

Maven

Conceitos, finalidade e exemplos práticos de aplicação com Java

1. O que é o Maven?

Apache Maven é uma ferramenta de automação e gerenciamento de projetos de software, amplamente utilizada no ecossistema Java. Criado pela Apache Software Foundation, o Maven tem como objetivo padronizar a forma como os projetos são construídos, organizados e documentados, seguindo o conceito de "**Convenção sobre Configuração**" (Convention over Configuration). Isso significa que, ao seguir uma estrutura de diretórios e regras padronizadas, o desenvolvedor precisa escrever muito menos configuração manual do que em outras abordagens.

O Maven descreve um projeto por meio de um arquivo central chamado **pom.xml** (Project Object Model), no qual são declaradas informações como o nome do projeto, a versão, as dependências externas necessárias, os plugins utilizados e as fases do processo de build.

2. Conceitos fundamentais

2.1 POM (Project Object Model)

É o arquivo XML (pom.xml) que representa o "coração" de qualquer projeto Maven. Nele são definidos o identificador do projeto, suas dependências, plugins, propriedades e informações sobre o processo de construção (build).

2.2 Coordenadas do projeto (GAV)

Todo projeto e toda dependência no Maven são identificados de forma única por três coordenadas, conhecidas como GAV:

- **groupId** — identifica a organização ou o grupo responsável pelo projeto (ex.: org.springframework);
- **artifactId** — identifica o nome do projeto ou da biblioteca (ex.: spring-core);
- **version** — indica a versão específica do artefato (ex.: 6.1.2).

2.3 Dependências

São bibliotecas externas necessárias para o funcionamento do projeto. O Maven baixa automaticamente essas dependências (e as dependências delas, chamadas de dependências transitivas) a partir de repositórios remotos, dispensando o download manual de arquivos JAR.

2.4 Repositórios

São locais onde os artefatos (bibliotecas, plugins) ficam armazenados. Existem três tipos principais:

- **Repositório local** — pasta no computador do desenvolvedor (geralmente ~/.m2/repository), onde ficam os artefatos já baixados;
- **Repositório central** — repositório público mantido pela comunidade Maven (Maven Central), com milhares de bibliotecas disponíveis;
- **Repositório remoto/privado** — servidores internos de empresas (como Nexus ou Artifactory) usados para hospedar artefatos próprios ou privados.

2.5 Ciclo de vida do build (Build Lifecycle)

O Maven organiza o processo de construção do software em fases sequenciais. As principais fases do ciclo de vida padrão (default lifecycle) são:

- **validate** — valida se o projeto está correto e todas as informações necessárias estão disponíveis;
- **compile** — compila o código-fonte do projeto;
- **test** — executa os testes unitários utilizando um framework de testes adequado;
- **package** — empacota o código compilado em seu formato distribuível, como JAR ou WAR;
- **verify** — executa verificações sobre os resultados de testes de integração;
- **install** — instala o pacote no repositório local, tornando-o disponível para outros projetos locais;
- **deploy** — copia o pacote final para um repositório remoto, disponibilizando-o para outros desenvolvedores ou ambientes.

2.6 Plugins e Goals

Plugins são extensões que adicionam funcionalidades ao Maven, como compilar código, gerar documentação, criar relatórios ou empacotar aplicações. Cada plugin oferece um ou mais **goals** (objetivos), que são tarefas específicas executadas em uma fase do ciclo de vida.

3. Para que serve o Maven?

O Maven resolve diversos problemas comuns no desenvolvimento de software Java, tornando o processo mais organizado, previsível e produtivo. Entre as principais finalidades, destacam-se:

- **Gerenciamento de dependências** — elimina a necessidade de baixar e configurar bibliotecas manualmente, resolvendo automaticamente conflitos de versão;
- **Padronização de projetos** — define uma estrutura de diretórios única, facilitando a compreensão de qualquer projeto Maven por outros desenvolvedores;
- **Automação de build** — automatiza tarefas repetitivas como compilação, testes, empacotamento e implantação (deploy);
- **Reutilização de código** — permite publicar e reutilizar bibliotecas próprias entre diferentes projetos e equipes;
- **Integração com ferramentas de CI/CD** — é amplamente suportado por ferramentas como Jenkins, GitHub Actions e GitLab CI, facilitando pipelines de integração e entrega contínua;
- **Geração de relatórios e documentação** — pode gerar automaticamente relatórios de testes, cobertura de código e documentação Javadoc.

