無料電子ブック

学習

gradle

Free unaffiliated eBook created from
Stack Overflow contributors.

#gradle

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.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

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

1: gradleをいめる

Gradleはオープンソースのビルドツールです。これはJavaコミュニティでくしており、Androidのツールです 。

ハイライトされたGradleの

- なビルドスクリプトは 、 GroovyまたはKotlinでかれたコードです。
- なコンベンションベースのアプローチをするくのコアプラグインとコミュニティプラグインは 、がされていないタスクがされないようにビルドします。
- MavenとIvyのみみ。したプラグインはnpmのようなの`repositories`からのをし`repositories`。
- ファーストクラスのマルチプロジェクトビルド。
- Maven 、 Antなどののビルドツールとの
- がしてGradleビルドをするをめるビルドスキャン 。

しくは

Gradleののについては、 Gradle User Guideの を してください 。

Gradleをしてみるには、ここでガイドをチェックしてみてください。 Javaのクイックスタートガイドをて、めてGradleをするをび、のビルドツールからすることができます。

Examples

Gradleのインストール

インストールされたJava JDKまたはJREGradle 3.xバージョンのはバージョン7

インストール

1. WebサイトからGradleのをダウンロードする
2. ZIPの
3. `GRADLE_HOME`をします。このは、ののされたファイルをすがあります。
4. `GRADLE_HOME/bin`を`PATH`にすると、コマンドラインインターフェイスCLIからGradleをできます。
5. CLIで`gradle -v`として、Gradleのインストールをテストします。には、インストールされているGradleのバージョンとのGradleののがまれているがあります

しいは、 のユーザーガイドにされています。

OS X / macOSでしてインストールする

homebrewのユーザーは、してgradleをインストールできます

```
brew install gradle
```

SdkManでインストールする

SdkManのユーザーは、 をしてGradleをインストールできます。

```
sdk install gradle
```

のバージョンをインストールする

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

バージョンをりえる

```
sdk use gradle 2.12
```

EclipseGradleプラグインのインストール

EclipseにGradleプラグインをインストールするためになはのとおりです。

1. Eclipseをき、[ヘルプ] - > [**Eclipse Marketplace]**にします
2. バーに**buildship**として**Enter**キーをします
3. **"Buildship Gradle Integration 1.0"**をし、 **Install**をクリックします。
4. のウィンドウで、[ ]をクリックします。
5. その、のとライセンスをけれてから、 **Finish**をクリックします。
6. インストール、Eclipseをし、「はい」をクリックします。

こんにちは

Gradleタスクは、プロジェクトbuild.gradleファイルのGroovyコードをしてできます。これらのタスクは、ターミナルで`> gradle [taskname]`をするか、EclipseなどのIDEからタスクをすることでできます。

GrailsでHello Worldのをするには、Groovyをしてをコンソールにするタスクをするがあります。Groovyの`println`を`println`て、Javaの`System.out.println`メソッドをびして、テキストをコンソールにします。

build.gradle

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

```
}
```

`> gradle hello`または`> gradle -q hello`をってこのタスクをすることができます。 `-q`は、gradleログメッセージをするためにされ、タスクののみがされます。

> gradle -q hello

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

タスクの

まずに`<<` leftShiftは`doLast {closure}`とです。 **gradle 3.2**からはされていません 。すべてのタスクコードは**build.gradle**にきまれています。

> タスクは、ビルドがするいくつかのアトミックなをします。これは、いくつかのクラスをコンパイルすること、JARをすること、Javadocをすること、またはいくつかのアーカイブをリポジトリにすることであるがあります。

Gradleは、きくけて2つのタスクをサポートしています。

いくつかのタスクスタイルをてみましょう

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

または

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

のはです。また、 `dependsOn` 、 `mustRunAfter` 、 `type`などのようにタスクにいくつかのをすることができます。タスクのにのようにアクションをすることでタスクをすることができます

