Ant vs Maven vs Gradle

1. Introdução

Neste artigo, exploraremos **três ferramentas de automação de compilação Java que dominaram o ecossistema da JVM: Ant, Maven e Gradle** .

Vamos apresentar cada uma delas e explorar como as ferramentas de automação de compilação Java evoluíram.

2. Apache Ant

Inicialmente, o Make era a única ferramenta de automação de compilação disponível além de soluções desenvolvidas internamente **.** O Make existe desde 1976 e, como tal, era usado para compilar aplicações Java nos primórdios da linguagem.

No entanto, muitas convenções dos programas em C não se encaixavam no ecossistema Java, então, com o tempo, o Ant assumiu o papel de uma alternativa melhor.

Apache Ant (“Outra Ferramenta Incrível”) é uma biblioteca Java usada para automatizar processos de compilação de aplicações Java . Além disso, o Ant pode ser usado para compilar aplicações que não utilizam Java. Inicialmente, fazia parte do código-fonte do Apache Tomcat e foi lançado como um projeto independente em 2000.

Em muitos aspectos, o Ant é muito semelhante ao Make, e é simples o suficiente para que qualquer pessoa possa começar a usá-lo sem pré-requisitos específicos. Os arquivos de compilação do Ant são escritos em XML e, por convenção, são chamados de build.xml .

As diferentes fases de um processo de construção são chamadas de "metas".

Aqui está um exemplo de um arquivo build.xml para um projeto Java simples com a classe principal HelloWorld :

```
<project>
    <target name="clean">
        <delete dir="classes" />
    </target>

    <target name="compile" depends="clean">
        <mkdir dir="classes" />
        <javac srcdir="src" destdir="classes" />
    </target>

    <target name="jar" depends="compile">
        <mkdir dir="jar" />
        <jar destfile="jar/HelloWorld.jar" basedir="classes">
            <manifest>
                <attribute name="Main-Class"
```

```
                        value="antExample.HelloWorld" />
                </manifest>
            </jar>
        </target>


        <target name="run" depends="jar">
            <java jar="jar/HelloWorld.jar" fork="true" />
        </target>
</project>Cópia
```

Este arquivo de compilação define quatro alvos: `clean` , `compile` , `jar` e `run` . Por exemplo, podemos compilar o código executando o seguinte comando:

```
ant compileCópia
```

Isso acionará primeiro o comando ` clean` , que excluirá o diretório "classes". Depois disso, o comando ` compile` recriará o diretório e compilará a pasta `src` dentro dele.

A principal vantagem do Ant é a sua flexibilidade. O Ant não impõe convenções de codificação ou estruturas de projeto específicas. Consequentemente, isso significa que o Ant exige que os desenvolvedores escrevam todos os comandos por conta própria, o que às vezes resulta em arquivos XML de compilação enormes e difíceis de manter.

Como não existem convenções, o simples conhecimento de Ant não significa que entenderemos rapidamente qualquer arquivo de compilação Ant. Provavelmente levará algum tempo para nos acostumarmos com um arquivo Ant desconhecido, o que é uma desvantagem em comparação com outras ferramentas mais recentes.

Inicialmente, o Ant não possuía suporte integrado para gerenciamento de dependências. No entanto, como o gerenciamento de dependências se tornou imprescindível nos anos seguintes, o Apache Ivy foi desenvolvido como um

subprojeto do Apache Ant. Ele é integrado ao Apache Ant e segue os mesmos princípios de design.

No entanto, as limitações iniciais do Ant, devido à falta de suporte integrado para gerenciamento de dependências e às frustrações ao trabalhar com arquivos de compilação XML difíceis de gerenciar, levaram à criação do Maven.

3. Apache Maven

O Apache Maven é uma ferramenta de gerenciamento de dependências e automação de compilação, usada principalmente para aplicações Java. **O Maven continua a usar arquivos XML, assim como o Ant, mas de uma forma muito mais gerenciável.** O princípio básico aqui é a convenção sobre a configuração.

Enquanto o Ant oferece flexibilidade e exige que tudo seja escrito do zero, **o Maven se baseia em convenções e fornece comandos predefinidos (objetivos).**

Em resumo, o Maven nos permite focar no que nossa compilação deve fazer e nos fornece a estrutura para isso. Outro aspecto positivo do Maven era o suporte integrado para gerenciamento de dependências.

O arquivo de configuração do Maven, que contém instruções de compilação e gerenciamento de dependências, é chamado por convenção de pom.xml . Além disso, o Maven também prescreve uma estrutura de projeto rígida, enquanto o Ant oferece flexibilidade nesse aspecto.

Aqui está um exemplo de um arquivo pom.xml para o mesmo projeto Java simples com a classe principal HelloWorld de antes:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
      http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>baeldung</groupId>
```

```
    <artifactId>mavenExample</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <description>Maven example</description>

    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
            <scope>test</scope>
        </dependency>
    </dependencies>
</project>Cópia
```

No entanto, agora a estrutura do projeto também foi padronizada e está em conformidade com as convenções do Maven:

```
+---src
|   +---main
|   |   +---java
|   |   |   \---com
|   |   |       \---baeldung
|   |   |           \---maven
|   |   |                   HelloWorld.java
|   |   |
|   |   \---resources
|   \---test
|       +---java
|       \---resourcesCópia
```

Diferentemente do Ant, não há necessidade de definir manualmente cada uma das fases do processo de construção. Em vez disso, podemos simplesmente chamar os comandos integrados do Maven.

Por exemplo, podemos compilar o código executando o seguinte comando:

