eBook Gratuit

APPRENEZ gradle

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#gradle

Table des matières

À propos

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: gradle

It is an unofficial and free gradle ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official gradle.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapitre 1: Démarrer avec gradle

Remarques

Gradle est un outil de création open-source à usage général. Il est populaire dans la communauté Java et est l' outil de construction préféré pour Android .

Caractéristiques de Gradle mises en évidence

- Les scripts de génération déclaratifs *sont du* code écrit en Groovy ou Kotlin .
- Beaucoup de plugins de base et de communauté utilisant une approche flexible basée sur des conventions
- La génération incrémentielle de telle sorte que les tâches dont les dépendances n'ont pas changé ne sont pas réexécutées.
- Résolution de dépendance intégrée pour Maven et Ivy . Les plugins fournis fournissent une résolution de dépendance à partir d'autres `repositories` tels que npm .
- Des versions multi-projets de première classe.
- Intégration avec d'autres outils de construction comme Maven , Ant et autres.
- Créez des analyses qui augmentent la capacité des développeurs à collaborer et à optimiser les versions de Gradle.

Plus d'information

Si vous souhaitez en savoir plus sur les fonctionnalités de Gradle, consultez la section Présentation du Guide de l'utilisateur de Gradle .

Si vous voulez essayer, Gradle peut consulter les guides ici . Vous pouvez parcourir un guide de démarrage rapide Java, apprendre à utiliser Gradle pour la première fois et migrer depuis un autre outil de génération.

Examples

Gradle Installation

Configuration requise: JDK ou JRE Java installé (version 7 ou supérieure pour la version Gradle 3.x)

Étapes d'installation:

1. Télécharger la distribution Gradle sur le site officiel
2. Déballer le ZIP
3. Ajoutez la variable d'environnement `GRADLE_HOME` . Cette variable doit pointer vers les fichiers décompressés de l'étape précédente.
4. Ajoutez `GRADLE_HOME/bin` à votre variable d'environnement `PATH` pour pouvoir exécuter Gradle à partir de l'interface de ligne de commande (CLI)

5. Testez votre installation Gradle en tapant `gradle -v` dans la CLI. La sortie doit contenir la version de Gradle installée et les détails de configuration de Gradle actuels

Plus d'informations peuvent être trouvées dans le guide d'utilisation officiel

Installation avec homebrew sur OS X / macOS

Les utilisateurs de homebrew peuvent installer graduellement en cours d'exécution

```
brew install gradle
```

Installation avec SdkMan

Les utilisateurs de SdkMan peuvent installer Gradle en exécutant:

```
sdk install gradle
```

Installer une version spécifique

```
sdk list gradle
sdk install gradle 2.14
```

Changer de version

```
sdk use gradle 2.12
```

Installer le plugin Gradle pour Eclipse

Voici les étapes à suivre pour installer le plug-in Gradle dans Eclipse:

1. Ouvrez Eclipse et allez dans **Aide** -> **Eclipse Marketplace**
2. Dans la barre de recherche, entrez **bâtiment** et appuyez sur Entrée
3. Sélectionnez **"Buildship Gradle Integration 1.0"** et cliquez sur **Installer**
4. Dans la fenêtre suivante, cliquez sur **Confirmer**
5. Ensuite, **acceptez** les termes et la licence de l'accord, puis cliquez sur **Terminer**
6. Après l'installation, Eclipse devra redémarrer, cliquez sur **Oui**

Bonjour le monde

Les tâches Gradle peuvent être écrites à l'aide du code Groovy à partir d'un fichier build.gradle de projets. Ces tâches peuvent alors être exécutées en utilisant `> gradle [taskname]` sur le terminal ou en exécutant la tâche depuis un IDE tel qu'Eclipse.

Pour créer l'exemple Hello World dans gradle, nous devons définir une tâche qui imprimera une chaîne sur la console à l'aide de Groovy. Nous utiliserons `printLn` de Groovy pour appeler la méthode `System.out.printLn` de Java pour imprimer le texte sur la console.

build.gradle

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

Nous pouvons alors exécuter cette tâche en utilisant `> gradle hello` ou `> gradle -q hello` . Le `-q` est utilisé pour supprimer les messages de journal de dégradé afin que seule la sortie de la tâche soit affichée.

Sortie de `> gradle -q hello` :

```
> gradle -q hello
Hello world!
```

Plus sur les tâches

Tout d'abord: operator `<<` (leftShift) est équivalent à `doLast {closure}` . De **gradle 3.2,** il est **obsolète** . Tout le code de tâche écrit dans un **build.gradle** .

> Une tâche représente un travail atomique effectué par une construction. Cela peut être la compilation de certaines classes, la création d'un JAR, la génération de Javadoc ou la publication de certaines archives dans un référentiel.

Gradle prend en charge deux grands types de tâches: simples et améliorées.

Observons quelques styles de définition de tâche:

```
task hello {
    doLast{
       //some code
    }
}
```

Ou la:

```
task(hello) {
    doLast{
       //some code
    }
}
```

Ces tâches ci-dessus sont équivalentes. En outre, vous pouvez fournir des extensions à la tâche, tels que: `dependsOn` , `mustRunAfter` , le `type` etc. Vous pouvez étendre la tâche en ajoutant des actions après la définition des tâches, comme celle - ci:

```
task hello {
    doLast{
       println 'Inside task'
```

```
    }
}
hello.doLast {
    println 'added code'
}
```

Quand nous exécuterons ceci, nous avons:

```
> gradle -q hello
    Inside task
    added code
```

Questions sur les dépendances de tâches et la commande examinées ici

Parlons de deux grands types de tâches.

Simple:

Tâches que nous définissons avec une fermeture d'action:

```
    task hello {
        doLast{
        println "Hello from a simple task"
        }
    }
```

Renforcée

Amélioré c'est une tâche avec un comportement préconfiguré. Tous les plugins que vous utilisez dans votre projet sont les **tâches** *étendues* ou **améliorées** . Créons les nôtres et vous comprendrez comment cela fonctionne:

```
task hello(type: HelloTask)

class HelloTask extends DefaultTask {
    @TaskAction
    def greet() {
        println 'hello from our custom task'
    }
}
```

En outre, nous pouvons passer des paramètres à notre tâche, comme ceci:

```
class HelloTask extends DefaultTask {
    String greeting = "This is default greeting"
    @TaskAction
    def greet() {
```

```
        println greeting
    }
}
```

Et à partir de maintenant, nous pouvons réécrire notre tâche comme suit:

```
   //this is our old task definition style
task oldHello(type: HelloTask)
   //this is our new task definition style
task newHello(type: HelloTask) {
    greeting = 'This is not default greeting!'
}
```

Quand nous exécuterons ceci, nous avons:

```
> gradle -q oldHello
This is default greeting

> gradle -q newHello
This is not default greeting!
```

Toutes les questions sur le développement des plug-ins progressifs sur le site officiel

Lire Démarrer avec gradle en ligne: https://riptutorial.com/fr/gradle/topic/894/demarrer-avec-gradle