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

これをすると、のようになります。

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

タスクのにするとここでべられたけ

2つのきなタイプのタスクについてしましょう。

シンプル

アクションクロージャでしたタスク

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

された

されているのは、あらかじめされたをつタスクです。プロジェクトでしているすべてのプラグインは、 タスクまたはタスクです 。たちをりましょう。あなたはそれがどのようにするのかします

```
task hello(type: HelloTask)

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

また、のようにパラメータをタスクにすこともできます。

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

これからはのようにタスクをきすことができます

```
   //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!'
}
```

これをすると、のようになります。

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

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

のすべてのについては、 サイトへのプラグイン

オンラインでgradleをいめるをむ https://riptutorial.com/ja/gradle/topic/894/gradleをいめる

2: Androidアプリケーションの Gradle スクリプトをしたインクリメントバージョン

Examples

ビルドにインクリメントメソッドをびす

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

インクリメントメソッドの

```
 /*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!")
    }
}
```

プロパティファイルからへのバージョンのみりとりて

def versionPropsFile = file '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!")
}
```

オンラインでAndroidアプリケーションのGradleスクリプトをしたインクリメントバージョンをむ
https://riptutorial.com/ja/gradle/topic/10696/androidアプリケーションのgradleスクリプトをしたイ

ンクリメントバージョン

3: Gradle Initスクリプト

Examples

すべてのプロジェクトにデフォルトリポジトリをする

ユーザーのgradleフォルダにinit.gradleをします。 init.gradleはすべてのプロジェクトでされます。

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

これらは、initスクリプトをにしてロードできるのです。

- **USER_HOME / .gradle / init.d**にある* **.gradle**ファイル
- Gradleインストールの**init.d**ディレクトリにある* **.gradle**ファイル

すべてのプロジェクトのリポジトリとしてmavenLocalをつinit.gradle

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

これにより、すべてのリポジトリでローカルのMavenキャッシュをできるようになります。ユースケースは、build.gradleにmavenLocalリポジトリをしたり、ネクサス/アーティファクトサーバをせずに、のプロジェクトに "gradle install"をけてjarをすることです。

オンラインでGradle Initスクリプトをむ https://riptutorial.com/ja/gradle/topic/4234/gradle-initスクリプト

4: Gradle Performance

Examples

ビルドのプロファイリング

パフォーマンスのためにGradleビルドをするに、ベースラインをして、ビルドのどのがもをやしているかをするがあります。これをうには、Gradleコマンドに`--profile`をしてビルドのプロファイルをします。

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

ビルドがすると、ビルドのHTMLプロファイルレポートが`./build/reports/profile/`、のようになります。

Profile report

Profiled build: build

Started on: 2016/07/23 - 17:47:33

Summary Configuration Depend

Description	Duration
Total Build Time	20.654s
Startup	0.598s
Settings and BuildSrc	0.001s
Loading Projects	0.003s
Configuring Projects	0.061s
Task Execution	19.611s

Generated by Gradle 2.14.1 at Jul 23, 2016 5:47:53 PM

[ ]のにあるタブをクリックすると、がどこでやされたかをよりにかります。

がするがあります。

あなたはによってオンデマンドモ—ドにをにすることができます `$GRADLE_USER_HOME/.gradle/gradle.properties ~/.gradle/gradle.properties`デフォルトでは、および `org.gradle.configureondemand`。

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

のプロジェクトにしてのみにするには、わりにそのプロジェクトの`gradle.properties`ファイルをします。

オンデマンドでをにすると、すべてのプロジェクトをにするのではなく、のタスクになプロジェクトのみをします。

Gradleマニュアルから

> オンデマンドモ—ドのでは、されたタスクにするプロジェクトのみをしようとします。つまり、ビルドにしているプロジェクトの`build.gradle`ファイルのみをします。このようにして、なマルチプロジェクトのをすることができます。には、このモ—ドはデフォルトモ—ドになります。であれば、Gradleビルドののモ—ドになります。

