Gradle

Victor Herrero Cazurro

Table of Contents

Introducción

Es una herramienta de construcción como Maven o ANT, basada en un DSL Groovy.

La documentacion del DSL se encuentra aquí y una guia de usuario aquí

En gradle se definen principalmente tres tipos de objetos

- Projects
- Task
- Settings

Instalacion

Descargar la distribucion de aquí

Definir la variable de entorno **GRADLE_HOME** con la siguiente ruta **<directorio de descompresion de gradle>**

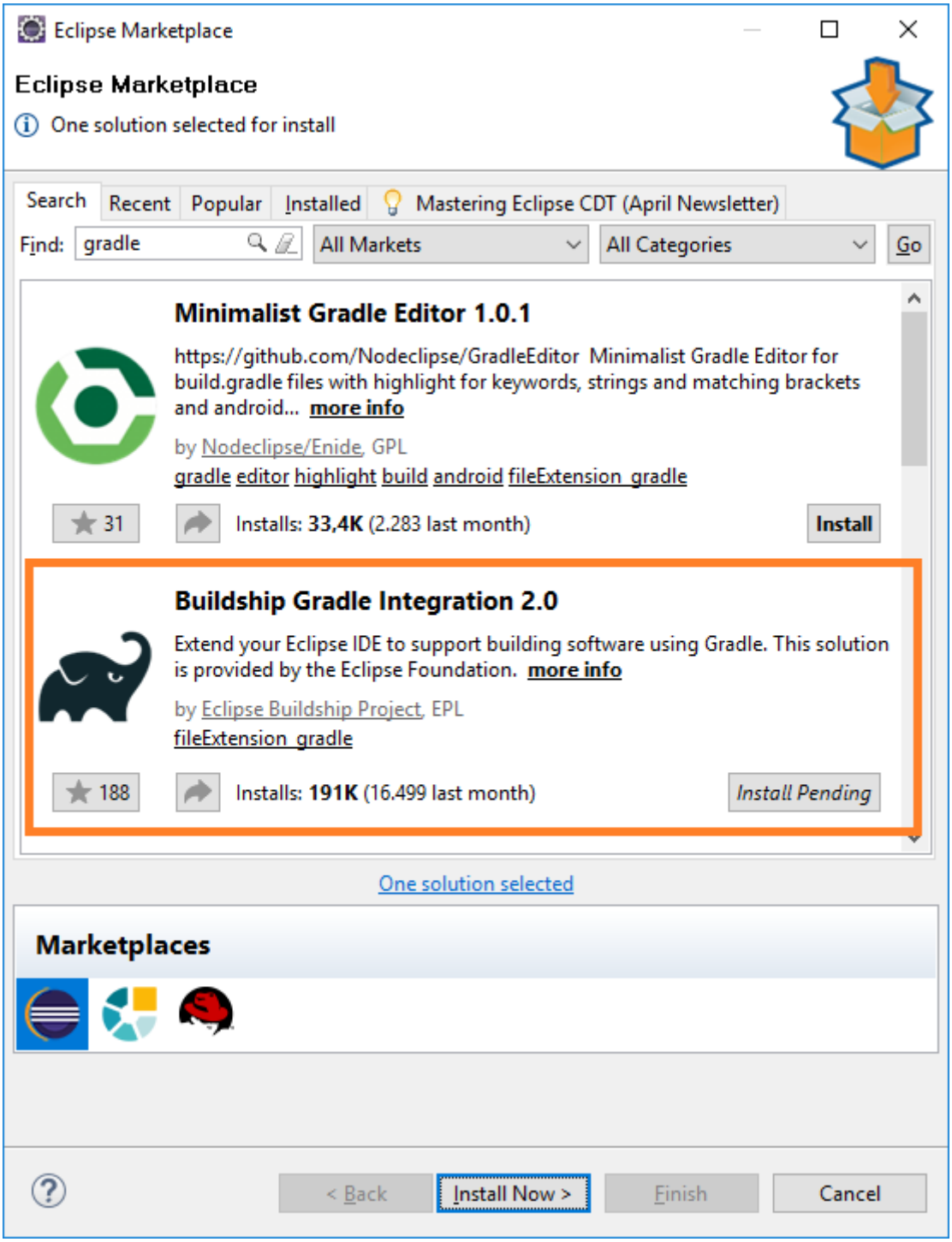
Definir la variable de entorno **GRADLE_HOME** con la siguiente ruta **<directorio de descompresion de gradle>**