Chapitre 2: Dépendances de tâches

Remarques

doLast

Notez que dans un 3.x gradle plus la définition des tâches de façon idiomatiques: en utilisant **doLast explicite** notation à la place « leftShift » (<<) **{fermeture}** opérateur préférable **(leftShift** a été désapprouvée dans un gradle 3.2 est prévue pour être supprimée dans gradle 5.0. .)

```
task oldStyle << {
    println 'Deprecated style task'
 }
```

est équivalent à:

```
task newStyle {
    doLast {
    println 'Deprecated style task'
    }
 }
```

Examples

Ajout de dépendances à l'aide de noms de tâches

Nous pouvons changer l'ordre d'exécution des tâches avec la méthode `dependsOn` .

```
task A << {
    println 'Hello from A'
}
task B(dependsOn: A) << {
    println "Hello from B"
}
```

Ajouter `ensuiteOn: causes:

- la tâche B dépend de la tâche A
- Gradle pour exécuter `A` tâche à chaque fois **avant** l'exécution de la tâche `B`

Et la sortie est la suivante:

```
> gradle -q B
Hello from A
Hello from B
```

Ajout de dépendances d'un autre projet

```
project('projectA') {
    task A(dependsOn: ':projectB:B') << {
        println 'Hello from A'
    }
}

project('projectB') {
    task B << {
        println 'Hello from B'
    }
}
```

Pour faire référence à une tâche dans un autre projet, vous **préfixez le nom de la tâche** avec le chemin du projet `:projectB:B` elle appartient `:projectB:B`

Et la sortie est la suivante:

```
> gradle -q B
Hello from A
Hello from B
```

Ajout d'une dépendance à l'aide d'un objet tâche

```
task A << {
    println 'Hello from A'
}

task B << {
    println 'Hello from B'
}

B.dependsOn A
```

C'est un autre moyen de définir la dépendance au lieu d'utiliser le nom de la tâche .

Et la sortie est la même:

```
> gradle -q B
Hello from A
Hello from B
```

Ajout de plusieurs dépendances

Vous pouvez ajouter plusieurs dépendances.

```
task A << {
    println 'Hello from A'
}

task B << {
    println 'Hello from B'
}
```

```
task C << {
    println 'Hello from C'
}

task D << {
    println 'Hello from D'
}
```

Maintenant, vous pouvez définir un ensemble de dépendances:

```
B.dependsOn A
C.dependsOn B
D.dependsOn C
```

La sortie est la suivante:

```
> gradle -q D
Hello from A
Hello from B
Hello from C
Hello from D
```

Autre exemple:

```
B.dependsOn A
D.dependsOn B
D.dependsOn C
```

La sortie est la suivante:

```
> gradle -q D
Hello from A
Hello from B
Hello from C
Hello from D
```

Plusieurs dépendances avec la méthode princesse

Vous pouvez ajouter plusieurs dépendances.

```
task A << {
    println 'Hello from A'
}

task B(dependsOn: A) << {
    println 'Hello from B'
}

task C << {
    println 'Hello from C'
}

task D(dependsOn: ['B', 'C'] << {
    println 'Hello from D'
```

```
}
```

La sortie est la suivante:

```
> gradle -q D
Hello from A
Hello from B
Hello from C
Hello from D
```

Lire Dépendances de tâches en ligne: https://riptutorial.com/fr/gradle/topic/5545/dependances-de-taches

Chapitre 3: Gradle Init Scripts

Examples

Ajouter un référentiel par défaut pour tous les projets

Ajoutez un init.gradle à votre dossier de classement utilisateur. Le init.gradle est reconnu sur chaque projet.

```
Unix: ~/.gradle/init.gradle
```

> Ce sont également des emplacements alternatifs où le script init peut être placé et chargé automatiquement: -
>
> - N'importe **quel** fichier * **.gradle** dans **USER_HOME / .gradle / init.d**
> - Tout fichier * **.gradle** dans le répertoire **init.d de** l'installation de Gradle

init.gradle avec mavenLocal comme référentiel dans tous les projets.

```
allprojects {
    repositories {
        mavenLocal()
    }
}
```

Avec cela, vous avez votre cache local maven disponible dans tous les référentiels. Un cas d'utilisation pourrait être d'utiliser un fichier jar que vous avez ajouté avec "gradle install" dans un autre projet sans ajouter le référentiel mavenLocal à build.gradle ou en ajoutant un serveur Nexus / artifactory.

Lire Gradle Init Scripts en ligne: https://riptutorial.com/fr/gradle/topic/4234/gradle-init-scripts