GradleのJVMメモリ—パラメ—タ—のチュ—ニング

`$GRADLE_USER_HOME/.gradle/gradle.properties` デフォルトでは`~/.gradle/gradle.properties` を `~/.gradle/gradle.properties`し、 `org.gradle.jvmargs`をすることで、GradleビルドとGradleデ—モンにされるメモリまたはそののJVMをまたはやすことができます`org.gradle.jvmargs`。

これらのをのプロジェクトにしてのみするには、わりにそのプロジェクトの`gradle.properties`ファイルをします。

GradleビルドとGradleデ—モンのデフォルトのメモリのはのとおりです。

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

これにより、1GBのなメモリりてヒ—プサイズと256MBのな ""オブジェクトにするメモリりてがになります。これらのサイズにすると、ガベ—ジコレクションがし、パフォ—マンスがにするがあります。

をっているとすると、のようにに2にすることができます

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

`XX:MaxPermSize`が`Xmx`をったには、 `XX:MaxPermSize`すぐにするのをめることができます。

Gradleデ—モンをする

Gradle Daemonをにすると、ビルドのパフォーマンスがします。

Gradleデーモンは、Gradle Frameworkをしてしたままにし、プロジェクトデータをメモリにキャッシュしてパフォーマンスをさせます。

シングルビルドの

ののためのデーモンをにするには、にすことができ`--daemon`あなたにを`gradle`コマンドやGradleのラッパースクリプト。

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

プロジェクトのすべてのビルドについて

プロジェクトのすべてのビルドにしてデーモンをにするには、をします。

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

プロジェクトの`gradle.properties`ファイル。

すべてのビルド

Gradleデーモンをデフォルトでにするには、システムのユーザーアカウントによってわれたすべてのビルドにして、 `$GRADLE_USER_HOME/.gradle/gradle.properties` デフォルトで `~/.gradle/gradle.properties` を`~/.gradle/gradle.properties`し、のをします。

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

また、Mac / Linux / * nixシステムののコマンドでこれをうこともできます

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

またはWindowsの

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

デーモンの

`--no-daemon`をしてのビルドのデーモンをにするか、プロジェクトの`gradle.properties`ファイルに`org.gradle.daemon=false`にしてのプロジェクトにして`--no-daemon`にすることができます。

デーモンの

デーモンプロセスをでするは、オペレーティングシステムのタスクマネージャをしてプロセスを

`gradle --stop`するか、 `gradle --stop`コマンドをします。 `--stop`スイッチは、GradleにコマンドをするためにされたじGradleバージョンののすべてのDaemonプロセスがするようにします。、デーモンプロセスは、* 3のがあったににします*。

Gradle Parallelビルド

Gradleは、プロジェクトのになく、デフォルトでに1つのタスクしかしません。 `--parallel`スイッチをすることで、Gradleに、またはなプロジェクトのないサブプロジェクトをにさせることができます。これらのタスクがするり、のタスクをにできますなるプロジェクト。

モードでプロジェクトをビルドするには

```
gradle build --parallel
```

また、プロジェクトのgradle.propertiesファイルにのをすることで、プロジェクトのデフォルトをにすることもできます。

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

のGradleバージョンをする

Gradleチームは、Gradleビルドのさまざまなのパフォーマンスをさせるためににいています。バージョンのGradleをしている、そののメリットをしてしまいます。 Gradleのバージョンにアップグレードして、どのようながあるかしてください。そうすることは、Gradleのマイナーバージョンでにわずかながこるため、リスクはいです。

Gradleラッパーのプロパティーファイルは、プロジェクトフォルダーの`gradle-wrapper.properties` `gradle/wrapper/`にあり、 `gradle-wrapper.properties`という`gradle-wrapper.properties`です。そのファイルのはのようになります。

```
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
```

でバージョン`XXX` のバージョンを`YYY` しいバージョンにすることができます。にラッパーをすると、しいバージョンがにダウンロードされます。

オンラインでGradle Performanceをむ https://riptutorial.com/ja/gradle/topic/3443/gradle-performance

5: Gradle Wrapper

Examples

Gradle WrapperとGit

でしたように、 `gradlew`コマンドがされたときにされるjarがプロジェクトにダウンロードされるため、 `gradlew` wrapperがします。しかし、これはコミットされないかもしれませんし、プロジェクトがチェックアウトされた、 `gradlew`はエラーでされません

