02

Plugins

Transcrição

[00:00] Agora que já completamos os dois principais pilares, a parte de gerenciamento de dependências e de *build*; nessa aula, vamos conhecer alguns outros recursos adicionais do Maven. São alguns recursos que podem te ajudar em algumas situações. Um deles é justamente esse recurso de *plugin*.

[00:21] Eu tinha comentado bem por alto quando precisamos configurar a versão do Java da nossa aplicação, que por padrão o Maven estava considerando que era Java 5. Para você trabalhar com *plugins* aqui na *tag* `<build>` tem essa *tag* chamada `<plugins>`.

[00:36] Um *plugin* nada mais é do que algo que vai modificar, manipular o *build* da aplicação. Durante o *build* do projeto você pode adicionar *plugins*, que farão alguma coisa. Existem diversos *plugins*, cada um com um objetivo.

[00:49] Tínhamos conhecido esse *plugin* aqui, que é o `maven-compiler-plugin`. Éum *plugin* do próprio Maven, onde configuramos qual é a versão do Java, dos arquivos `.java` da aplicação e do `.class` na hora em que ele for fazer a compilação. Passamos aqui que é a versão 11.

[01:08] Um *plugin* tem esse objetivo, de adicionar algum recurso para o *build* da aplicação. Se abrirmos o navegador e no Google pesquisarmos "maven plugins", existe aqui site do Maven acessível neste link (https://maven.apache.org/plugins/) - que explica o que são os *plugins* e ele tem uma lista de *plugins* que são suportados por ele.

[01:28] Tem os *plugins* padrões do Maven para fazerem *clean, deploy, test*. Tem alguns *plugins* para empacotamento de acordo com algumas tecnologias - tipo `ear`, `ejb` e aplicações de JavaEE. *Plugins* para relatórios de `changeLog` de mudanças que foram feitas na aplicação, e documentação do site, `pmd` para testes e boas práticas de programação.

[01:54] *Plugins* para ferramentas do Ant. Tem vários *plugins* aqui, alguns antigos estão aposentados, esses daqui não funcionam mais. Tem alguns que estão fora da zona do Maven e não foi Maven que desenvolveu, mas ele suporta e destaca aqui.

[02:11] Alguns deles são interessantes. Por exemplo: esse do `jetty`, esse daqui é um *plugin* para você ter um servidor embutido na aplicação. Ao invés de ter um `tomcat` adicionado externo no Eclipse, você pode adicionar o `jetty` diretamente no Maven e executá-lo pelo Maven.

[02:32] No caso do `jetty`, como é que funciona? Se você adicionar esse *plugin* do `jetty` você poderia vir aqui e executar `mvn jetty:run`. Esse "Jetty:" é o nome do *plugin*, o prefixo do *plugin* e o "run" seria o *goal*.

[02:47] Perceba que um dos objetivos dos *plugins* é eu posso adicionar novos *goals* ao Maven nas etapas de *build* do Maven. Não existe esse *goal* `run` no Maven, por isso que ele está com `jetty:` aqui. Aí, seu eu tentar executar, vai falhar porque eu não adicionei esse *plugin* no “pom.xml” da aplicação.

[03:04] Essa é a ideia do *plugin*, ele serve para fazer alguma coisa no projeto - em especial na etapa de *build*. Tem vários *plugins* aqui e um deles eu vou mostrar para vocês - é um relacionado com testes automatizados, com a parte de relatório e cobertura de testes.

[03:21] É o *plugin* chamado "Jacoco". É uma ferramenta que analisa o código-fonte da aplicação e te dá um relatório de cobertura de teste. A porcentagem de cada classe, de cada pacote está coberto por testes automatizados. Para você encontrar classes que ainda não foram testadas e coisas do gênero.

[03:37] Para adicionar um novo *plugin* na *tag* `<build>` tem a *tag* `<plugins>`. Já tem aqui um *plugin*. Uso a tecla "Enter" e vou colar aqui. Eu já tinha copiado é um *plugin* bem extenso. Vem um detalhe meio chato, cada *plugin* funciona de um jeito, cada *plugin* tem uma configuração diferente.

[03:54] O *plugin* no `maven-compiler-plugin` era só declarar o `artifactId` e essa *tag* `<configuration>` com a versão do Java, `<source>` e o `<target>`. Simples assim!

[04:04] Agora esse do Jacoco não. Como ele não é do Maven, eu preciso do `<groupId>org.jacoco</groupId>` do `<artifactId>jacoco-maven-plugin</artifactId>`. Qual é a versão? `<version>0.8.2</version>`. Se ele tiver várias versões. Nesse tem uma *tag* chamada `<executions>` - já que é para ele executar algum *goal* em alguma fase do *build* do projeto.

[04:21] Tem uma aqui que é exigida por ele, que é para preparar e fazer uma análise do projeto. Tem uma outra `<execution>` - que é a principal, que é a de *report*, `<id>report</phase>`. O *goal* se chama *report* e ele será executado na fase de teste.

