ATTSW Exam: Gradle come sostituto di Maven

Gabriele Puliti - `5300140` - *gabriele.puliti@stud.unifi.it*

December 2017

Indice

1 Introduzione: Gradle

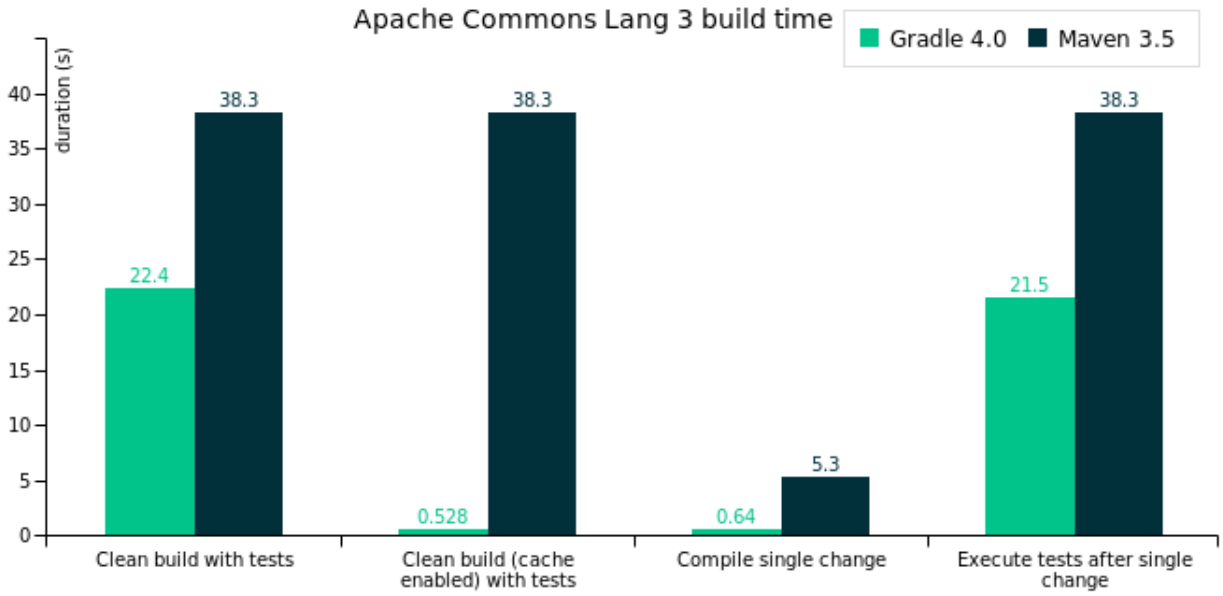
Gradle è un progetto open source che fornisce un tool di build automation, che può essere un ottimo sostituto di Maven. Offre un modello in grado di sostenere l'intero ciclo di vita dello sviluppo del software ed è stato progettato per supportare build automation attraverso più linguaggi e piattaforme. Nel nostro caso considereremo questo tool per lo sviluppo di software Java.

1.1 Differenze tra Gradle e Maven

Ci sono molte differenze tra questi due tools: flessibilità, performance, gestione delle dipendenze e molto altro. La configurazione di Gradle in un progetto ha una convenzione molto più facile e comprensibile rispetto alla tediosa e a volte impossibile configurazione del pom.xml di Maven, anche se entrambi usano dei metodi di miglioramento della velocità di esecuzione delle build. Grandle usufruisce di:

- **Incrementality:** evitando il lavoro di monitoraggio dei task di I/O eseguendo solo il necessario e quando possibile processare solo i files che sono cambiati;
- **Build Cache:** utilizza un sistema di cache riusando gli outputs di altre build Gradle con gli stessi inputs;
- **Deamon:** sfrutta un long-lived process che mantiene tutte le informazioni in memoria.

Queste 3 caratteristiche rendono Gradle molto veloce, ad esempio una build Gradle con Maven verrebbe completata con un tempo 3 volte maggiore. Tutto questo è anche possibile grazie a un sistema di esecuzioni parallele di task e intra-task.



1.2 Installazione

L'istallazione di Gradle può essere fatta in più modi: tramite installazione manuale o utilizzando un package manager (tutte le informazioni possono essere trovate in **questo link**). Personalmente consiglio l'utilizzo del software development kit manager **SDKMAN!** che non solo permette l'installazione molto facilitata di Gradle, ma anche della JVM e di tanti altri tools.

1.2.1 installazione tramite SDKMAN!