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

これはあなたの.gitignoreにJavaプロジェクトの`*jar`がまれるがいためです。グラデルラッパーがされると、それは`gradle/wrapper/gradle-wrapper.jar`ファイルにコピーされ`gradle/wrapper/gradle-wrapper.jar`。したがって、gitインデックスにしてコミットするがあります。そうする

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

`-f`をすることで

Gradleラッパーの

Gradleには、プロジェクトにラッパーをするがあります。このラッパーは、すべてのユーザーまたはインテグレーションシステムがGradleをインストールするをします。また、プロジェクトがするバージョンとユーザーがインストールしたバージョンとのにがないバージョンのをします。これは、プロジェクトにローカルにバージョンのgradleをインストールすることでいます。

このプロジェクトのユーザーは、にをします。

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

ラッパーをするようにプロジェクトをセットアップするには、はのようにします。

1.

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

`--gradle-version X`はオプションで、されていないまたはにすようにラッパータスクがまれていない、されているバージョンはされているgradleのバージョンです。

1. プロジェクトでのバージョンをするようにするには、 `build.gradle`のをします。

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

`gradle wrapper`コマンドをすると、ファイルがされます。

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

こののドキュメントはhttps://docs.gradle.org/current/userguide/gradle_wrapper.htmlにあります。

Gradle Wrapperでローカルにされるgradleをする

オンプレミスのGradleのコピーをし、Wrapperがビルドでそれをできるようにするには、 `wrapper` タスクのコピーをす`distributionUrl`をします。

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

`gradlew` `gradle wrapper`をすると、シェルスクリプト`gradlew`がされ、 `gradle/wrapper/gradle-wrapper.properties`は、されたURLをしてGradleをダウンロードするようにされます。

プロキシのにあるGradle Wrapperの

ユーザーは、プロジェクトの`gradlew` 、つのなことをうということをすべきです。

1. ラッパーがしているgradleのバージョンがに/ .gradle / wrapper / distsにあるかどうかをしてください
2. そうでないは、インターネットからバージョンのアーカイブをダウンロードしてください

すべてのトラフィックがプロキシをするがあるにいる、ステップ2はするなプロキシでないり。その、 *JVM*プロキシパラメータがされていることをするがあります。

たとえば、なしでなプロキシをっているは、`JAVA_OPTS`または`GRADLE_OPTS`をのようにします。

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

したがって、したウィンドウのはのようになります。

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

ただし、にもがなは、のオプションもhttps://docs.oracle.com/javase/8/docs/api/java/net/doc-files/net-propertiesでしてください。 html 。

このプロキシは、リポジトリへのアクセスのプロキシにされています。

オンラインでGradle Wrapperをむ https://riptutorial.com/ja/gradle/topic/3006/gradle-wrapper

6: Gradleの

- タスク - ビルドがするアトミックな。タスクには、 `inputs` 、 `outputs` 、およびタスクのがあります。
- `dependencies {}` - タスクのにな`File`またはバイナリのをし`File`。 たとえば、 `org.slf4j:slf4j-api:1.7.21`は、Mavenへのです。
- `repositories {}` - のためにGradleがファイルをつける。に、グループ、、バージョンにされたファイルのまりです。 `jcenter()`は、 Bintray Mavenリポジトリである`maven { url 'http://jcenter.bintray.com/' }`なメソッドです 。

Examples

しいJavaライブラリの

Gradleのインストール

Gradleをインストールしたら、することでしいプロジェクトまたはのプロジェクトをセットアップできます

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

*Scala*のようなのプロジェクトタイプもありますが、このでは*Java*をします。

あなたはのようになります

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

は`gradle tasks`をし、 `build.gradle`ファイルがのものであっても、 `jar`をビルドし、 `test`し、`javadoc`することができることをできます。