Chapitre 4: Gradle Performance

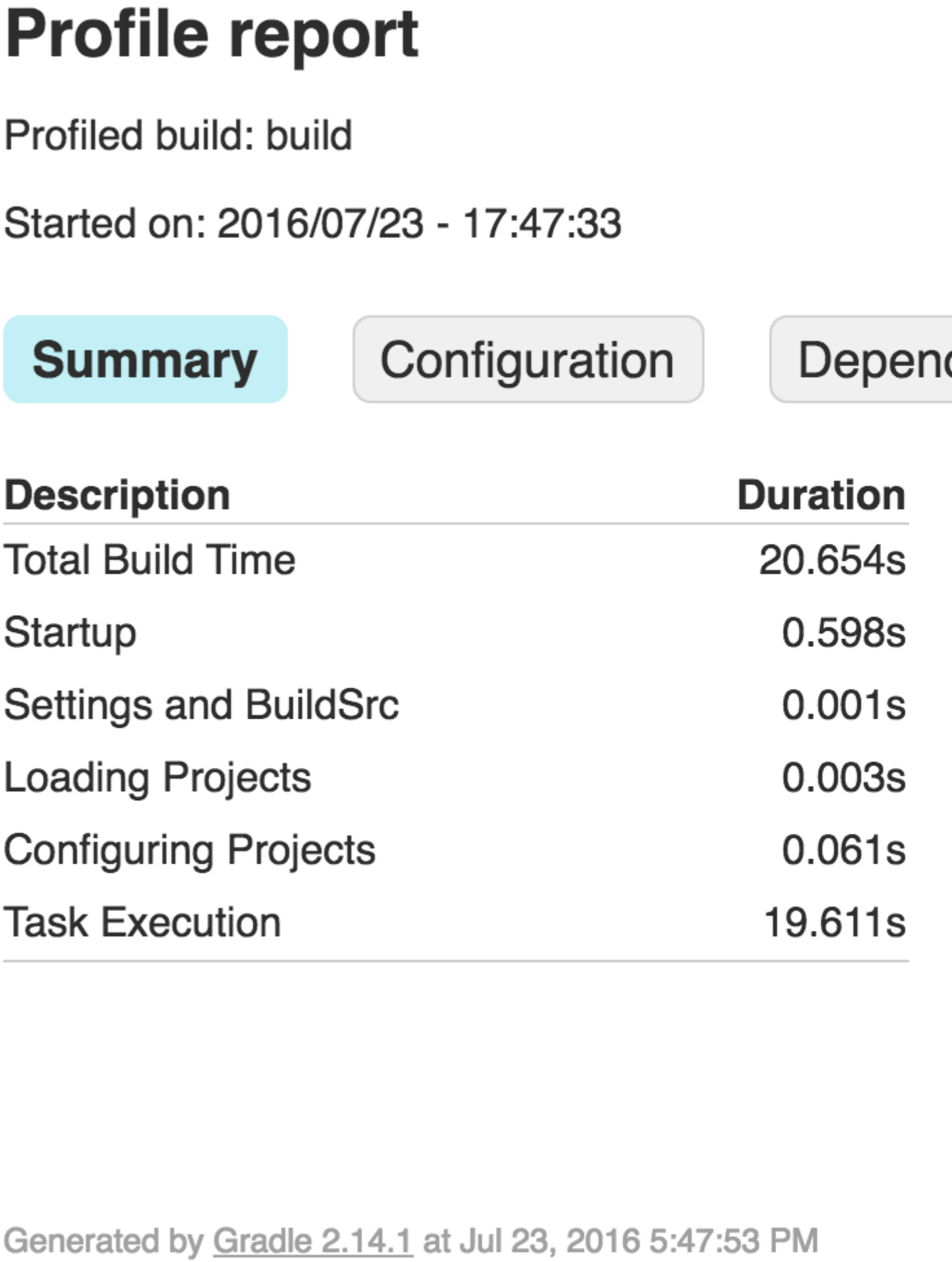
Examples

Profilage d'une construction

Avant de commencer à adapter votre version de Gradle aux performances, vous devez établir une référence et déterminer quelles parties de la génération prennent le plus de temps. Pour ce faire, vous pouvez profiler votre build en ajoutant l'argument `--profile` à votre commande Gradle:

```
gradle --profile
./gradlew --profile
```

Une fois la construction terminée, vous verrez un rapport de profil HTML pour la construction sous `./build/reports/profile/` , ressemblant à ceci:



En cliquant sur les onglets situés à côté de **Résumé**

, vous pouvez voir une répartition plus détaillée du temps passé.

Configurer à la demande

Si le profilage de votre génération montre une perte de temps importante dans la **configuration des projets** , l'option Configurer à la demande peut améliorer vos performances.

Vous pouvez activer le mode Configure on Demand en modifiant `$GRADLE_USER_HOME/.gradle/gradle.properties` (`~/.gradle/gradle.properties` par défaut) et en définissant `org.gradle.configureondemand` .

```
org.gradle.configureondemand=true
```

Pour l'activer uniquement pour un projet spécifique, modifiez `gradle.properties` fichier `gradle.properties` ce projet.

Si l'option Configurer à la demande est activée, au lieu de configurer tous les projets à l'avance, Gradle ne configurera que les projets nécessaires à l'exécution de la tâche.

Extrait du manuel de Gralde :

> Le mode Configuration à la demande tente de configurer uniquement les projets pertinents pour les tâches demandées, c'est-à-dire qu'il n'exécute que le fichier `build.gradle` des projets participant à la génération. De cette façon, le temps de configuration d'une grande construction multi-projets peut être réduit. À long terme, ce mode deviendra le mode par défaut, probablement le seul mode d’exécution de la version Gradle.

Réglage des paramètres d'utilisation de la mémoire JVM pour Gradle

Vous pouvez définir ou augmenter les limites d'utilisation de la mémoire (ou d'autres arguments JVM) utilisés pour les versions Gradle et le démon Gradle en modifiant `$GRADLE_USER_HOME/.gradle/gradle.properties` (`~/.gradle/gradle.properties` par défaut) et `org.gradle.jvmargs` .

Pour configurer ces limites uniquement pour un projet spécifique, modifiez `gradle.properties` fichier `gradle.properties` ce projet.

Les paramètres d'utilisation de la mémoire par défaut pour les versions de Gradle et le démon Gradle sont les suivants:

```
org.gradle.jvmargs=-Xmx1024m -XX:MaxPermSize=256m
```

Cela permet une allocation de mémoire maximale générale (taille de segment de mémoire) de 1 Go et une allocation de mémoire maximale pour les objets "internes" permanents de 256 Mo. Lorsque ces tailles sont atteintes, le nettoyage de la mémoire se produit, ce qui peut réduire considérablement les performances.

En supposant que vous avez la mémoire à perdre, vous pourriez facilement les doubler comme ceci:

```
org.gradle.jvmargs=-Xmx2024m -XX:MaxPermSize=512m
```

Notez que vous `XX:MaxPermSize` voir les avantages de l'augmentation de `XX:MaxPermSize` plus tôt que lorsque `Xmx` augmente pour ne plus être bénéfique.

Utilisez le démon Gradle

Vous pouvez activer le démon Gradle pour améliorer les performances de vos builds.

Le démon Gradle conserve l'initialisation et l'exécution de Framework Gradle et met en cache les données de projet en mémoire pour améliorer les performances.

Pour une seule construction

Pour activer le démon pour une seule génération, vous pouvez simplement transmettre l'argument `--daemon` à votre commande `gradle` ou à votre `gradle` Gradle Wrapper.

```
gradle --daemon
./gradlew --daemon
```

Pour toutes les versions d'un projet

Pour activer le démon pour toutes les versions d'un projet, vous pouvez ajouter:

```
org.gradle.daemon=true
```

Dans le fichier `gradle.properties` votre projet.

Pour toutes les constructions

Pour activer le démon Gradle par défaut, pour chaque génération effectuée par votre compte d'utilisateur sur votre système, éditez `$GRADLE_USER_HOME/.gradle/gradle.properties` (`~/.gradle/gradle.properties` par défaut) et ajoutez cette ligne:

```
org.gradle.daemon=true
```

Vous pouvez également le faire en une seule commande sur les systèmes Mac / Linux / * nix:

```
touch ~/.gradle/gradle.properties && echo "org.gradle.daemon=true" >>
~/.gradle/gradle.properties
```

Ou sous Windows:

```
(if not exist "%USERPROFILE%/.gradle" mkdir "%USERPROFILE%/.gradle") && (echo
org.gradle.daemon=true >> "%USERPROFILE%/.gradle/gradle.properties")
```

Désactiver le démon

Vous pouvez désactiver le démon pour une construction spécifique à l'aide de l'argument `--no-daemon` ou le désactiver pour un projet spécifique en définissant explicitement `org.gradle.daemon=false` dans le fichier `gradle.properties` du projet.

Arrêter le démon

Si vous souhaitez arrêter manuellement un processus Daemon, vous pouvez soit le supprimer via le gestionnaire de tâches de votre système d'exploitation, soit exécuter la commande `gradle --stop` . Le commutateur `--stop` oblige Gradle à demander que tous les processus Daemon en cours d'exécution, de la même version Gradle utilisée pour exécuter la commande, se terminent. D'ordinaire, les processus Daemon se terminent automatiquement * après * *3 heures d'inactivité ou moins* .

Gradle Parallel construit

Gradle exécutera une seule tâche à la fois par défaut, quelle que soit la structure du projet. En utilisant le commutateur `--parallel` , vous pouvez forcer Gradle à exécuter des sous-projets indépendants - ceux qui n'ont pas de dépendances de projet implicites ou explicites entre eux - en parallèle, ce qui lui permet d'exécuter plusieurs tâches en même temps. différents projets.