L'installazione si basa su 2 semplici comandi:

```
$ curl -s "https://get.sdkman.io" | bash

$ source "$HOME/.sdkman/bin/sdkman-init.sh"
```

A questo punto se tutto è andato a buon fine SDKMAN! è stato installato correttamente, è possibile verificarlo digitando il comando su terminale:

```
$ sdk version
```

l'output risultante dovrebbe essere qualcosa del tipo:

```
SDKMAN 5.5.15+284
```

Ora è possibile procedere con l'installazione di Gradle. Prima di tutto visualizziamo la lista delle versioni di Gradle:

```
$ sdk list gradle
```

L'output corrispondente sarà:

```
================================================================================
Available Gradle Versions
================================================================================
     4.4.1               4.2-rc-2            3.0                 2.10
     4.4-rc-6            4.2-rc-1            2.9                 2.1
     4.4-rc-5            4.2                 2.8                 2.0
     4.4-rc-4            4.1                 2.7                 1.9
     4.4-rc-3            4.0.2               2.6                 1.8
     4.4-rc-2            4.0.1               2.5                 1.7
     4.4-rc-1            4.0                 2.4                 1.6
     4.4                 3.5.1               2.3                 1.5
     4.3.1               3.5                 2.2.1               1.4
     4.3-rc-4            3.4.1               2.2                 1.3
     4.3-rc-3            3.4                 2.14.1              1.2
     4.3-rc-2            3.3                 2.14                1.12
     4.3-rc-1            3.2.1               2.13                1.11
     4.3                 3.2                 2.12                1.10
     4.2.1               3.1                 2.11                1.1
================================================================================
+ - local version
* - installed
> - currently in use
================================================================================
```

La versione che vogliamo installare è quella più recente che in questo caso è la 4.4.1, possiamo quindi eseguire il comando:

```
$ sdk install gradle 4.4.1
```

appena il download e l'installazione sarà finita possiamo verificare il completamento tramite:

```
$ gradle -v
```

che non solo stamperà su terminale la versione di Gradle, ma anche:

- Groovy (linguaggio di programmazione usato per scrivere i file di configurazione)

- Ant (software usato per le build delle Java applications)
- Java Virtual Machine
- sistema operativo in uso

se l'output ha queste informazioni allora Gradle è stato completamente installato. SDKMAN! si preoccupa anche di creare la variabile $GRADLE_HOME che è possibile visualizzare con il comando

```
$ echo $GRADLE_HOME
```

Se ci sono errori di tipo Java, i problemi possono essere:

- Gradle non riesce a trovare la jdk, problema risolvibile installando java con sdkman con il comando

  ```
  $ sdk install java <versione>
  ```

- Java è aggiornato alla versione 9 o superiori (infatti attualmente Gradle 4.4.1 non è aggiornato per versioni superiori alla 8), basterà fare un downgrade ad una versione precedente (possibile farlo anche tramite SDKMAN!).

In entrambi i casi sarà necessario anche comunicare al sistema la versione da usare:

```
$ sdk dafault java <versione_installata>
```

per essere sicuri che è stata installata la giusta versione di java possiamo controllare gli outputs dei seguenti comandi:

- `$ echo $JAVA_HOME`
- `$ java -version`

il primo comando dovrà restituire in output il giusto percorso della JVM installata, il secondo serve a controllare la versione java attualmente in uso.

2 Tutorial

Propongo nei seguenti paragrafi alcuni esempi.

2.1 Primo tutorial: gestione dei tasks

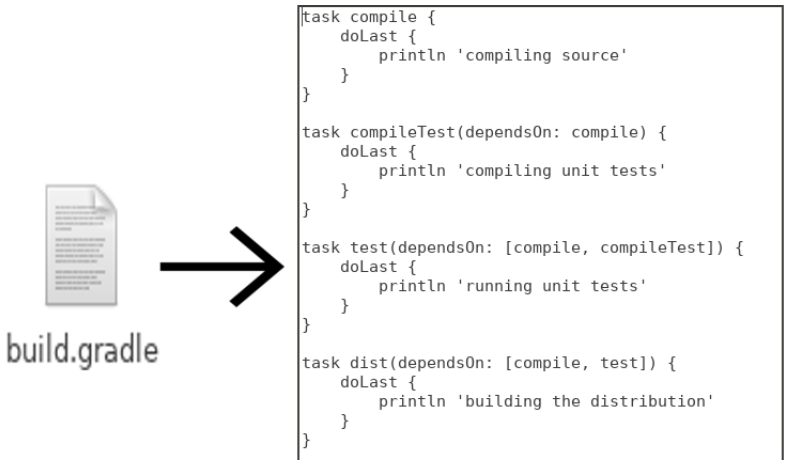
Come in Maven ci sono i goals, in Gradle ci sono i task, ognuno dei quali ha il suo scopo definito nella sua implementazione. Gradle si basa su build multi-task, il primo esempio si basa sulla comprensione del funzionamento delle build Maven.