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

```
repositories {
    jcenter()
}

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

オンラインでGradleのをむ https://riptutorial.com/ja/gradle/topic/2247/gradleの

7: Gradleプラグイン

Examples

`buildSrc`からのなgradleプラグイン

あなたのgradleプロジェクトのカスタムプラグインとDSLののな。
このサンプルでは、プラグインをする3つののうちの1つをしています。
3つのがあります

- をなして
- buildSrc
- スタンドアロンプラグイン

このは、 **buildSrc**フォルダからプラグインをするをしています。

このサンプルでは、5つのファイル

```
// 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
```

このをすると、DSLブロックのびしからされたをることができます

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

スタンドアロンプラグインの

javaをしてカスタムスタンドアロンのGradleプラグインをするにはGroovyもできます、のようなをするがあります。

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

セットアップのグラデル

`build.gradle`ファイルでプロジェクトをします。

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

dependencies {
    compile gradleApi()
}
```

`java`プラグインは、Javaコードのにされます。
`gradleApi()`は、Gradleプラグインのになすべてのメソッドとプロパティをします。

`settings.gradle`ファイルでのようにします。

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

Mavenにアーティファクト**ID**をします。
`settings.gradle`ファイルがプラグインディレクトリにしない、デフォルトはディレクトリのになります。

プラグインの

`Plugin`インタフェースをする`src/main/java/org/sample/MyPlugin.java`クラスをします。

```
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);
    }

}
```

`DefaultTask`クラスをするタスクをします。

```
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");
    }
}
```

プラグインクラスの

`META-INF/gradle-plugins`フォルダでは、プラグインクラスをする`implementation-class`プロパティをするプロパティファイルをするがあります。

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

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

プロパティ**filename**がプラグイン**ID**とすることにしてください。

それをする

`build.gradle`ファイルをして、プラグインをMavenリポジトリにアップロードするためのをします。

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

dependencies {
    compile gradleApi()
}
```

```
repositories {
    jcenter()
}


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

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

のコマンドをして、 `plugin/build.gradle`ファイルでされたMavenレポにGradleプラグインをビルドしてすることができます。

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

どうやってうのですか

プロジェクトの`build.gradle`でプラグインのをするには

```
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
```

に、のコマンドをしてタスクをびすことができます。

```
 $ ./gradlew myTask
```

オンラインでGradleプラグインをむ https://riptutorial.com/ja/gradle/topic/1900/gradleプラグイン

8: IntelliJ IDEAタスクのカスタマイズ

- groovy.util.Node = node.find {childNode - > trueをす|| false}
- node.appendnodeYouWantAsAChild
- groovy.util.Node parsedNode =しいXmlParser。parseTextsomeRawXMLString
- "'マルチラインされない'"

IntelliJプロジェクトの3つのファイルipr、iws、およびimlファイルには、アイデアタスクのgradleでアクセスすることができます。

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

.withXmlをすると、xmlにアクセスできます。 .asNodeをすると、Groovy XMLノードになります。

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

そこからかなりです - あなたのためにIntelliJプロジェクトをするためにgradleをし、にファイルをりし、IntelliJでgradleをしたいアクションをし、いファイルでしいファイルをします。アイデアジョブをカスタマイズするためになXMLがされます。また、XMLのどこにしているのかをメモするがあります。

すべきのもう1つは、グラデーションのアイデアをすると、IntelliJファイルにノードがしないようにすることです。したがって、するノードをしたいは、そのノードをしてすることができます。

とし

によっては、findメソッドのに==をするとします。テストするとはそれがであるとわかります、は.containsをします。

ノードをするときは、としてしているがすべてのノードにあるわけではありませんので、ずnullをしてください。

Examples

をする

このの

- あなたにはクラス`foo.bar.Baz`ます。
- メインメソッドをするをしたいとします。

- これは`fooBar`というモジュールにあります。

あなたのgradleファイル

```
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}"
        }
    }
}
```

オンラインでIntelliJ IDEAタスクのカスタマイズをむ
https://riptutorial.com/ja/gradle/topic/2297/intellij-ideaタスクのカスタマイズ

