitymanager/5.1.0.Final)
- jsf-api (http://mvnrepository.com/artifact/com.sun.faces/jsf-api/2.2.13)
- jsf-impl (http://mvnrepository.com/artifact/com.sun.faces/jsf-impl/2.2.13)

Porém, alguns `.jar` podem não existir no repositório do Maven. Para isso, precisamos adicionar no `pom` o repositório externo, para aí sim adicionar a dependência. Isso acontece com o *primefaces themes*. Você pode achar o repositório do PrimeFaces aqui (http://www.primefaces.org/downloads). Fora de `<dependencies>`, você adiciona o novo repositório:

```
<repository>
    <id>prime-repo</id>
    <name>PrimeFaces Maven Repository</name>
    <url>http://repository.primefaces.org</url>
    <layout>default</layout>
</repository>
```

E agora podemos pegar aqui (http://www.primefaces.org/themes) as dependências que não existiam no mvnrepository, porém existem no repositório deles adicionado.

```
<dependency>
    <groupId>org.primefaces.themes</groupId>
    <artifactId>all-themes</artifactId>
    <version>1.0.10</version>
</dependency>
```

Removendo Erros

Com as dependências todas gerenciadas pelo Maven, a maior parte dos erros sumiu. Porém, ao criarmos o projeto no Eclipse como *Maven Project*, ele acabou por se utilizar de algumas configurações bem antigas. A versão da JRE usada é a 1.5, e precisamos da JRE 1.8. Por isso, precisamos ir em *properties* do projeto, e no *Java Compiler*, desmarcamos a opção *Enable project specific settings*. Agora em *Java Build Path*, apagamos a *Library* `JRE System Library [J2SE-1.5]` e adicionamos uma nova, com a JRE 1.8.

Ainda sobra um erro, por causa da versão do `Java project facet` , basta clicarmos no erro, apertar `Ctrl+1` e mudar o `Java facet` para 1.8.

Rodando com o Maven

Como falado no início, podemos também rodar a aplicação através do Maven, para assim não ficarmos tão ligados ao Eclipse. Para isso, botão direito no projeto, e `Run As... -> Maven Build` . Em goals, ou seja, quando executarmos com o Maven, queremos que ele, por exemplo, dê um *clean*, compile o projeto e gere um WAR para *deploy*, para isso, colocamos:

```
clean compile package
```

Agora podemos rodar o projeto tanto com o Eclipse quanto com o Maven. Para terminar, vamos tirar o projeto antigo do Tomcat, adicionar o `livraria-maven` e rodar o Tomcat. Repare que tudo está funcionando normalmente.