Panoramica delle funzionalità – Parte I

Cosa imparerai

- Quali sono le principali funzionalità di Maven



Come funziona Maven

Per comprendere meglio Maven e la logica di funzionamento, è necessario partire dai componenti principali.

- **File POM (Project Object Model)**: è un file XML che contiene tutte le informazioni del progetto (definizione, dipendenze, test…);
- **Plugin**: sono la caratteristica centrale di Maven e consentono l'esecuzione di una serie di operazioni (ad es. compilazione, deploy, packaging, test…);
- **Goal**: è un'azione che può essere eseguita sul progetto. Il goal è direttamente associato ad un plugin;
- **Repository**: è una directory contenente le librerie (tipicamente i file jar). Il repository può essere locale o remoto (questo è un repository remoto: https://mvnrepository.com/).

Come funziona Maven

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>
```

NOTA: **modelVersion contiene 4.0.0**. Questa è attualmente l'unica versione POM supportata ed è sempre richiesta.

Come funziona Maven

Gli elementi indispensabili di un POM sono: **groupId – artifactId – version**. Sono tutti campi obbligatori.

I tre campi insieme consentono di identificare univocamente un progetto in un repository.

```
<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>
```

groupId: generalmente identifica univocamente un'azienda o un progetto. Ad esempio, se creo un progetto myproject, potrò avere ome groupId *it.test.myproject* o semplicemente *myproject*.

Il *groupId* non deve corrispondere necessariamente alla struttura dei package di un progetto anche se è una buona pratica da seguire…

Quando una risorsa (ad es. un jar) viene archiviata in un repository, Maven gestisce il groupId allo stesso modo di come vengono gestite le classi Java nei package nel file system, ovvero raggruppando i file in directory (ad es. *it.test.myproject* corrisponderà alla directory *it* che conterrà la directory *test* che ecc…)

Come funziona Maven

```
<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>
```

artifactId: L'artifactId rappresenta generalmente il nome del progetto. (ad es. se creo una webapp che si chiama *my-first-webapp*, l'artifactId sarà *my-first-webapp*).

L'artifactId, unitamente al groupId, crea una chiave che dovrebbe (e dico dovrebbe! ☺) identificare univocamente questo progetto da ogni altro progetto al mondo.

Quindi, attenzione a come definite groupId ed artifactId!

Come funziona Maven

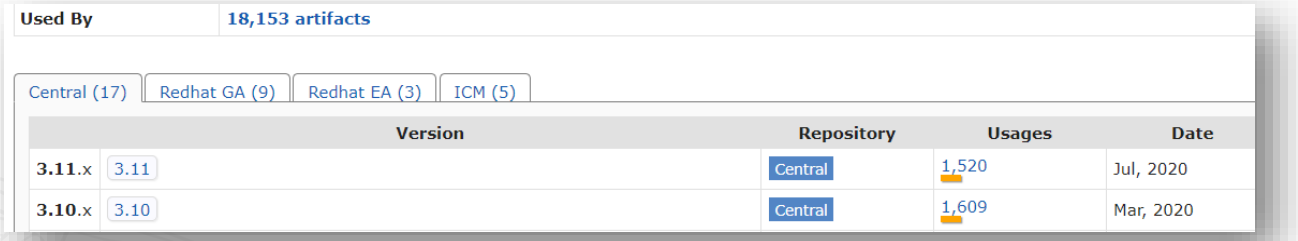
```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

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

version: se la coppia *groupId:artifactId* identifica univocamente un singolo progetto, *version* definisce di quale versione del progetto stiamo parlando.

In questo contesto, l'elemento *version* definisce la versione del pacchetto che stiamo realizzando.

Se utilizzato per includere una risorsa all'interno del nostro progetto (tipicamente un jar), l'elemento *version* definisce la versione della risorsa da includere (ad es. commons-lang versione 3.11, 3.10, ecc…).



Used By	18,153 artifacts			
Central (17)	Redhat GA (9)	Redhat EA (3)	ICM (5)	
Version		Repository	Usages	Date
3.11.x	3.11	Central	1,520	Jul, 2020
3.10.x	3.10	Central	1,609	Mar, 2020

Come funziona Maven

```
<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>

  <packaging>war</packaging>
</project>
```

packaging: definisce il tipo di risorsa che deve essere prodotto dal POM (jar, war, ear,…). Se non viene specificato, Maven presume che la risorsa sia quella predefinita, ovvero un jar.

Di cosa abbiamo parlato in questa lezione

- Quali sono le principali funzionalità di Maven