Pour construire un projet en mode parallèle:

```
gradle build --parallel
```

Vous pouvez également rendre la construction par défaut parallèle à un projet en ajoutant le paramètre suivant au fichier gradle.properties du projet:

```
org.gradle.parallel=true
```

Utiliser la dernière version de Gradle

L'équipe de Gradle travaille régulièrement sur l'amélioration des performances des différents aspects des versions de Gradle. Si vous utilisez une ancienne version de Gradle, vous ne profitez pas des avantages de ce travail. Essayez de passer à la dernière version de Gradle pour voir quel impact cela a. Cela est peu risqué car très peu de choses se cassent entre les versions mineures de Gradle.

Le fichier de propriétés de l'encapsuleur Gradle se trouve dans le dossier de votre projet sous `gradle/wrapper/` et s'appelle `gradle-wrapper.properties` . Le contenu de ce fichier pourrait ressembler à ceci:

```
distributionBase=GRADLE_USER_HOME
distributionPath=wrapper/dists
zipStoreBase=GRADLE_USER_HOME
zipStorePath=wrapper/dists
distributionUrl=https\://services.gradle.org/distributions/gradle-X.X.X.zip
```

Vous pouvez modifier manuellement le numéro de version `XXX` (version actuelle) à `YYY` (version plus récente) et la prochaine fois que vous exécuterez le wrapper, la nouvelle version sera téléchargée automatiquement.

Lire Gradle Performance en ligne: https://riptutorial.com/fr/gradle/topic/3443/gradle-performance

Chapitre 5: Gradle Plugins

Examples

Simple plugin de gradule de `buildSrc`

Exemple simple de création d'un plugin personnalisé et de DSL pour votre projet de graduation.
Cet exemple utilise l'une des trois manières possibles de créer des plug-ins.
Les trois manières sont:

- en ligne
- buildSrc
- plugins autonomes

Cet exemple montre **comment** créer un plug-in à partir du dossier **buildSrc** .

Cet exemple va créer cinq fichiers

```
// project's build.gradle
build.gradle
// build.gradle to build the `buildSrc` module
buildSrc/build.gradle
// file name will be the plugin name used in the `apply plugin: $name`
// where name would be `sample` in this example
buildSrc/src/main/resources/META-INF/gradle-plugins/sample.properties
// our DSL (Domain Specific Language) model
buildSrc/src/main/groovy/so/docs/gradle/plugin/SampleModel.groovy
// our actual plugin that will read the values from the DSL
buildSrc/src/main/groovy/so/docs/gradle/plugin/SamplePlugin.groovy
```

build.gradle:

```
group 'so.docs.gradle'
version '1.0-SNAPSHOT'

apply plugin: 'groovy'
// apply our plugin... calls SamplePlugin#apply(Project)
apply plugin: 'sample'

repositories {
    mavenCentral()
}

dependencies {
    compile localGroovy()
}

// caller populates the extension model applied above
sample {
    product = 'abc'
    customer = 'zyx'
}
```

```
// dummy task to limit console output for example
task doNothing <<{}
```

buildSrc / build.gradle

```
apply plugin: 'groovy'

repositories {
    mavenCentral()
}

dependencies {
    compile localGroovy()
}
```

buildSrc / src / main / groovy / so / docs / gradle / plugin / SamplePlugin.groovy:

```
package so.docs.gradle.plugin

import org.gradle.api.Plugin
import org.gradle.api.Project

class SamplePlugin implements Plugin<Project> {
    @Override
    void apply(Project target) {
        // create our extension on the project for our model
        target.extensions.create('sample', SampleModel)
        // once the script has been evaluated the values are available
        target.afterEvaluate {
            // here we can do whatever we need to with our values
            println "populated model: $target.extensions.sample"
        }
    }
}
```

buildSrc / src / main / groovy / so / docs / gradle / plugin / SampleModel.groovy:

```
package so.docs.gradle.plugin

// define our DSL model
class SampleModel {
    public String product;
    public String customer;

    @Override
    public String toString() {
        final StringBuilder sb = new StringBuilder("SampleModel{");
        sb.append("product='").append(product).append('\'');
        sb.append(", customer='").append(customer).append('\'');
        sb.append('}');
        return sb.toString();
    }
}
```

buildSrc / src / main / resources / META-INF / gradle-plugins / sample.properties

```
implementation-class=so.docs.gradle.plugin.SamplePlugin
```

En utilisant cette configuration, nous pouvons voir les valeurs fournies par l'appelant dans votre bloc DSL

```
 $ ./gradlew -q doNothing
SampleModel{product='abc', customer='zyx'}
```

Comment écrire un plugin autonome

Pour créer un plug-in Gradle autonome personnalisé avec java (vous pouvez également utiliser Groovy), vous devez créer une structure comme celle-ci:

```
plugin
|-- build.gradle
|-- settings.gradle
|-- src
    |-- main
    |   |-- java
    |   |-- resources
    |       |-- META-INF
    |           |-- gradle-plugins
    |-- test
```

Configuration de la configuration progressive

Dans le fichier `build.gradle` , vous définissez votre projet.

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

dependencies {
    compile gradleApi()
}
```

Le plugin `java` sera utilisé pour écrire du code Java.
La dépendance `gradleApi()` nous donnera toutes les méthodes et propriétés nécessaires pour créer un plugin Gradle.

Dans le fichier `settings.gradle` :

```
rootProject.name = 'myplugin'
```

Il définira l' **identifiant de** l' **artefact** dans Maven.
Si le fichier `settings.gradle` n'est pas présent dans le répertoire du plugin, la valeur par défaut sera le nom du répertoire.

Créer le plugin

Définissez une classe dans `src/main/java/org/sample/MyPlugin.java` implémentant l'interface du `Plugin` .

```
import org.gradle.api.Plugin;
import org.gradle.api.Project;

public class MyPlugin implements Plugin<Project> {

    @Override
    public void apply(Project project) {
         project.getTasks().create("myTask", MyTask.class);
    }

}
```

Définissez la tâche étendant la classe `DefaultTask` :

```
import org.gradle.api.DefaultTask;
import org.gradle.api.tasks.TaskAction;

public class MyTask extends DefaultTask {

    @TaskAction
    public void myTask() {
        System.out.println("Hello World");
    }
}
```

Déclaration de classe de plugin

Dans le dossier `META-INF/gradle-plugins` , vous devez créer un fichier de propriétés définissant la propriété de `implementation-class` qui identifie la classe d'implémentation du plug `META-INF/gradle-plugins` .

Dans le `META-INF/gradle-plugins/testplugin.properties`

```
implementation-class=org.sample.MyPlugin.java
```

Notez que le **nom du fichier de propriétés correspond à l'ID du plug-in** .

Comment le construire et le publier

Modifiez le fichier `build.gradle` en ajoutant des informations pour télécharger le plug-in dans un dépôt maven:

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

```
apply plugin: 'maven'

dependencies {
    compile gradleApi()
}

repositories {
    jcenter()
}


group = 'org.sample'
version = '1.0'

uploadArchives {
    repositories {
        mavenDeployer {
        repository(url: mavenLocal().url)
        }
    }
}
```

Vous pouvez créer et publier le plug-in Gradle sur le `plugin/build.gradle` Maven défini dans le fichier `plugin/build.gradle` à l'aide de la commande suivante.

