Panoramica delle funzionalità – Parte II

Cosa imparerai

- Quali sono le principali funzionalità di Maven





Panoramica delle funzionalità

Il file POM

POM è l'acronimo di **Project Object Model** ed è un file XML (chiamato appunto **pom.xml**)

Il POM contiene tutte le informazioni necessarie di un progetto e le configurazioni dei plugin da utilizzare durante il processo di build.

```
<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 https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>it.test.myproject</groupId>
  <artifactId>my-first-webapp</artifactId>
  <version>1.0</version>
</project>
```

Panoramica delle funzionalità

Gestione delle dipendenze di progetto

Una delle principali funzionalità di Maven è la gestione delle dipendenze di un progetto, cioè le librerie di cui necessita il nostro progetto per funzionare correttamente (ad es. log4j, apache-commons,…)

Nel file POM è possibile specificare la lista delle risorse necessaria alla build del progetto.

Se una risorsa è esterna (ad es. log4j), Maven la scarica in automatico dai repository che abbiamo configurato e le utilizza durante la build del progetto. Se la risorsa richiesta ha bisogno di altre risorse (dipendenza transitiva), Maven le scarica in autonomia e le utilizza durante la build.

```
<project xmlns="…>
  …
  <dependencies>
    <dependency>
      <groupId>org.apache.commons</groupId>
      <artifactId>commons-lang3</artifactId>
      <version>3.11</version>
      <type>jar</type>
      <scope>compile</scope>
    </dependency>
    ...
  </dependencies>
</project>
```

Panoramica delle funzionalità

```
<project xmlns="…>
  …
  <dependencies>
    <dependency>
      <groupId>org.apache.commons</groupId>
       <artifactId>commons-lang3</artifactId>
       <version>3.11</version>
       <type>jar</type>
       <scope>compile</scope>
    </dependency>
    …
  </dependencies>
</project>
```

groupId, artifactId, version: come già specificato, identificano univocamente una risorsa e la versione di cui necessitiamo.

type: identifica il tipo di risorsa richiesta (se non presente viene utilizzato il valore predefinito, ovvero jar)

Panoramica delle funzionalità

scope: indica per quale fase della build è necessaria la libreria (compilazione e runtime, test, ecc…). Sono disponibili cinque ambiti:

- **compile**: specifica che la risorsa è necessaria per la compilazione. È il valore predefinito se lo *scope* non viene indicato; una dipendenza con *scope* «*compile*» viene propagata ai progetti dipendenti.
- **provided**: specifica che la risorsa è necessaria per la compilazione ed è fornita dal JDK o dal server runtime o container (ad es. Tomcat, TomEE, ecc…) su cui verrà installata;
- **runtime**: specifica che la risorsa è necessaria solo per l'esecuzione (non per la compilazione);
- **test**: specifica che la libreria è necessaria durante la fase di testing (ad es. la dipendenza da junit ha generalmente questo *scope*…);
- **system**: simile al *provided*, solo che in questo caso il programmatore deve indicare, tramite il tag *systemPath*, dove si trova la risorsa. In questo caso Maven si aspetta che la risorsa esista e non viene cercata in un repository.

```
<project xmlns="…>
  …
  <dependencies>
    <dependency>
      <groupId>org.apache.commons</groupId>
      <artifactId>commons-lang3</artifactId>
      <version>3.11</version>
      <type>jar</type>
      <scope>compile</scope>
    </dependency>
    ...
  </dependencies>
</project>
```

Panoramica delle funzionalità

Definizione di proprietà tramite le properties
Le properties possono essere associate a delle variabili globali, utilizzate all'interno del POM tramite la notazione *${nome-property}*.

Ad esempio, Java (*utilizzando il comando javac NDR*) consente di compilare un progetto in diverse versioni utilizzando i parametri *-source* e *-target*. Nel POM possiamo specificare due properties per indicare a Maven di compilare (tramite il suo compiler plugin predefinito) i sorgenti nella versione indicata.

```
<project…>
  ...
  <properties>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <liferay.version>6.2.5</liferay.version>
    <plugin.type>portlet</plugin.type>
    ...
</properties>
  ...
</project>
```

Le properties sono molto comode anche quando abbiamo più progetti (ognuno con il suo file POM) che sono gestiti attraverso un Super POM (un file POM che specifica le regole generali da usare per ogni progetto).

Di cosa abbiamo parlato in questa lezione

- Quali sono le principali funzionalità di Maven