**Apache Maven**

O Maven é uma ferramenta de gerenciamento e automação de construção (build) de projetos. Entretanto, por fornecer diversas funcionalidades adicionais através do uso de plugins e estimular o emprego de melhores práticas de organização, desenvolvimento e manutenção de projetos, é muito mais do que apenas uma ferramenta auxiliar. Maven é uma ferramenta de automação e gerenciamento de projetos Java, embora também possa ser utilizada com outras linguagens.

Benefícios do apache Maven para construção e integração de projetos, pode-se citar:

* Existência de plug-ins para as principais IDEs: Eclipse e NetBeans;
* Configuração inicial do projeto com o uso de boas práticas e sem a necessidade de codificação;
* Ao criar um projeto Maven utilizando archetypes (templates), automaticamente são criadas pastas com arquivos, classes com exemplos de código e o pom.xml já previamente configurado;
* Uso do repositório Maven Central para gerenciamento das bibliotecas open source;
* Uso do repositório local, permitindo que a mesma biblioteca seja utilizada por qualquer projeto configurado na máquina;
* Possibilidade de criação de repositórios próprios para o gerenciamento de bibliotecas internas, facilitando a reutilização destas nos projetos;
* Possibilidade de usar tarefas Ant para o deployment, facilitando a reutilização do legado;
* Gerenciamento de bibliotecas e versionamento com pouca ou nenhuma configuração adicional.

A unidade básica de configuração do Maven é um arquivo chamado [pom.xml](http://maven.apache.org/guides/introduction/introduction-to-the-pom.html), que deve ficar na raiz do seu projeto. Ele é um arquivo conhecido como Project Object Model: lá você declara a estrutura, dependências e características do seu projeto.

A idéia é bem parecida com o build.xml do Ant: você deixa o pom.xml na raiz do seu projeto para poder chamar as targets de build do seu projeto. O [menor arquivo pom.xml](http://maven.apache.org/guides/introduction/introduction-to-the-pom.html#Minimal_POM) válido é o seguinte:  
  
<project>  
  <modelVersion>4.0.0</modelVersion>  
  <groupId>br.com.caelum</groupId>  
  <artifactId>teste</artifactId>  
  <version>1.0</version>  
</project>

Que contém apenas a identificação do projeto, e uma informação a mais: modelVersion, que é a identificação da versão do arquivo pom.xml e deve ser sempre 4.0.0. A identificação do projeto consiste em três informações:

* groupId: um identificador da empresa/grupo ao qual o projeto pertence. Geralmente o nome do site da empresa/grupo ao contrário. Ex: br.com.caelum.
* artifactId: o nome do projeto. Ex: teste.
* version: a versão atual do projeto. Ex: 1.0-SNAPSHOT.

No [Repositório de Bibliotecas do Maven](http://mvnrepository.com/) você encontra os jars que você pode colocar como dependência do seu projeto, e o pedaço de xml que você deve copiar e colar dentro da tag dependencies do seu pom para incluir essas bibliotecas.

Todos os jars baixados pelo Maven são guardados na pasta repository dentro da M2\_HOME que você configurou quando instalou o Maven. Assim, se mais de um projeto seu depende do mesmo jar, ele não é baixado de novo.

A grande diferença entre o build.xml do Ant e o pom.xml do Maven é o paradigma. No Ant usamos esse XML praticamente como uma linguagem de programação, onde você da comandos em relação ao build do projeto.

No Maven usamos o XML para definir a estrutura do projeto, e a partir dessas declarações o Maven possui targets bem definidos que usam essas informações para saber como realizar aquela tarefa.

Um exemplo: para compilar com o Ant criamos um target que chama o javac, mas para compilar com o Maven usamos um target já existente (não o criamos), e ele vai usar a informação que define onde está o código fonte e para onde ele deve ser compilado (sendo que muitas dessas informações possuem convenções e defaults, e nem precisam ser configuradas).

**O ciclo de vida do Maven**

Agora com um projeto Maven já preparado, vamos para a principal funcionalidade: o build. O build do Maven é baseado no conceito de [ciclo de vida](http://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html): o processo de construção e distribuição da sua aplicação é dividido em partes bem definidas chamadas fases, seguindo um ciclo. O ciclo padrão é o seguinte:

* **compile** – compila o código fonte do projeto
* **test** – executa os testes unitários do código compilado, usando uma ferramenta de testes unitários, como o junit.
* **package** – empacota o código compilado de acordo com o empacotamento escolhido, por exemplo, em JAR.
* **integration-test** – processa e faz o deploy do pacote em um ambiente onde os testes de integração podem ser rodados.
* **install** – instala o pacote no repositório local, para ser usado como dependência de outros projetos locais
* **deploy** – feito em ambiente de integração ou de release, copia o pacote final para um repositório remoto para ser compartilhado entre desenvolvedores e projetos

Você pode invocar qualquer dessas fases na linha de comando, digitando:  
mvn [fase]

Por exemplo, se você digitar mvn package o Maven vai executar todas as fases anteriores do ciclo até a fase package. Uma lista completa das fases do ciclo de vida possíveis pode ser encontrada [aqui](http://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html#Lifecycle_Reference).

