Gerenciamento de Dependências e Build em Java com Maven

Apache Maven é uma ferramenta para gerenciar build e dependências de um projeto.

Quando há um projeto java em que tenha dezenas ou centenas de classes, seria inviável compilar e executar todas as classes via linha de comando. Assim, o Maven serve para realizar tal função, facilitando os processos de testes e desenvolvimento.

Para o desenvolvedor, e outros que querem contribuir com o projeto, entender todas as versões, dependências de frameworks, e outras características fica mais organizado e facilita a compreensão, fornecendo informações de qualidade (por exemplo, verificar se os testes estão bons, se falta algo).

Facilita o download e importação de projetos ou bibliotecas.

Primeiro projeto e conceitos

- *Criando um projeto via linha de comando*

No diretório onde estará o projeto:

- mvn archetype:generate -DgroupId=one.digitalinnovation -DartifactId=quick-start-maven -Darchetype=maven-archetype-quickstart -DinteractiveMode=false

Esse comando busca um template que existe na internet (nos repositórios do Maven) um padrão, ou uma estrutura, de projeto Java pré-definido.

-DgroupId: é como o id da organização. Segue as regras de nomes de pacote Java

-DartifactId: nome do projeto

- *Comandos que auxiliam o dia a dia*

No diretório do projeto:

- mvn compile

  Isso irá compilar as classes e levar ao diretório target

- mvn test

  Com isso é possível que o maven encontre todas as classes de teste que existe no projeto e execute as validações de cada uma delas. Executa e mostra se falha ou não.

- mvn package

Cria o .jar, a aplicação, na pasta target.

- mvn clean

Isso irá limpar o diretório de trabalho (apaga a pasta target). Isso é feito quando o trabalho é concluído e funcionando corretamente.

- *Criando diferentes tipos de projeto*

Isso é possível através da utilização do Maven archetype, que é um template que possibilita a personalização e a configuração de como o projeto vai ser construído. Neste template definimos versão de componentes, quais componentes vão ser inseridos automaticamente, organização de pacotes e organização de arquivos. Dessa forma é possível construir diversos tipos de templates para diversas finalidades.

Por exemplo: projeto web que faça empacotamento de arquivos.

"**maven archetype list**" – pesquisar na internet para visualizar melhor.

- *POM*

POM – Project Object Model – está no formato XML

É uma unidade fundamental de trabalho

Detalha o projeto e detalhe como construir aquele projeto

Maven sempre procura pelo pom.xml para realizar sua execução

Informações que aparecem no arquivo pom.xml

- Nome do projeto

- Dependências

- Módulos

- Configurações de build

- Detalhes do projeto (nome, descrição, licença, url)

- Configurações de ambiente (repositórios, tracking, profiles)

Super POM é o modelo base onde tem todas as configurações padrão pelas quais o Maven utiliza. O pom.xml estende do super POM.

- *Repositórios*

Locais onde podemos encontrar plugins e bibliotecas que o Maven provê. Há dois tipos de repositórios que podemos encontrar: local e remoto.

- O repositório remoto é o local central utilizado pelo Maven para buscar os artefatos, configurado automaticamente pelo Super POM para utilizar o Maven Central (é um site com todas as dependências).

É possível fazer uma configuração específica de repositório remoto via pom.xml do projeto.

- O repositório local é o repositório na máquina utilizado pelo Maven para buscar os artefatos. É uma estratégia de caching para evitar que todas as execuções do Maven localmente tenham que fazer uma busca externa.

Como isso funciona?

Por exemplo, quando executamos mvn compile. Se existir um repositório local, essa chamada é realizada e executa – até de forma mais rápida – a compilação do nosso projeto. Agora, se for adicionado uma dependência que não exista no nosso repositório local, maven irá buscar a referência do repositório remoto, fazer o download deste componente e gravar no repositório local para futuras consultas.

- *Como adicionar dependências*

É adicionado a tag dependency dentro de dependencies.

Gerenciando dependências

- *Tipos de dependências*

Direta: dependências declaradas no pom.xml

Transitiva: dependências obrigatórias das dependências declaradas no pom.xml

- mvn install

Publica o projeto localmente no repositório

- *Transitividade e Escopos*

Para reutilizar um outro projeto com um novo criado, há o problema de transitividade entre projetos.

Escopo default: É o escopo padrão.

Escopo provided: Indica que a dependência será fornecida em tempo de execução por uma implementação na JDK ou via container.

- *Dicas sobre escopos, dependências opcionais e exclusions*

Pedir para Maven apresentar o classpath construído pelos tipos de escopos:

- mvn dependency:build-classpath -DincludeScope=compile
- mvn dependency:build-classpath -DincludeScope=test
- mvn dependency:build-classpath -DincludeScope=runtime

Dependências opcionais. A dependência que tiver a tag optional com valor true não será enviado quando reutilizar o pom.xml.

Exclusions. Utilizado quando o componente que você usa compartilha uma biblioteca que você já tem ou não quer ter disponível. Utiliza-se a tag exclusion (dentro de exclusions – dentro de certa dependência) com groupId e artifactId da dependência que se deseja excluir.

Maven Build Lifecycle

- Conceito de ciclo de vida de construção – estruturar a arquitetura, estrutura de pacotes, executar testes e caminhar até uma publicação.

- Conceito e os comandos da ferramenta

- Composto por 3 ciclos de vida

- Cada um desses ciclos possui fases – Maven Phases

- Cada fase possui objetivos – Maven Goals

Ciclos de vida:

Default Lifecycle: principal ciclo. É responsável pelo deploy local. Composto por 23 fases.

Clean Lifecycle: ciclo intermediário. Responsável pela limpeza do projeto. Composto por 3 fases.

Site Lifecycle: ciclo final. Responsável pela criação do site de documentação do projeto. Composto por 4 fases.

Projetos multi-módulos

Pode ter os submódulos dentro de um projeto agregador. Por exemplo, um projeto agregador tenha submódulos “core”, “service”, “controller”.

Mostrando na prática...

Construir um projeto agregador:

- mvn archetype:generate -DgroupId=one.digital.innovation -DartifactId=Project-parent -Darchetype=maven-quick-start

Entrando no arquivo pom.xml de dentro do projeto agregador criado acima, é adicionado a tag, abaixo de version, <packaging>, com o valor pom. Isto quer dizer que ele é um projeto agregador, que terá submódulos.

Construindo os submódulos dentro do projeto agregador:

- mvn archetype:generate -DgroupId=one.digital.innovation -DartifactId=core -Darchetype=maven-quick-start -DinteractiveMode=false
- mvn archetype:generate -DgroupId=one.digital.innovation -DartifactId=service -Darchetype=maven-quick-start -DinteractiveMode=false
- mvn archetype:generate -DgroupId=one.digital.innovation -DartifactId=controller -Darchetype=maven-quick-start -DinteractiveMode=false

Dessa forma que é realizada a construção de um projeto multi-módulo.

Plugins

Compile, Clean, Test, etc. são todos plugins.

Utiliza o plugin: mvn [plugin-name]:[goal-name]

A configuração do plugin é sempre feito dentro da tag <build>.

Para fazer a configuração do javadoc, é colocado o plugin do javadoc no pom.xml e depois chamado pelo mvn (mvn javadoc:javadoc).