2.1.1 Task e Task dependency

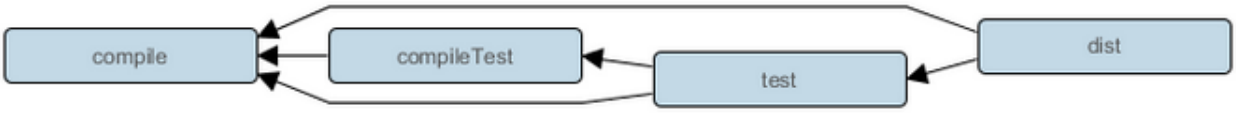
Creiamo una cartella in cui inserire il file di configurazione della build Gradle chiamato build.gradle e modifichiamo il suo contenuto usando un editor di testo:

```
task compile {
    doLast {
        println 'compiling source'
    }
}

task compileTest(dependsOn: compile) {
    doLast {
        println 'compiling unit tests'
    }
}

task test(dependsOn: [compile, compileTest]) {
    doLast {
        println 'running unit tests'
    }
}

task dist(dependsOn: [compile, test]) {
    doLast {
        println 'building the distribution'
    }
}
```

a questo punto dovremmo avere una cosa di questo tipo:



abbiamo creato un albero di dipendenze tra tasks di questo tipo:



è possibile fare diverse build con questa configurazione:

- `$ gradle compile`
- `$ gradle compileTest`
- `$ gradle test`
- `$ gradle dist`
- una combinazione qualunque di 2 o più task.

Possiamo notare che anche se eseguiamo la build:

```
$ gradle compile test
```

il task **compile** verrà eseguito solo una volta:

```
> Task :compile
compiling source

> Task :compileTest
compiling unit tests

> Task :test
running unit tests


BUILD SUCCESSFUL in 0s
3 actionable tasks: 3 executed
```

2.1.2 Escludere task da una build

È possibile escludere un task di una build, aggiungendo come argomento il task da escludere preceduto da -x:

```
$ gradle <task_da_eseguire> -x <task_da_escludere>
```

questo viene usato al fine di eliminare un task inutile per lo scopo della build che abbiamo intenzione di eseguire. Riprendendo l'esempio del paragrafo precedente se andiamo ad eseguire la build:

```
$ gradle dist
```

vediamo che vengono eseguiti tutti i tasks, compresi i tasks **test** e **compileTest**, supponendo di voler fare solo la build del sorgente possiamo scrivere:

```
$ gradle dist -x test
```

l'output risultante sarà:

```
> Task :compile
compiling source

> Task :dist
building the distribution


BUILD SUCCESSFUL in 0s
2 actionable tasks: 2 executed
```

Il task **compileTest** non viene eseguito perchè dipendenza di **test**, ma non di **dist** (vedi pag. 2), escludendo il primo quindi non è necessario eseguire il task **compileTest**. Nel file di configurazione della build è possibile inserire anche un descrizione inserendo in testa al file build.gradle la seguente riga:

```
description = 'Descrizione'
```

Questo è importante per progetti multi-builds consentendo di avere una descrizione di ogni singola build.

2.1.3 Abbreviazione del nome del task

è possibile abbreviare il nome del task da eseguire stando però attenti ad identificare unicamente il task che vogliamo eseguire, per esempio se volessi eseguire il task **compileTest** potrei farlo semplicemente con il comando:

```
$ gradle comTes
```

considerando i task creati precedentemente notiamo che il task è univocamente identificato.

2.1.4 Selezionare la build da eseguire

consideriamo che esista in una subdirectory chiamata subdir una build chiamata subbild.gradle, partendo dalla directory root è possibile eseguire questa build eseguendo il comando:

```
$ gradle -b subdir/subbuild.gradle <task_da_eseguire>
```

è possibile anche indicare direttamente la project directory da usare, nel nostro caso indicheremo subdir:

```
$ gradle -p subdir
```

2.1.5 Forzare l'esecuzione di un task

a causa della Gradle cache è possibile che un task o più di uno non vengano eseguiti perchè marcati come UP-TO-DATE (anche se dalla versione Gradle 4.0 non viene più mostrato in output), in questo caso è possibile forzarne l'esecuzione con:

```
$ gradle --rerun-tasks <tasks_da_eseguire>
```