```
$ ./gradlew clean uploadArchives
```

Comment l'utiliser

Pour utiliser le plug-in ajouter dans le `build.gradle` de votre projet:

```
buildscript {
     repositories {
         mavenLocal()
     }
 dependencies {
    classpath group: 'org.sample',    // Defined in the build.gradle of the plugin
              name: 'myplugin',       // Defined by the rootProject.name
              version: '1.0'
    }
 }

apply plugin: 'testplugin'            // Defined by the properties filename
```

Ensuite, vous pouvez appeler la tâche en utilisant:

```
 $ ./gradlew myTask
```

Lire Gradle Plugins en ligne: https://riptutorial.com/fr/gradle/topic/1900/gradle-plugins

Chapitre 6: Gradle Wrapper

Examples

Gradle Wrapper et Git

Comme nous l'avons vu dans l'introduction, la fonctionnalité d'encapsulation progressive fonctionne car un fichier JAR est téléchargé dans le projet pour être utilisé lors de l' `gradlew` commande `gradlew` . Toutefois, cela risque de ne pas être `gradlew` et après la prochaine `gradlew` du projet, `gradlew` ne pourra pas s'exécuter avec l'erreur:

```
Error: Could not find or load main class org.gradle.wrapper.GradleWrapperMain
```

Ce sera parce que votre fichier .gitignore inclura probablement `*jar` pour les projets Java. Lors de l'initialisation de l'encapsuleur, il est `gradle/wrapper/gradle-wrapper.jar` dans le fichier `gradle/wrapper/gradle-wrapper.jar` . Vous devez donc l'ajouter à l'index git et le valider. Faites-le avec:

```
git add -f gradle/wrapper/gradle-wrapper.jar
git ci
```

Avec le `-f` pour le forcer.

Gradle Wrapper introduction

Gradle a la capacité d'ajouter un wrapper aux projets. Ce wrapper réduit le besoin pour tous les utilisateurs ou les systèmes d'intégration continue d'installer Gradle. Il empêche également les problèmes de version où il existe une incompatibilité entre la version utilisée par le projet et celle installée par les utilisateurs. Cela se fait en installant une version de gradle localement dans le projet.

Les utilisateurs du projet exécutent simplement:

```
> ./gradlew <task> # on *Nix or MacOSX
> gradlew <task>   # on Windows
```

Pour configurer un projet afin d'utiliser un wrapper, les développeurs:

1. Exécuter:

```
gradle wrapper [--gradle-version 2.0]
```

Où `--gradle-version X` est facultatif et s'il n'est pas fourni (ou la tâche wrapper n'est pas incluse, comme indiqué ci-dessous), la version utilisée est la version de gradle utilisée.

1. Pour forcer le projet à utiliser une version spécifique, ajoutez ce qui suit à `build.gradle` :

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

Lorsque la commande `gradle wrapper` est exécutée, elle crée les fichiers:

```
the_project/
  gradlew
  gradlew.bat
  gradle/wrapper/
    gradle-wrapper.jar
    gradle-wrapper.properties
```

La documentation officielle sur cette fonctionnalité est à https://docs.gradle.org/current/userguide/gradle_wrapper.html .

Utilisez Gradle dans la Gradle Wrapper

Si vous souhaitez conserver la copie locale de Gradle et laisser le wrapper l'utiliser dans les builds, vous pouvez définir la `distributionUrl` pointant vers votre copie sur la tâche `wrapper` :

```
task wrapper(type: Wrapper) {
    gradleVersion = '2.0'
    distributionUrl = "http\://server/dadada/gradle-${gradleVersion}-bin.zip"
}
```

Après l'exécution de `gradle wrapper` , le script shell `gradlew` est créé et le `gradle/wrapper/gradle-wrapper.properties` est configuré pour utiliser l'URL fournie pour télécharger le fichier Gradle.

Utiliser le Gradle Wrapper derrière un proxy

La première fois qu'un utilisateur exécute d'un projet `gradlew` , il faut se rendre compte qu'il fera deux éléments clés:

1. Vérifiez si la version du dégradé utilisée par le wrapper est déjà dans ~ / .gradle / wrapper / dists
2. Sinon, téléchargez les archives de la version depuis Internet

Si vous vous trouvez dans un environnement qui nécessite que tout le trafic externe passe par un proxy, la deuxième étape échoue (à moins que ce ne soit un environnement proxy transparent). Par conséquent, vous devez vous assurer que les paramètres du proxy *JVM sont* définis.

Par exemple, si vous avez une configuration proxy de base sans authentification, définissez simplement la variable d'environnement `JAVA_OPTS` ou `GRADLE_OPTS` avec:

```
-Dhttps.proxyPort=<proxy_port> -Dhttps.proxyHost=<hostname>
```

Ainsi, un exemple complet sur Windows serait:

```
set JAVA_OPTS=-Dhttps.proxyPort=8080 -Dhttps.proxyHost=myproxy.mycompany.com
```

Si toutefois votre environnement nécessite également une authentification, vous souhaiterez également examiner vos autres options à l' adresse https://docs.oracle.com/javase/8/docs/api/java/net/doc-files/net-properties. html .

*REMARQUE: Cette configuration de proxy **s'ajoute** à toute configuration de proxy pour l'accès à votre référentiel de dépendances.*

Lire Gradle Wrapper en ligne: https://riptutorial.com/fr/gradle/topic/3006/gradle-wrapper

Chapitre 7: Incrément automatique du numéro de version à l'aide du script Gradle pour les applications Android

Examples

Comment appeler la méthode d'incrémentation automatique lors de la génération

```
gradle.taskGraph.whenReady {taskGraph ->
    if (taskGraph.hasTask(assembleDebug)) {  /* when run debug task */
        autoIncrementBuildNumber()
    } else if (taskGraph.hasTask(assembleRelease)) { /* when run release task */
        autoIncrementBuildNumber()
    }
}
```

Méthode d'incrémentation automatique

```
 /*Wrapping inside a method avoids auto incrementing on every gradle task run. Now it runs
only when we build apk*/
ext.autoIncrementBuildNumber = {

    if (versionPropsFile.canRead()) {
        def Properties versionProps = new Properties()
        versionProps.load(new FileInputStream(versionPropsFile))
        versionBuild = versionProps['VERSION_BUILD'].toInteger() + 1
        versionProps['VERSION_BUILD'] = versionBuild.toString()
        versionProps.store(versionPropsFile.newWriter(), null)
    } else {
        throw new GradleException("Could not read version.properties!")
    }
}
```

Lire et affecter un numéro de version d'un fichier de propriétés à une variable

def versionPropsFile = fichier ('version.properties') def versionBuild