9: サードパーティのプラグインの

Examples

build.gradleにサードパーティのプラグインをする

Gradleすべてのバージョン このメソッドは、すべてのバージョンの*gradle*

buildscriptコードをbuild.gradleファイルのにします。

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

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

Gradleバージョン2.1+ このメソッドは、*Gradle 2.1*をするプロジェクトでのみします。

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

のサードパーティのプラグインをしたbuild.gradle

Gradleすべてのバージョン

のサードパーティプラグインをする、それらをbuildscriptAllまたはPlugin2.1+コードののインスタンスにするはなく、しいプラグインをのプラグインとにできます。

```
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バージョン2.1

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

オンラインでサードパーティのプラグインのをむ https://riptutorial.com/ja/gradle/topic/9183/サードパーティのプラグインの

10: タスクの

doLast

してください、 **gradle** 3.xよりなでは、タスク な **doLast {closure}**をするわりに、 "leftShift"<< をすることをおめします **leftleft**は**gralele** 5.0でです。 。

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

のものとです。

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

Examples

タスクをしたの

`dependsOn`メソッドでタスクのをできます。

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

`dependsOnをします

- タスクBはタスクAにする
- `B`タスクがされるにタスクをするため`A` Gradle。

はのとおりです。

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

のプロジェクトからをする

```
project('projectA') {
    task A(dependsOn: ':projectB:B') << {
```

```
        println 'Hello from A'
    }
}

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

のプロジェクトのタスクをするには、タスクののに `:projectB:B`というプロジェクトのパスを `:projectB:B`。

はのとおりです。

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

タスクオブジェクトをしたの

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

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

B.dependsOn A
```

これは、 タスクをするわりにをするのです。

そしてはじです

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

のの

のをできます。

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

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

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

```
}

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

これでのをできます。

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

はのとおりです。

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

そのの

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

はのとおりです。

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

dependsOn メソッドによるの

のをできます。

```
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'
}
```

はのとおりです。

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

オンラインでタスクのをむ https://riptutorial.com/ja/gradle/topic/5545/タスクの

11: ネイティブソースをむ.

パラメーター

パラメーター	
model.android.ndk.toolchain	ndk-bundleフォルダにあるネイティブツールチェイン

Examples

Basic JNI Gradle Config

rootbuild.gradle

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

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

アプリ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"
                    }
                }
            }
        }
    }
}
```

めビルドされたライブラリと**OpenGL ES 2.0**の

rootbuild.gradle

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

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

アプリ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
```

オンラインでネイティブソースをむ - をむ https://riptutorial.com/ja/gradle/topic/4460/ネイティブソースをむ---

12:

Examples

ローカルJARファイルのをする

シングルJAR

によっては、GradleビルドにとしてするがあるローカルJARファイルがあります。これをうはのとおりです

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

pathはファイルシステムのディレクトリパスで、 local_dependency.jarはローカルJARファイルのです。 pathは、ビルドファイルからのpathすることができます。

JARのディレクトリ

jarのディレクトリをしてコンパイルすることもできます。これはそうすることができます

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

ここで、 libsはjarと*.jarをむディレクトリになります。これには、ファイルにめるフィルタがあります。

リポジトリとしてのJARのディレクトリ

リポジトリのjarファイルをパスにするものとしてするのではなく、するがあるは、flatDirリポジトリをできます。

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

*libs*ディレクトリとそのディレクトリのjarファイルをします。

をする

Gradleのは、 Mavenとじフォーマットにいます。はのようにされます。

```
group:name:version
```

ここにがあります

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

コンパイルのとしてするには、このをGradleビルドファイルの`dependency`ブロックにするだけです。

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

このためののでは、のコンポーネントににをけます。

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

コンパイルにがされます。

は、テストにのみすることもできます。ここにがあります

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

のGradleプロジェクトにする

マルチプロジェクトのグラデルビルドの、ビルドののプロジェクトにするがあることがあります。これをするには、プロジェクトのにのようにします。

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

`':OtherProject'`は、ディレクトリのルートからされるプロジェクトの`':OtherProject'`です。

`':OtherProject'` `build.gradle`ファイルのコンテキストでできるようにするには、これをする`settings.gradle`し`settings.gradle`