4. Estrutura padrão de um projeto Maven

Ao seguir a convenção do Maven, um projeto Java típico possui a seguinte organização de pastas:

```

meu-projeto/
├── pom.xml
├── src/
│   ├── main/
│   │   ├── java/          -> código-fonte da aplicação
│   │   └── resources/     -> arquivos de configuração, propriedades, etc.
│   └── test/
│       ├── java/          -> código-fonte dos testes
│       └── resources/     -> recursos usados nos testes
└── target/                -> arquivos gerados pelo build (.class, .jar, etc.)

```

5. Exemplos de aplicação com Java

5.1 Exemplo de arquivo pom.xml básico

Abaixo está um exemplo de arquivo pom.xml para um projeto Java simples, incluindo a dependência do JUnit para testes:

pom.xml

```

<project xmlns="http://maven.apache.org/POM/4.0.0">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.exemplo</groupId>
  <artifactId>calculadora</artifactId>
  <version>1.0.0</version>
  <packaging>jar</packaging>

  <properties>
    <maven.compiler.source>17</maven.compiler.source>
    <maven.compiler.target>17</maven.compiler.target>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <dependencies>
    <dependency>
      <groupId>org.junit.jupiter</groupId>
      <artifactId>junit-jupiter</artifactId>
      <version>5.10.0</version>
      <scope>test</scope>
    </dependency>
  </dependencies>

</project>

```

5.2 Adicionando uma dependência externa (Spring Boot)

Para utilizar frameworks como o Spring Boot em uma aplicação Java, basta declarar as dependências desejadas no pom.xml. O Maven cuida automaticamente do download:

```

<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
    <version>3.2.0</version>
  </dependency>

```

</dependencies>

5.3 Classe Java de exemplo

Calculadora.java

```
package com.exemplo;

public class Calculadora {

    public int somar(int a, int b) {
        return a + b;
    }

    public static void main(String[] args) {
        Calculadora calc = new Calculadora();
        System.out.println("Resultado: " + calc.somar(5, 7));
    }
}
```

5.4 Teste unitário com JUnit

CalculadoraTest.java

```
package com.exemplo;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertEquals;

public class CalculadoraTest {

    @Test
    void deveSomarDoisNumeros() {
        Calculadora calc = new Calculadora();
        assertEquals(12, calc.somar(5, 7));
    }
}
```

5.5 Principais comandos do Maven

Os comandos abaixo são executados no terminal, dentro do diretório do projeto (onde está o pom.xml):

```
mvn validate    # Valida o projeto
mvn compile     # Compila o código-fonte
mvn test        # Executa os testes automatizados
mvn package     # Gera o arquivo .jar ou .war
mvn install     # Instala o pacote no repositório local
mvn clean       # Remove os arquivos gerados no build anterior
mvn clean install # Combinação comum: limpa e reconstrói o projeto
```

5.6 Executando a aplicação a partir do JAR gerado

```
java -jar target/calculadora-1.0.0.jar
```

6. Vantagens de utilizar o Maven

-
- Redução significativa da configuração manual de projetos;
 - Gerenciamento automático de dependências e suas versões;
 - Ampla adoção pela comunidade Java, com grande suporte e documentação;
 - Integração nativa com IDEs populares, como IntelliJ IDEA, Eclipse e VS Code;
 - Facilidade de integração com pipelines de CI/CD e ferramentas de qualidade de código;
 - Estrutura padronizada, o que facilita a manutenção e a colaboração em equipe.

7. Considerações finais

O Maven é uma das ferramentas mais consolidadas do ecossistema Java, sendo utilizado tanto em pequenos projetos pessoais quanto em grandes sistemas corporativos. Seu modelo declarativo, baseado no arquivo `pom.xml`, simplifica o gerenciamento de dependências e a automação do processo de build, permitindo que os desenvolvedores foquem mais na lógica da aplicação e menos em tarefas repetitivas de configuração. Embora existam alternativas modernas, como o Gradle, o Maven continua sendo amplamente utilizado e é considerado uma ferramenta essencial para quem trabalha com desenvolvimento Java.