```
/*Setting default value for versionBuild which is the last incremented value stored in the
file */
if (versionPropsFile.canRead()) {
    def Properties versionProps = new Properties()
    versionProps.load(new FileInputStream(versionPropsFile))
    versionBuild = versionProps['VERSION_BUILD'].toInteger()
} else {
    throw new GradleException("Could not read version.properties!")
}
```

Lire Incrément automatique du numéro de version à l'aide du script Gradle pour les applications Android en ligne: https://riptutorial.com/fr/gradle/topic/10696/increment-automatique-du-numero-de-version-a-l-aide-du-script-gradle-pour-les-applications-android

Chapitre 8: Initialiser Gradle

Remarques

Terminologie

- Tâche - une pièce atomique exécutée par une construction. Les tâches ont des `inputs` , des `outputs` et des dépendances de tâches.
- `dependencies {}` - Indique les dépendances de `File` ou binaires nécessaires à l'exécution des tâches. Par exemple, `org.slf4j:slf4j-api:1.7.21` correspond aux coordonnées abrégées d'une dépendance de Maven.
- `repositories {}` - Comment Gradle trouve les fichiers pour les dépendances externes. Vraiment, juste une collection de fichiers organisés par groupe, nom et version. Par exemple: `jcenter()` est une méthode pratique pour `maven { url 'http://jcenter.bintray.com/' }` , un référentiel Bintray Maven .

Examples

Initialisation d'une nouvelle bibliothèque Java

Condition préalable: Installation de Gradle

Une fois que Gradle est installé, vous pouvez configurer un projet nouveau ou existant en exécutant

```
cd $PROJECT_DIR
gradle init --type=java-library
```

Notez qu'il existe d' autres types de projets comme Scala avec lesquels vous pouvez démarrer, mais nous utiliserons Java pour cet exemple.

Vous allez vous retrouver avec:

```
.
├── build.gradle
├── gradle
│   └── wrapper
│       ├── gradle-wrapper.jar
│       └── gradle-wrapper.properties
├── gradlew
├── gradlew.bat
├── settings.gradle
└── src
    ├── main
    │   └── java
    │       └── Library.java
    └── test
        └── java
```

```
            └── LibraryTest.java
```

Vous pouvez maintenant exécuter des `gradle tasks` et voir que vous pouvez créer un `jar` , exécuter des `test` , produire des `javadoc` et bien plus encore, même si votre fichier `build.gradle` est:

```
apply plugin: 'java'

repositories {
    jcenter()
}

dependencies {
    compile 'org.slf4j:slf4j-api:1.7.21'
    testCompile 'junit:junit:4.12'
}
```

Lire Initialiser Gradle en ligne: https://riptutorial.com/fr/gradle/topic/2247/initialiser-gradle

Chapitre 9: IntelliJ IDEA Personnalisation des tâches

Syntaxe

- groovy.util.Node = node.find {childNode -> return true || faux }
- node.append (nodeYouWantAsAChild)
- groovy.util.Node parsedNode = (nouveau XmlParser ()). parseText (someRawXMLString)
- " 'chaîne de caractères multi-lignes (non interpolée)' "

Remarques

Les trois fichiers de base d'un projet IntelliJ - les fichiers ipr, iws et iml - sont accessibles comme dans la tâche d'idée via

```
project.ipr
module.iml
workspace.iws
```

L'utilisation du fichier .withXml vous permet d'accéder au fichier XML. En utilisant le fichier .asNode (), il devient un nœud XML groovy.

Ex:

```
project.ipr.withXml { provider ->
    def node = provider.asNode()
```

À partir de là, il est assez simple de modifier graduellement pour configurer les projets IntelliJ, de prendre le fichier au démarrage, d'effectuer les actions que vous souhaitez exécuter (dans IntelliJ), puis de modifier le nouveau fichier avec l'ancien fichier. Vous devriez voir quel XML vous aurez besoin pour personnaliser le travail d'idée. Vous devrez également prendre note de l'emplacement du fichier XML.

Une autre chose à prendre en compte est que vous ne voulez pas de doublons dans les fichiers IntelliJ si vous exécutez l'idée de dégradé plusieurs fois. Donc, vous voudrez rechercher le nœud que vous souhaitez créer et si ce n'est pas le cas, vous pouvez le créer et l'insérer.

Pièges:

Parfois, lorsqu'on utilise == pour la comparaison de chaînes dans la méthode find, cela échoue. En testant et je trouve que c'est le cas, j'utilise .contains.

Lors de la recherche de nœuds, tous les nœuds ne possèdent pas l'attribut que vous utilisez en tant que critère. Assurez-vous donc de vérifier la valeur null.

Examples

Ajouter une configuration d'exécution de base

Hypothèses pour cet exemple:

- Vous avez une classe, `foo.bar.Baz` .
- Vous souhaitez créer une configuration d'exécution qui exécute la méthode principale.
- C'est dans un module appelé `fooBar` .

Dans votre dossier de classement:

```
idea {
    workspace.iws.withXml { provider ->
        // I'm not actually sure why this is necessary
        def node = provider.asNode()

        def runManager = node.find { it.@name.contains('RunManager')}

        // find a run configuration if it' there already
        def runner = runManager.find { it.find ({ mainClass ->
            return mainClass.@name != null && mainClass.@name == "MAIN_CLASS_NAME" &&
mainClass.@value != null && mainClass.@value.contains('Baz');
        }) != null }

        // create and append the run configuration if it doesn't already exists
        if (runManager != null && runner == null){
            def runnerText = '''
                <configuration default="false" name="Baz" type="Application"
factoryName="Application" nameIsGenerated="true">
                  <extension name="coverage" enabled="false" merge="false" runner="idea">
                    <pattern>
                      <option name="PATTERN" value="foo.bar.Baz" />
                      <option name="ENABLED" value="true" />
                    </pattern>
                  </extension>
                  <option name="MAIN_CLASS_NAME" value="foo.bar.Baz" />
                  <option name="VM_PARAMETERS" value="" />
                  <option name="PROGRAM_PARAMETERS" value="" />
                  <option name="WORKING_DIRECTORY" value="file://$PROJECT_DIR$" />
                  <option name="ALTERNATIVE_JRE_PATH_ENABLED" value="false" />
                  <option name="ALTERNATIVE_JRE_PATH" />
                  <option name="ENABLE_SWING_INSPECTOR" value="false" />
                  <option name="ENV_VARIABLES" />
                  <option name="PASS_PARENT_ENVS" value="true" />
                  <module name="foobar" />
                  <envs />
                  <method />
                </configuration>'''
            runner = (new XmlParser()).parseText(runnerText)
            runManager.append(config);
        }

        // If there is no active run configuration, set the newly made one to be it
        if (runManager != null && runManager.@selected == null) {
            runManager.@selected="${runner.@factoryName}.${runner.@name}"
        }
```

```
    }
}
```

Lire IntelliJ IDEA Personnalisation des tâches en ligne:
https://riptutorial.com/fr/gradle/topic/2297/intellij-idea-personnalisation-des-taches

Chapitre 10: Les dépendances

Examples

Ajouter une dépendance de fichier JAR local

JAR simple

Vous avez parfois un fichier JAR local que vous devez ajouter comme dépendance à votre version de Gradle. Voici comment procéder:

```
dependencies {
    compile files('path/local_dependency.jar')
}
```

Où `path` est un chemin de répertoire sur votre système de fichiers et `local_dependency.jar` est le nom de votre fichier JAR local. Le `path` peut être relatif au fichier de construction.

Répertoire des JAR

Il est également possible d'ajouter un répertoire de jars à compiler. Cela peut être fait comme ça:

```
dependencies {
        compile fileTree(dir: 'libs', include: '*.jar')
}
```

Où `libs` serait le répertoire contenant les jars et `*.jar` serait le filtre des fichiers à inclure.

Répertoire des JAR en tant que référentiel

Si vous souhaitez uniquement rechercher des fichiers JAR dans un référentiel au lieu de les ajouter directement en tant que dépendance avec leur chemin, vous pouvez utiliser un référentiel flatDir.