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

よりなについては、 ここで Gradleのドキュメントをすることができます 。

リストの

`dependencies`タスクをびすと、ルートプロジェクトのをることができます。

```
gradle dependencies
```

そのは、によってされるグラフなをしたものです。されたをするには、すことができ`--configuration`するために1つのしたにくオプションを

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

サブプロジェクトのをするには、 `<subproject>:dependencies`タスクをします。たとえば、 `api`というのサブプロジェクトのをするには

```
gradle api:dependencies
```

リポジトリの

Gradleがプラグインのをすようにして、Gradleがそれらをつけることができるようにするがあります。これをうには、 `build.gradle repositories { ... }`をし`repositories { ... }`。

に、3つのリポジトリ、 JCenter 、 Mavenリポジトリ 、およびMavenスタイルのをするカスタムリポジトリをするをします。

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

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

gradleをって.aarファイルをAndroidプロジェクトにする

1. プロジェクトの`app`モジュールにし、 `libs`ディレクトリをします。
2. そこに`.aar`ファイルをきます。たとえば、 `myLib.aar`です。
3. のコードを`app`レベルの`build.gradle`ファイルの`android`ブロックにします。

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

こので、 `app`モジュールの`libs`フォルダをすのしいリポジトリをしました。

4. のコードを`dependencies`ブロックまたは`build.gradle`ファイルにします。

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

オンライ ンでをむ https://riptutorial.com/ja/gradle/topic/2524/

13: タスク

`mustRunAfter`と`shouldRunAfter`は「インキュベート」Gradle 3.0とマークされています。
これはなであり、のリリースでがされるがあることをしています。

なルールは2つあります。

- `mustRunAfter`
- `shouldRunAfter`

`mustRunAfter`けルールをするときは、taskAとtaskBのがされるたびにtaskBがにtaskAのにされるがあることをします。

`shouldRunAfter`けルールは、2つのでされるため、ていますがではありません。

- そのルールをしているは、サイクルがされます。
- をしていて、タスクのすべてのが`shouldRunAfter`タスクとはにたされている、このタスクは`shouldRunAfter`がされたかどうかにかかわらずされます。

Examples

mustRunAfterメソッドによる

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

B.mustRunAfter A
```

`B.mustRunAfter A`は、としてされたタスクのにタスクをするようにGradleにします。

はのとおりです。

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

けは、Aタスクと Bタスクのにをしませんが、 のタスクがのスケジュールされているにのみがあります。

これは、タスクAとBをしてできることをします。

はのとおりです。

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

オンラインでタスクをむ https://riptutorial.com/ja/gradle/topic/5550/タスク

クレジット

S. No		Contributors
1	gradleをいめる	Afterfield, bassim, Community, Emil Burzo, Eric Wendelin, Hamzawey, Hillkorn, Matthias Braun, Nikem, Pepper Lebeck-Jobe, Sergey Yakovlev, Stanislav, user2555595, vanogrid, Will
2	AndroidアプリケーションのGradleスクリプトをしたインクリメントバージョン	Jayakrishnan PM
3	Gradle Initスクリプト	ambes, Hillkorn
4	Gradle Performance	ambes, Sergey Yakovlev, Will
5	Gradle Wrapper	ajoberstar, Fanick, HankCa, I Stevenson
6	Gradleの	Eric Wendelin, Will
7	Gradleプラグイン	Gabriele Mariotti, JBirdVegas
8	IntelliJ IDEAタスクのカスタマイズ	IronHorse, Sam Sieber, Will
9	サードパーティのプラグインの	Afterfield
10	タスクの	Gabriele Mariotti, Sergey Yakovlev, Stanislav
11	ネイティブソースをむ -	iHowell
12		Afshin, Andrii Abramov, GameScripting, Hillkorn, leeor, Matthias Braun, mcarlin, mszymborski, Will
13	タスク	Gabriele Mariotti