Añadir a la variable de entorno PATH la siguiente ruta **GRADLE_HOME/bin**

IDEs

Gradle se puede emplear desde varios IDEs, incluidos los mas habituales: eclipse, netbeans, sts, intellij, ...

Para eclipse, el plugin recomendado, que habrá que añadir es **BuildShip Gradle Integration**



Bases de los scripts de construcción

Los proyectos gestionados con **Gradle**, se basan en la configuración definida en los ficheros **.gradle**, por defecto el fichero de configuración se llama **build.gradle**, aunque esto se puede cambiar, no es recomendable por favorecer la legibilidad.

A estos ficheros, se les denomina ficheros de **Script**, y están escritos en **groovy**

La pieza más pequeña que compone un script de gradle es un **task**, una tarea.

Ejecucion

Se ejecutan los scripts haciendo referencias a las **task**.

Se definen una serie de task por defecto, algunas son

- **init** → Crea un nuevo proyecto gradle.
- **dependencies** → Muestra las dependencias del proyecto.
- **help** → Muestra la ayuda.
- **projects** → Muestra los subproyectos del proyecto principal.
- **properties** → Muetra las propiedades del proyecto.
- **tasks** → lista las tareas disponibles.

Para ejecutarlas, desde la linea de comandos, se hace referencia al comando **gradle** y a la tarea o conjunto de tareas deseadas

```
>gradle tasks

>gradle tasks projects
```

El comando muestra una salida con información algo extendida, que puede acortarse con el modificador **-q**

```
>gradle -q tasks
```

Task

Para definir un **task**, se emplea la palabra reservada para el DSL **task**

```
task helloWorld {
    doLast {
        println 'Hello world!'
    }
}
```

Acciones

Las task de gradle, están formadas por acciones como **doLast**, en este caso es la última acción que se ejecutará asociada al **task**, y ademas es la accion por defecto, que tiene una sintaxis reducida

```
task helloWorld << {
    println 'Hello world!'
}
```

Existen más acciones como:

- dofirst
- onlyIf
- mustRunAfter
- hasProperty

Para invocar un **task** desde la consola de **Gradle** se emplea

```
gradle –q helloWorld
```

NOTE	El modificador -q indica a **gradle** que solo muestre el resultado de la ejecución de la tarea.

Propiedades

Se pueden añadir más propiedades a las tareas, empleando la propiedad **ext**.

```
task myTask {
    ext.myProperty = "myValue"
}

task printTaskProperties {
    doLast {
        println myTask.myProperty
    }
}
```

Tipos de Proyectos

Se pueden crear proyectos independientes, cada uno con su script de gradle, o se pueden establcer relaciones entre ellos, lo que se denomina un multiproyecto.

Para ello por defecto se supone que ls proyectos estaran definidos en forma de arbol de directorios

```
--Proyecto
-------Subproyecto
```

Cada uno con su propio **build.gradle**

Para relacionarlos, lo primero es añadir el **settings.gradle** en el principal, donde se hace referencia a los otros.

```
include 'project1', 'project2:child', 'project3:child1'
```

El método **include**, recibe como parametros paths de proyectos relativos al directorio donde se encuentra el fichero **settings.gradle**, el formato cambia las / o \ por **:** para estandarizar los distintos SO, por lo que 'services:api' equivale a '<root dir>/services/api'.

Solo es necesario indicar los proyectos finales, no hace falta indicar los intermedios, por lo que si se tienen distintos **build.gradle** definidos en un arbol de carpetas, al incluir **services:hotels:api**, se incluiran los proyectos: **services**, **services:hotels** y **services:hotels:api**.