[04:37] Durante o *goal* de teste, o Jacoco vai executar logo sequência esse *goal* aqui de *report*, `<goal>report</goal>` - que é o que vai gerar o relatório em si. Essa é a configuração do Jacoco. Você tem que consultar a documentação do *plugin* para entender como você configura ele e como faz para executar. No caso do Jacoco é dessa maneira.

[04:56] Vou salvar aqui e vamos rodar pelo *prompt* aqui mesmo. Para rodar o Jacoco não se cria um *goal* a mais - ele até cria aquele *goal report*, mas se você rodar o próprio `mvn test` já vai funcionar porque aqui o *report* está sendo configurado para rodar na fase de teste.

[05:13] Perceba que ele rodou os testes e na sequência, `jacoco-maven-plugin:0.8.2:report`. Ele executou o Jacoco, o *goal* de *report*. Ele gerou um arquivo dizendo aqui `target/jacoco.exec`.

[05:26] Se entrarmos lá no nosso *target* tem um "jacoco.exec", mas é um arquivo `.exec`, não tem como executarmos esse arquivo. Se você reparar aqui tem uma pasta "site", que é onde o Maven gera a documentação da aplicação. No caso não estamos usando nenhum *plugin* para isso, por isso ele não tinha gerado.

[05:41] O Jacoco gera um site para mostrar esse relatório aqui. Na pasta "site" tem uma pasta "jacoco" e aqui dentro tem um "index.html". Ele gera uma página HTML com um relatório, se abrirmos aqui está aqui o relatório do Jacoco.

[05:56] É uma página bem feia, não tem CSS e não tem um visual bacana, mas para nós isso não importa muito. O que nos importa aqui é o conteúdo. Ele lista todos os pacotes da aplicação, no caso a nossa aplicação só tem 1 pacote, o nível de cobertura se estiver bom ficaria verde. Está ruim, está vermelho, porque está com 0% de cobertura.

[06:15] Podemos detalhar. Quando você clica no pacote, ele te mostra todas as classes e em cada classe ele mostra a cobertura. Nós só temos uma classe e ele mostra aqui métodos, construtores e o percentual de cobertura. Está tudo zerado, não tem nada sendo testado.

[06:30] Vamos tentar escrever um teste só para mudarmos isso daí. Nós até temos o `ProdutoTest`, só que tem um teste de vazio aqui que não faz nada, `@Test public void test () {}`.

[06:37] Vamos instanciar um produto aqui: `Produto p = new Produto()`.Lá na classe `produto`, se dermos uma olhada tem um atributo de nome, um atributo de preço, um construtor que recebe os dois atributos e um *getter* para cada um dos dois atributos.

[06:51] Vou criar aqui um produto `Produto p = new Produto("teste", BigDecimal.TEN)` e aí vou fazer um *assert* aqui: `Assert.assertEquals("teste",`

`p.getNome())`. É um teste bem ridículo aqui, é um teste não recomendado. Testar o método 'get' é algo bizarro. É só para vermos o relatório ali.

[07:24] Eu vou fazer um outro teste para ver se ele pega o preço aqui como sendo `(BigDecimal.TEN)`. Está dando um erro aqui. Foi o `Import`, não é desse pacote, é `assert` do "org Junit".

[07:38] É um teste bem meia boca, só para simular. Vamos rodar aqui novamente, `mvn test`. Ele vai recompilar tudo, retestar o relatório geral. Se usar a tecla "F5", olhe só que legal, agora já muda a figura!

[07:55] Vou mudar aqui para "Home". 100% de cobertura de teste. A barra verde dizendo que está boa a sua cobertura. Detalhou a classe produto. Aqui tudo coberto, o construtor, os *getters*. É um *plugin* para cobertura de testes.

[08:11] Essa é que é a ideia dos *plugins*. Existem vários *plugins* na comunidade para relatório de testes - para modificar o *build* do projeto, para executar um servidor, para gerar um PDF e para rodar SQL. Tem *plugins* para tudo que você possa imaginar. Inclusive, você pode criar um *plugin*. Você pode criar um *plugin* que vai fazer alguma coisa durante o *build* de uma aplicação. É totalmente possível fazer isso!

[08:37] Essa é uma das vantagens do Maven. Ele é extensível, não é engessado, só tem aquelas funcionalidades "x", "y", "z" e acabou; você só pode usar aquilo. Ele te permite estendê-lo, adicionar *plugins* para ter novos comportamentos e novas funcionalidades. Isso é algo extremamente importante em uma aplicação, essa capacidade de extensão para você adicionar novos comportamentos. Tudo bem?

[08:59] Espero que tenham gostado de entender melhor como funciona esses *plugins* e essa que é ideia de um *plugin* que está associado com o *build* do projeto e que ele vai estender o Maven, adicionando novas características. Tudo bem?

[09:12] Vejo vocês na próxima aula onde vamos conhecer outros recursos do Maven. Até lá!