```
repositories {
    flatDir {
        dirs 'libs'
    }
}
```

Recherche des fichiers JAR dans le répertoire *`libs`* et ses répertoires enfants.

Ajouter une dépendance

Les dépendances à Gradle suivent le même format que Maven . Les dépendances sont structurées comme suit:

```
group:name:version
```

Voici un exemple:

```
'org.springframework:spring-core:4.3.1.RELEASE'
```

Pour ajouter une dépendance à la compilation, ajoutez simplement cette ligne dans votre bloc de `dependency` dans le fichier de compilation Gradle:

```
compile 'org.springframework:spring-core:4.3.1.RELEASE'
```

Une syntaxe alternative pour ceci nomme explicitement chaque composant de la dépendance, comme ceci:

```
compile group: 'org.springframework', name: 'spring-core', version: '4.3.1.RELEASE'
```

Cela ajoute une dépendance à la compilation.

Vous pouvez également ajouter des dépendances uniquement pour les tests. Voici un exemple:

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

Dépend d'un autre projet Gradle

Dans le cas d'une construction en plusieurs projets, vous devrez peut-être parfois dépendre d'un autre projet dans votre version. Pour ce faire, vous devez entrer les éléments suivants dans les dépendances de votre projet:

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

Où `':OtherProject'` est le chemin de dégradé du projet, référencé à partir de la racine de la structure de répertoires.

Pour que `':OtherProject'` disponible dans le contexte du fichier `build.gradle` , ajoutez-le aux `settings.gradle` correspondants

```
include ':Dependency'
project(':Dependency').projectDir = new File('/path/to/dependency')
```

Pour une explication plus détaillée, vous pouvez consulter la documentation officielle de Gradle ici .

Dépendances de liste

L'appel de la tâche de `dependencies` vous permet de voir les dépendances du projet racine:

```
gradle dependencies
```

Les résultats sont des graphes de dépendance (prenant en compte les dépendances transitives), ventilés par configuration. Pour restreindre les configurations affichées, vous pouvez passer l'option `--configuration` suivie d'une configuration choisie pour analyser:

```
gradle dependencies --configuration compile
```

Pour afficher les dépendances d'un sous-projet, utilisez la tâche `<subproject>:dependencies` . Par exemple pour lister les dépendances d'un sous-projet nommé `api` :

```
gradle api:dependencies
```

Ajouter des référentiels

Vous devez indiquer à Gradle l'emplacement de vos plug-ins pour que Gradle puisse les trouver. Pour ce faire, ajoutez un `repositories { ... }` à votre `build.gradle` .

Voici un exemple d'ajout de trois référentiels, JCenter , Maven Repository et un référentiel personnalisé offrant des dépendances dans le style Maven.

```
repositories {
  // Adding these two repositories via method calls is made possible by Gradle's Java plugin
  jcenter()
  mavenCentral()

  maven { url "http://repository.of/dependency" }
}
```

Ajouter le fichier .aar au projet Android en utilisant gradle

1. Accédez au module d' `app` du projet et créez le répertoire `libs` .
2. Placez votre fichier `.aar` ici. Par exemple `myLib.aar` .
3. Ajoutez le code ci-dessous au bloc `android` du fichier `build.gradle` de niveau `app` .

```
  repositories {
        flatDir {
            dirs 'libs'
        }
    }
```

De cette façon, vous avez défini un nouveau référentiel supplémentaire qui pointe vers le dossier `libs` du module d' `app` .

4. Ajoutez le code ci-dessous au bloc de `dependencies` ou au fichier `build.gradle` :

```
compile(name:'myLib', ext:'aar')
```

Lire Les dépendances en ligne: https://riptutorial.com/fr/gradle/topic/2524/les-dependances

Chapitre 11: Tâches de commande

Remarques

> Veuillez noter que `mustRunAfter` et `shouldRunAfter` sont marqués comme "incubant" (à partir de Gradle 3.0), ce qui signifie que ce sont des fonctionnalités expérimentales et que leur comportement peut être modifié dans les versions ultérieures.

Deux règles de commande sont disponibles:

- `mustRunAfter`
- `shouldRunAfter`

Lorsque vous utilisez la règle de commande `mustRunAfter` , vous spécifiez que taskB doit toujours s'exécuter après taskA, à chaque fois que taskA et taskB seront exécutés.

La règle de commande `shouldRunAfter` est similaire mais moins stricte car elle sera ignorée dans deux situations:

- si l'utilisation de cette règle introduit un cycle de commande.
- Lorsque vous utilisez l'exécution parallèle et que toutes les dépendances d'une tâche ont été satisfaites, à l'exception de la tâche `shouldRunAfter` , cette tâche sera exécutée, que ses dépendances `shouldRunAfter` aient été exécutées ou non.

Examples

Commande avec la méthode mustRunAfter

```
task A << {
    println 'Hello from A'
}
task B << {
    println 'Hello from B'
}

B.mustRunAfter A
```

La ligne `B.mustRunAfter A` indique à Gradle d'exécuter la tâche après la tâche spécifiée en tant qu'argument.

Et la sortie est la suivante:

```
> gradle -q B A
Hello from A
Hello from B
```

La règle de classement n'introduit pas de dépendance entre les tâches A et B, mais n'a d'effet que lorsque les **deux tâches sont planifiées** pour exécution.

Cela signifie que nous pouvons exécuter les tâches A et B indépendamment.

La sortie est la suivante:

```
> gradle -q B
Hello from B
```

Lire Tâches de commande en ligne: https://riptutorial.com/fr/gradle/topic/5550/taches-de-commande

Chapitre 12: Utiliser des plugins tiers

Examples

Ajouter un plug-in tiers à build.gradle

Gradle (toutes les versions) *Cette méthode fonctionne pour toutes les versions de gradle*

Ajoutez le code de compilation au début de votre fichier build.gradle.

```
buildscript {
  repositories {
    maven {
      url "https://plugins.gradle.org/m2/"
    }
  }
  dependencies {
    classpath "org.example.plugin:plugin:1.1.0"
  }
}

apply plugin: "org.example.plugin"
```

Gradle (Versions 2.1+) *Cette méthode ne fonctionne que pour les projets utilisant Gradle 2.1 ou une version ultérieure.*

```
plugins {
  id "org.example.plugin" version "1.1.0"
}
```

build.gradle avec plusieurs plug-ins tiers

Gradle (toutes les versions)

Lorsque vous ajoutez plusieurs plug-ins tiers, vous n'avez pas besoin de les séparer en différentes instances du code buildscript (All) ou du plug-in (2.1+), de nouveaux plug-ins peuvent être ajoutés aux côtés de plug-ins préexistants.