```
mvn compileCópia
```

Em sua essência, como observado nas páginas oficiais, **o Maven pode ser considerado um framework de execução de plugins, já que todo o trabalho é realizado por eles.** O Maven suporta uma ampla gama de plugins disponíveis , e cada um deles pode ser configurado adicionalmente.

Um dos plugins disponíveis é o Apache Maven Dependency Plugin, que possui um objetivo chamado copy-dependencies , o qual copia nossas dependências para um diretório especificado.

Para demonstrar este plugin em ação, vamos incluí-lo no nosso arquivo pom.xml e configurar um diretório de saída para as nossas dependências:

```
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-dependency-plugin</artifactId>
            <executions>
                <execution>
                    <id>copy-dependencies</id>
                    <phase>package</phase>
                    <goals>
                        <goal>copy-dependencies</goal>
                    </goals>
                    <configuration>
```

```
                                    <outputDirectory>target/dependencies
                                        </outputDirectory>
                                </configuration>
                            </execution>
                        </executions>
                </plugin>
        </plugins>
</build>
```

Este plugin será executado na fase de pacote , portanto, se executarmos:

```
mvn packageCópia
```

Vamos executar este plugin e copiar as dependências para a pasta target/dependencies.

Existe também um artigo sobre como criar um arquivo JAR executável usando diferentes plugins do Maven. Além disso, para uma visão geral detalhada do Maven, consulte este guia básico sobre o Maven , onde alguns dos principais recursos do Maven são explorados.

O Maven tornou-se muito popular, pois os arquivos de compilação passaram a ser padronizados e a manutenção desses arquivos levou significativamente menos tempo em comparação com o Ant. No entanto, embora mais padronizados do que os arquivos do Ant, os arquivos de configuração do Maven ainda tendem a ser grandes e complexos.

As convenções rígidas do Maven têm como contrapartida uma flexibilidade muito menor em comparação com o Ant. A personalização de objetivos é bastante complexa, tornando a escrita de scripts de compilação personalizados muito mais difícil do que com o Ant.

Embora o Maven tenha trazido melhorias significativas em relação à facilidade e padronização dos processos de construção de aplicações, ele ainda apresenta uma desvantagem: sua menor flexibilidade em comparação com o Ant. Isso levou à criação do Gradle, que combina o melhor dos dois mundos: a flexibilidade do Ant e os recursos do Maven.

4. Gradle

O Gradle é uma ferramenta de gerenciamento de dependências e automação de compilação que **foi construída com base nos conceitos do Ant e do Maven.**

Uma das primeiras coisas que podemos notar sobre o Gradle é que ele não usa arquivos XML, ao contrário do Ant ou do Maven.

Com o tempo, os desenvolvedores ficaram cada vez mais interessados em ter e trabalhar com uma linguagem específica de domínio – o que, em termos simples, lhes permitiria resolver problemas em um domínio específico usando uma linguagem feita sob medida para esse domínio em particular.

Isso foi adotado pelo Gradle, que usa uma DSL baseada em Groovy ou Kotlin . **Isso resultou em arquivos de configuração menores e mais organizados, já que a linguagem foi projetada especificamente para resolver problemas de domínio específicos.** O arquivo de configuração do Gradle é chamado, por convenção, de build.gradle em Groovy ou build.gradle.kts em Kotlin.

Note que o Kotlin oferece melhor suporte de IDE do que o Groovy para autocompletar e detecção de erros.

Aqui está um exemplo de um arquivo build.gradle para o mesmo projeto Java simples com a classe principal HelloWorld de antes:

```
apply plugin: 'java'


repositories {
    mavenCentral()
}


jar {
    baseName = 'gradleExample'
```

```
    version = '0.0.1-SNAPSHOT'
}

dependencies {
    testImplementation 'junit:junit:4.12'
}Cópia
```

Podemos compilar o código executando o seguinte comando:

```
gradle classesCópia
```

Em sua essência, o Gradle fornece intencionalmente pouca funcionalidade. **Os plugins adicionam todos os recursos úteis.** Em nosso exemplo, estávamos usando o plugin Java , que nos permite compilar código Java e outros recursos valiosos.

O Gradle deu o nome de "tasks" às suas etapas de construção, em oposição aos "targets" do Ant ou às "phases" do Maven. Com o Maven, usávamos o Apache Maven Dependency Plugin, cujo objetivo específico era copiar as dependências para um diretório específico. Com o Gradle, podemos fazer o mesmo usando tasks:

```
task copyDependencies(type: Copy) {
    from configurations.compile
    into 'dependencies'
}Cópia
```

Podemos executar essa tarefa executando o seguinte comando:

```
gradle copyDependencies
```

5. Conclusão

Neste artigo, apresentamos Ant, Maven e Gradle – três ferramentas de automação de compilação em Java.

Não é surpresa que o Maven detenha a maior parte do mercado de ferramentas de compilação atualmente.

O Gradle, no entanto, tem tido boa aceitação em bases de código mais complexas, pelos seguintes motivos:

- Muitos projetos de código aberto, como o Spring, já o utilizam.
- É mais rápido que o Maven na maioria dos cenários, graças às suas compilações incrementais.
- Oferece **serviços avançados de análise e depuração.**

No entanto, o Gradle parece ter uma curva de aprendizado mais acentuada, especialmente se você não estiver familiarizado com Groovy ou Kotlin.