Una vez definida la relación, en el **build.gradle** del principal, se pueden definir configuraciones comunes empleando **allprojects**

```
allprojects {
    apply plugin : 'java'
}
```

A partir de esa configuracion, cuando se lancen tareas configuradas como comunes, sobre el principal, se lanzaran sobre todos los subproyectos tambien, si ses desea lanzar la tarea sobre un subproyecto concreto, se puede emplear la sintaxis

```
>gradle project2:build
```

Si solo se desea afectar a los subporectos, se dispone de **subprojects**

```
subprojects {
    apply plugin : 'java'
}
```

Y si se quiere afectar a un proyecto en concreto

```
project(':api') {

}
```

Esta ultima configuracion equivale a definir en cada pryeco su propio **build.gradle**

Se pueden definir dependencias entre proyectos, de forma abreviada

```
dependencies {
    compile project(':common')
}
```

Tareas Complejas

Dependencia entre Tareas

Se puede hacer depender una tarea de otras, empleando la propiedad **dependsOn**, de tal forma que antes de ejecutar la tarea, se ejecuten todas aquellas de las que depende.

```
task hello {
    doLast {
        println 'Hello world!'
    }
}
task intro(dependsOn: hello) {
    doLast {
        println "I'm Gradle"
    }
}
```

Las tareas se ejecutan por completo, no se pueden anidar.

Herencia en las Tareas

Se pueden definir nuevas tareas, que hereden la configuracion de la tarea de un tipo predefinido

```
task hello(type: GreetingTask)
```

En el Api se proporciona

```
class GreetingTask extends DefaultTask {
    @TaskAction
    def greet() {
        println 'hello from GreetingTask'
    }
}
```

Se pueden definir tareas basandose en la clase **DefaultTask**

Ejecucion de tareas multiple

Se puede asignar a los métodos de las tareas, varios bloques de código, ejecutandose todos, en el orden en el que se asignen

Así pues con la siguiente definición

```
task hello {
    doLast {
        println 'Hello Earth'
    }
}
hello.doFirst {
    println 'Hello Venus'
}
hello.doLast {
    println 'Hello Mars'
}
hello {
    doLast {
        println 'Hello Jupiter'
    }
}
```

Y lanzando la tarea

```
>gradle hello
```

Se obtiene la salida

```
Hello Venus
Hello Earth
Hello Mars
Hello Jupiter
```

Plugins

Los plugins son proyectos independientes que ofrecen un conjunto de **task** a otros proyectos.

Para refereniar a un lugin, hay que referenciarlo desde le script.

```
apply plugin: <nombre del plugin>
```

Para crear plugins basta con definir una nueva clase que herede de **Plugin<Project>**

```
class GreetingPlugin implements Plugin<Project> {
    void apply(Project project) {
        project.task('hello') {
            doLast {
                println "Hello from the GreetingPlugin"
            }
        }
    }
}
```

Esta herencia, proporciona un método **apply**, que permite asociar al **project** nuevas **task**.

Para emplear el plugin desde el mismo script que lo define, basta con añadir

```
apply plugin: GreetingPlugin
```

Una vez se aplica el nuevo **plugin** las **task** definidas, están disponiles y se puede invocar dicha tarea

```
gradle -q hello
```

Java Plugin

Uno de los plugins mas empleado es el de java, para ñadirlo

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

```
Añade tareas para poder procesar proyectos java.
```

- **assemble** → Genera el jar del proyecto.
- **build** → Genera el jar del proecto y lo prueba.
- **buildDependents** → Assembles and tests this project and all projects that depend on it.
- **buildNeeded** → Assembles and tests this project and all projects it depends on.
- **classes** → Assembles main classes.
- **clean** → Deletes the build directory.
- **jar** → Assembles a jar archive containing the main classes.
- **testClasses** → Assembles test classes.