Algumas das fases do ciclo possuem plugins associadas a elas, e esses plugins são executados assim que a fase é chamada para ser executada. Você pode também registrar plugins para rodarem em qualquer fase do ciclo, conseguindo, assim, personalizar o build do seu projeto facilmente.

Por exemplo, se você quiser criar um jar com o código fonte do projeto, e que esse jar seja gerado depois que o projeto foi empacotado, é só acrescentar no seu pom:

<project>

...

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-source-plugin</artifactId>

<executions>

<execution>

<id>attach-sources</id>

<phase>package</phase>

<goals>

<goal>jar</goal>

</goals>

</execution>

</executions>

</plugin>

</plugins>

</build>

...

</project>

Assim, o plugin [Source](http://maven.apache.org/plugins/maven-source-plugin/) vai executar seu goal jar na fase package do ciclo de vida. É como se fosse chamado mvn source:jar quando o build passa pela fase de package. A fase package já possui um plugin associado a ela: o jar:jar (supondo que é um projeto jar), então o plugin source só será executado depois do jar:jar.

Em geral se você registrar mais de um plugin pra mesma fase, eles serão executados na ordem em que eles forem declarados. O jeito de configurar o plugin para colocá-lo dentro de uma fase do ciclo geralmente está no site principal do plugin, na seção Usage.

**Principais Conceitos Básicos**

**Project Object Model (POM)**

Cada projeto Maven fornece um arquivo pom.xml que captura dependências, propriedades de estrutura do projeto, tarefas da fase de construção e comportamento. A maioria das propriedades POM possui padrões que resultam em um mecanismo compacto, porém poderoso, para descrever o comportamento da construção do projeto.

**Fases de construção e o ciclo de vida de construção**

Um ciclo de vida de construção consiste em várias fases. Quando um comando de fase é fornecido, o Maven executa cada fase na sequência até e incluindo a fase definida. Após o arquivo pom.xml ser definido, as ferramentas Maven priorizam as fases de construção específicas e reagem às fases de validação, geração de códigos, conjunto de recursos e compilação.

**Um ciclo de vida de construção consiste nas seguintes fases:**

validar

compilar

testar

empacotar

teste de integração

verificar

instalar

implementar

**Objetivo**

Um objetivo representa uma tarefa específica que é melhor do que uma fase de construção e que contribui para a criação e o gerenciamento de um projeto.

**Pacote**

A configuração do empacotamento designa um conjunto de objetivos padrão. Exemplos de valores de empacotamento válidos incluem jar, war, ear e pom.

**Plug-in Maven**

Um plug-in descreve um conjunto de objetivos que estão ligados a um esquema ou processo de empacotamento específico.

**Mojo**

Uma tarefa específica que é implementada dentro de um plug-in. Por exemplo, uma implementação de classe Java™ para implementar em seu ambiente de tempo de execução preferencial.

**Archetype**

Archetypes são utilizados como modelos de projeto para configurar novos projetos. Esses modelos tornam mais fácil ativar padrões dentro de sua organização, definindo objetivos de empacotamento, configurações de plug-in e dependências predefinidas para as bibliotecas padrão.

**Repositórios Maven**

Repositórios são utilizados para armazenar artefatos de construção e dependências de tipos variados. Por exemplo, archetypes, plug-ins, e arquivos JAR, entre outros. Repositórios locais são preenchidos vagarosamente, conforme necessário, a partir de repositórios remotos para propósitos de construção.

[**Visão geral de ferramentas e projetos**](https://www.ibm.com/support/knowledgecenter/pt-br/SSHR6W/com.ibm.websphere.wdt.doc/topics/tools_overview.htm?view=kc)

Existem várias ferramentas que são fornecidas para o desenvolvimento de Maven que você pode usar para definir as dependências do módulo Java EE. Também é possível usar entradas POM para definir a estrutura do projeto.

**Referencias**

<https://www.ibm.com/support/knowledgecenter/pt-br/SSHR6W/com.ibm.websphere.wdt.doc/topics/key_concepts.htm>

<http://blog.caelum.com.br/processo-de-build-com-o-maven/>

<http://blog.caelum.com.br/processo-de-build-com-o-maven/>

<http://www.devmedia.com.br/fundamentos-do-maven/29315>

<http://www.devmedia.com.br/introducao-ao-maven/25128>