```
buildscript {
  repositories {
    maven {
      url "https://plugins.gradle.org/m2/"
    }
  }
  dependencies {
    classpath "org.example.plugin:plugin:1.1.0"
    Classpath "com.example.plugin2:plugin2:1.5.2"
  }
}

apply plugin: "org.example.plugin"
```

```
apply plugin: "com.example.plugin2"
```

Gradle (Versions 2.1+)

```
plugins {
  id "org.example.plugin" version "1.1.0"
  id "com.example.plugin2" version "1.5.2"
}
```

Lire Utiliser des plugins tiers en ligne: https://riptutorial.com/fr/gradle/topic/9183/utiliser-des-plugins-tiers

Chapitre 13: Y compris la source native - Expérimental

Paramètres

Paramètres	Détails
model.android.ndk.toolchain	Chaîne d'outils native trouvée dans le dossier ndk-bundle

Examples

Basic JNI Gradle Config

root: build.gradle

```
buildscript {
    repositories {
        jcenter()
    }
    dependencies {
        classpath 'com.android.tools.build:gradle-experimental:0.8.0-alpha4'
    }
}

allprojects {
    repositories {
        jcenter()
    }
}
```

app: build.gradle

```
apply plugin: 'com.android.model.application'

dependencies {
    compile "com.android.support:support-v4:23.3.0"
    compile fileTree(dir: 'libs', include: '*.jar')
}

model {
    android {
        compileSdkVersion = 23
        buildToolsVersion = '23.0.3'

        defaultConfig {
            applicationId = 'com.example.hello'
            minSdkVersion.apiLevel = 9
            targetSdkVersion.apiLevel = 23

            buildConfigFields {
```

```
                create() {
                    type "int"
                    name "VALUE"
                    value "1"
                }
            }
        }

        ndk {
            platformVersion = 9
            moduleName "hello"

            toolchain "clang"

            stl "gnustl_static"
            CFlags.add("-DANDROID_NDK")
            cppFlags.add("-std=c++11")

            ldLibs.add("android")
            ldLibs.add("dl")
            ldLibs.add("log")
        }

        sources {
            main {
                jni {
                    exportedHeaders {
                        srcDirs "../../common/headers"
                    }
                    source {
                        srcDirs "../../common/src"
                    }
                }
            }
        }
    }
}
```

Utilisation de bibliothèques prédéfinies et OpenGL ES 2.0

root: build.gradle

```
buildscript {
    repositories {
        jcenter()
    }
    dependencies {
        classpath 'com.android.tools.build:gradle-experimental:0.8.0-alpha4'
    }
}

allprojects {
    repositories {
        jcenter()
    }
}
```

app: build.gradle

```
apply plugin: 'com.android.model.application'

dependencies {
    compile "com.android.support:support-v4:23.3.0"
    compile fileTree(dir: 'libs', include: '*.jar')
}

model {
    android {
        compileSdkVersion = 23
        buildToolsVersion = '23.0.3'

        defaultConfig {
            applicationId = 'com.example.glworld'
            minSdkVersion.apiLevel = 9
            targetSdkVersion.apiLevel = 23

            buildConfigFields {
                create() {
                    type "int"
                    name "VALUE"
                    value "1"
                }
            }
        }

        buildTypes {
            release {
                minifyEnabled = false
                proguardFiles.add(file('proguard-rules.txt'))
            }
        }

        ndk {
            platformVersion = 9
            moduleName "glworld"

            toolchain "clang"

            stl "gnustl_static"
            CFlags.add("-DANDROID_NDK")
            CFlags.add("-DDISABLE_IMPORTGL")
            CFlags.add("-DFT2_BUILD_LIBRARY=1")
            cppFlags.add("-std=c++11")

            ldLibs.add("EGL")
            ldLibs.add("android")
            ldLibs.add("GLESv2")
            ldLibs.add("dl")
            ldLibs.add("log")
        }

        sources {
            main {
                jni {
                    dependencies {
                        library "freetype2" linkage "shared"
                    }
                    exportedHeaders {
                        srcDirs "../../common/headers"
                    }
```

```
                    source {
                        srcDirs "../../common/src"
                    }
                }
            }
        }
    }

    repositories {
        prebuilt(PrebuiltLibraries) {
            freetype2 {
                headers.srcDir "../../common/freetype2-android/include"
                binaries.withType(SharedLibraryBinary) {
                    def localLib = "../../common/freetype2-android/Android/libs"
                    sharedLibraryFile =
                            file("$localLib/${targetPlatform.getName()}/libfreetype2.so")
                }
            }
        }
    }
}

// The next tasks compile a freetype library using a make file.
// These `.so`'s are then used as the shared libraries compiled above.
tasks.withType(JavaCompile) {
    compileTask -> compileTask.dependsOn buildNative
}

// Call regular ndk-build (.cmd) script from the app directory
task buildNative(type: Exec) {
    def ndkDir = "/Development/android-sdk-macosx/ndk-bundle"
    commandLine "$ndkDir/ndk-build",
            '-C',
            file('../../common/freetype2-android/Android/jni').absolutePath
}

task cleanNative(type: Exec) {
    def ndkDir = "/Development/android-sdk-macosx/ndk-bundle"
    commandLine "$ndkDir/ndk-build",
            '-C',
            file('../../common/freetype2-android/Android/jni').absolutePath,
            "clean"
}

clean.dependsOn cleanNative
```

Lire Y compris la source native - Expérimental en ligne:
https://riptutorial.com/fr/gradle/topic/4460/y-compris-la-source-native---experimental

Crédits

S. No	Chapitres	Contributeurs
1	Démarrer avec gradle	Afterfield, bassim, Community, Emil Burzo, Eric Wendelin, Hamzawey, Hillkorn, Matthias Braun, Nikem, Pepper Lebeck-Jobe, Sergey Yakovlev, Stanislav, user2555595, vanogrid, Will
2	Dépendances de tâches	Gabriele Mariotti, Sergey Yakovlev, Stanislav
3	Gradle Init Scripts	ambes, Hillkorn
4	Gradle Performance	ambes, Sergey Yakovlev, Will
5	Gradle Plugins	Gabriele Mariotti, JBirdVegas
6	Gradle Wrapper	ajoberstar, Fanick, HankCa, I Stevenson
7	Incrément automatique du numéro de version à l'aide du script Gradle pour les applications Android	Jayakrishnan PM
8	Initialiser Gradle	Eric Wendelin, Will
9	IntelliJ IDEA Personnalisation des tâches	IronHorse, Sam Sieber, Will
10	Les dépendances	Afshin, Andrii Abramov, GameScripting, Hillkorn, leeor, Matthias Braun, mcarlin, mszymborski, Will
11	Tâches de commande	Gabriele Mariotti
12	Utiliser des plugins tiers	Afterfield
13	Y compris la source native - Expérimental	iHowell