2.1.6 Ottenere informazioni generali

come già detto precedentemente nel caso di progetti multi-builds è necessario avere una descrizione di ogni file di configurazione della build, per ottenere le informazioni sul progetto corrispondente alla build è possibile eseguire il task `projects`:

```
$ gradle projects
```

2.1.7 Ottenere informazioni sui tasks

è possibile ricavare una lista dei tasks default eseguendo la build del task `tasks`:

```
$ gradle tasks
```

possiamo notare che l'output non mostra tutti i task, ma solo quelli predefiniti:

```
> Task :tasks

------------------------------------------------------------
All tasks runnable from root project - Descrizione
------------------------------------------------------------

Build Setup tasks
-----------------
init - Initializes a new Gradle build.
wrapper - Generates Gradle wrapper files.

Help tasks
----------
buildEnvironment - Displays all buildscript dependencies declared in root project 'first'.
components - Displays the components produced by root project 'first'. [incubating]
dependencies - Displays all dependencies declared in root project 'first'.
dependencyInsight - Displays the insight into a specific dependency in root project 'first'.
dependentComponents - Displays the dependent components of components in root project 'first'. [incubati
help - Displays a help message.
model - Displays the configuration model of root project 'first'. [incubating]
projects - Displays the sub-projects of root project 'first'.
properties - Displays the properties of root project 'first'.
tasks - Displays the tasks runnable from root project 'first'.

To see all tasks and more detail, run gradle tasks --all

To see more detail about a task, run gradle help --task <task>


BUILD SUCCESSFUL in 0s
1 actionable task: 1 executed
```

come dice l'output, per visualizzare la lista di tutti i tasks è necessario eseguire la build del task `tasks` con argomento –all:

```
$ gradle tasks --all
```

il risultato sarà molto più specifico rispetto al precedente. Possiamo notare che a differenza degli altri tasks quelli che abbiamo creato noi sono sprovvisti di una descrizione, per aggiungerla basterà inserire un campo `description` all'interno della definizione del task:

```
task dist(dependsOn: [compile, test]) {
    description = 'Build distribution'
    doLast {
        println 'building the distribution'
    }
}
```

rieseguendo il comando sopra otterremo una descrizione anche per il `dist`. Per ottenere informazioni più specifiche è possibile usare la build del task help seguito da –task e il nome del task:

```
$ gradle help --task <task>
```

quello che vedremo sarà un riepilogo generale del task.

2.2 Secondo tutorial: Gradle Wrapper

Molto spesso prima di poter usufruire di uno strumento di sviluppo è necessaria una installazione. Gradle mette a disposizione uno script che permette di usare tutte le sue funzionalità evitando di installare Gradle su tutte le macchine di sviluppo, questo strumento viene chiamato Gradle Wrapper. Se in un progetto è stato settato il Wrapper è possibile eseguire le builds direttamente dalla root del progetto con il comando:

```
$ ./gradlew <task>
```

Se più persone lavorano a un progetto può capitare che ci siano differenze tra le versioni di uno strumento, nel caso del wrapper non è possibile sbagliare perchè la sua versione è insita durante la sua creazione o durante il suo upgrade (o downgrade). Quindi è sempre consigliato l'uso del wrapper e lasciare tutte le sue informazioni anche nella cartella principale del VCS usato.

2.2.1 Aggiungere il Wrapper ad un progetto

Per poter usufruire del wrapper è necessario eseguire il comando:

```
$ gradle wrapper
```

è possibile trovare le versioni del Wrapper nella directory locale in cui è stato installato Gradle solitamente *$HOME/.gradle/wrapper/dists*. Il comando precedente creerà 4 files:

- **gradlew**: script del wrapper per sistemi Unix
- **gradlew.bat**: file batch per sistemi Windows
- **gradle/wrapper/gradle-wrapper.properties**: proprietà del wrapper
- **gradle/wrapper/gradle-wrapper.jar**: file jar del wrapper

per usare una versione Gradle specifica è possibile usare 3 metodi:

1. eseguire il solito comando con l'aggiunta dell'argomento –gradle-version:

   ```
   $ gradle wrapper --gradle-version <numero_versione>
   ```

   oppure se già inserito il wrapper:

   ```
   $ ./gradlew wrapper --gradle-version <numero_versione>
   ```

   per esempio se volessimo passare dalla versione 4.4.1 alla versione 2.0 basterà eseguire il comando:

   ```
   $ ./gradlew wrapper --gradle-version 2.0
   ```

   per aggiornare la versione gradle usata dal wrapper basterà eseguire:

   ```
   $ ./gradlew wrapper
   ```

2. modificare direttamente il file gradle-wrapper.properties in cui nell'ultima riga ci sarà la versione usata dal Wrapper:

   ```
   distributionUrl=https\://services.gradle.org/distributions/gradle-4.4.1-bin.zip
   ```

   per passare alla versione 2.0 possiamo modificare questa riga con:

```
distributionUrl=https\://services.gradle.org/distributions/gradle-2.0-bin.zip
```

3. infine possiamo modificare la definizione del task `wrapper` della build di gradle, per farlo è necessario creare (se non è già stato fatto) il file build.gradle e aggiungere:

   ```
   task wrapper(type: Wrapper) {
       gradleVersion = '2.0'
   }
   ```

   ed eseguire nuovamente il task `wrapper`:

   ```
   $ ./gradlew wrapper
   ```

in ogni caso possiamo visualizzare la versione usata dal wrapper con il comando:

```
$ ./gradlew --version
```

2.3 Terzo tutorial: Dependency Management

Una delle parti più importanti di uno strumento di questo tipo è la gestione delle dipendenze che si divide in 2 parti: incoming files e outgoing files. Infatti Gradle ha bisogno di conoscere di cosa il nostro progetto ha bisogno per poter essere compilato ed eseguito, queste vengono chiamate dipendenze (dipendencies) che in questo caso sono gli incoming files. Gli outgoing files sono invece tutto ciò che il tuo progetto produce chiamati anche pubblicazioni (pubblications). Il Dependency Manager di Gradle permette di scaricare le dipendenze del progetto da diversi remote repository specificando direttamente il nome e la versione della dipendenza voluta.

2.3.1 Dichiarazione delle dipendenze

Prima di tutto è necessario indicare in che linguaggio il nostro progetto viene rilasciato (consideriamo d'ora in poi solo il caso di Java), per farlo aggiungiamo in testa al file build.gradle:

```
apply plugin: 'java'
```

A questo punto per poter usufruire di una dipendenza è necessario specificare da dove Gradle deve andare a prenderla, dobbiamo quindi indicare il repository remoto di riferimento. Se per esempio vogliamo che il nostro repository di riferimento sia Maven allora dobbiamo aggiungere al build.gradle:

```
repositories {
    mavenCentral()
}
```

In questo modo tutte le dipendenze che andremo a indicare successivamente saranno riferimenti alle pubblicazioni su MavenCentral. La dichiarazione delle dipendenze deve essere inserita nel tag `dependencies` nel build.gradle file, per esempio vogliamo avere junit 4.12 come dipendenza al nostro progetto Gradle allora dobbiamo aggiungere:

```
dependencies {
    testCompile group: 'junit', name: 'junit', version: '4.12'
}
```

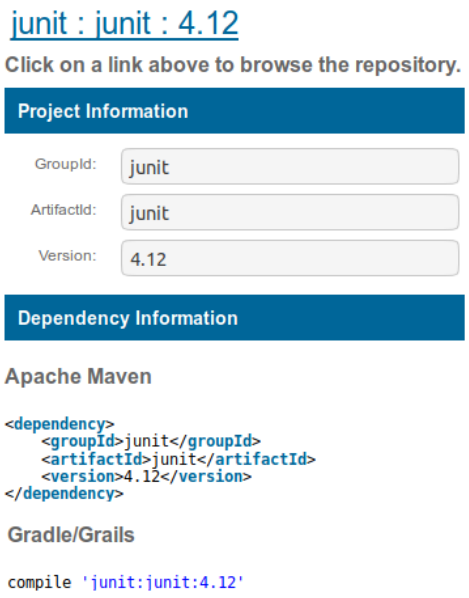
osserviamo che nella dichiarazione ci sono 4 diversi indicatori:

- `testCompile` indica a che tipo di build deve fare riferimento, in questo caso questa dipendenza sarà importata durante la compilazione dei test;
- `group, name, version` corrispondono rispettivamente al groupId, artifactId e al version definiti su Maven.

esiste un modo molto più diretto per indicare una dipendenza, considerando sempre la dipendenza junit possiamo scrivere:

```
dependencies {
    testCompile 'junit:junit:4.12'
}
```

ha lo stesso significato precedente ma ha una forma più compatta, forma che adotta anche la documentazione Maven.



Possiamo notare ora la differenza sostanziale della configurazione delle dipendenze tra il pom.xml di Maven e la build.gradle di Gradle. A questo punto possiamo scaricare le dipendenze, per farlo eseguiamo il comando gradle (usando il wrapper):

```
$ ./gradlew dependencies
```

L'output restituirà la lista di tutti i task con le relative dipendenze (se ce ne sono), nel caso la dipendenza richiesta non si trova nel progetto provvederà a scaricarla.