Se puede configurar el nombre del jar que se genera con las propiedades **archivesBaseName** y **version**

```
archivesBaseName = "miProyecto"
version = '0.0.1-SNAPSHOT'
```

War Plugin

Para añadirlo

```
apply plugin: 'war'
```

Permite generar un **war** en vez de un **jar** con el proyecto java, con la tarea **build**

```
gradle build
```

Jetty Plugin

Para añadirlo

```
apply plugin: 'jetty'
```

Permite arrancar un jetty y desplegar la aplicacion web

```
gradle jettyRunWar
```

Gestion de dependencias

Las dependencias en **Gradle**, al igual que en Maven, son las librerias de terceros que un proyecto necesita para ser compilado o ejecutado.

```
dependencies {
    compile group: 'org.hibernate', name: 'hibernate-core', version: '3.6.7.Final'
    testCompile group: 'junit', name: 'junit', version: '4.+'
}
```

Otro formato de definicion de las dependencias sería

```
dependencies {
    compile 'org.hibernate:hibernate-core:3.6.7.Final'
}
```

Las dependencias, se obtienen de repositorios

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

Ambitos

Se definen cuatro ambitos para las dependencias, según donde se necesiten y quien sea el responsable de que esten en el classpath.

- **compile** → La dependencia es necesaria para la compilacion del codigo del proyecto.

```
dependencies {
    compile group: 'org.hibernate', name: 'hibernate-core', version: '3.6.7.Final'
}
```

- **runtime** → La dependencia es necesaria para la ejecucion del proyecto.
- **testCompile** → La dependencia es necesaria para la compilacion de los test del proyecto.

```
dependencies {
    testCompile group: 'junit', name: 'junit', version: '4.+'
}
```

- **testRuntime** → La dependencia es necesaria para la ejecucion de los test.

Publicacion Local

Se proporciona un plugin **maven-publish**, que proporciona una tarea **publishToMavenLocal**, que permite generar en el repositorio local de maven <**userDir**>/**.m2/repository**, un nuevo artefacto con el contenido del proyecto

```
apply plugin: 'maven-publish'

publishing {
    publications {
        maven(MavenPublication) {
            groupId 'com.ejemplo.gradle'
            artifactId 'myprojecto'
            version '0.0.1-SNAPSHOT'

            from components.java
        }
    }
}
```

Publicacion Remota

El mismo plugin **maven-publish**, proporciona una tarea **publish**, que permite generar en un repositorio de maven configurado la distribucion del artefacto.

```
apply plugin: 'maven-publish'

publishing {
    publications {
        maven(MavenPublication) {
            groupId 'com.ejemplo.gradle'
            artifactId 'myprojecto'
            version '0.0.1-SNAPSHOT'

            from components.java
        }
    }
    repositories {
        maven {
            url "$buildDir/repo"
        }
    }
}
```

Ciclo de vida

Gradle define tres fases en su ciclo de vida

- **Initialization** → Como se soporta la construccion de multiples proyectos, en esta fase, se determina que proyectos participan del build. Creando una instancia de **Project** por cada uno de ellos.
- **Configuration** → En esta fase los scripts de todos los proyectos son ejecutados, que acaban definiendo los **Repository** y los **Task** disponibles.
- **Execution** → En esta fase se determina el conjunto de **Task** y el orden de ejecución y se ejecutan.

Inicializacion

Durante esta fase, se ejecutan los scripts definidos en el fichero **settings.gradle**, que genera un objeto **Settings**

Dado el fichero settings.gradle

```
println 'This is executed during the initialization phase.'
```

y el fichero build.gradle

```
println 'This is executed during the configuration phase.'

task configured {
    println 'This is also executed during the configuration phase.'
}

task test {
    doLast {
        println 'This is executed during the execution phase.'
    }
}

task testBoth {
    doFirst {
      println 'This is executed first during the execution phase.'
    }
    doLast {
      println 'This is executed last during the execution phase.'
    }
    println 'This is executed during the configuration phase as well.'
}
```

Ejecutando la tarea

```
> gradle test testBoth
```

Se obtiene la siguiente traza.

```
This is executed during the initialization phase.
This is executed during the configuration phase.
This is also executed during the configuration phase.
This is executed during the configuration phase as well.
:test
This is executed during the execution phase.
:testBoth
This is executed first during the execution phase.
This is executed last during the execution phase.

BUILD SUCCESSFUL

Total time: 1 secs
```