Gradle Cheat Sheet

Javaプロジェクトの標準レイアウト

```
+ project
  + src
    + main
      + java
      + resources
    + test
      + java
      + resources
  - build.gradle
  - gradle.properties
  - settings.gradle
```

build.gradle のテンプレート

```
// plugins
apply plugin: "java"
apply plugin: "maven"
apply plugin: "groovy"
apply plugin: "application"

// default tasks
defaultTasks 'clean', 'build'

// properties
sourceCompatibility = '1.7'
def defaultEncoding = 'UTF-8'
[
  compileJava,
  compileTestJava,
  javadoc
]*.options*.encoding = defaultEncoding

// repositories
repositories {
  mavenCentral()
  mavenLocal()
  maven {
    url "http://maven.seasar.org/maven2"
    url "http://repository.jboss.org/nexus/content/groups/public-jboss/"
  }
}

// configurations
configurations {
  doc // extra configuration
}

// dependencies
dependencies {
  compile 'org.codehaus.groovy:groovy-all:2.0.5'
  // runtime
  testCompile 'junit:junit:4.11'
  // testRuntime

  // extra configuration
  doc 'g:m:v@zip' // @ext

  // providedCompile(War plugin)
  // providedRuntime(War plugin)
}

// tasks
```

大元であるProjectのプロパティやメソッドが呼ばれる。例えばプラグインを適用するapplyはProject.apply()の呼び出し。

build.gradle の分割

```
apply from: "gradle/foo.gradle"
```

コマンドラインオプション

オプション	説明
-i	ログレベルをinfoにする
-q	Gradleのログメッセージを抑制し、タスクによる出力のみを表示させる
-a	マルチプロジェクトのビルド時に、依存するプロジェクトのビルドを行わないようにする
-m	空実行。実行されるタスクの順番を調べるために使う
--daemon	デーモンモードでビルドを実行する
--offline	常にキャッシュされた依存モジュールを使う

よく使うタスク

まったくプラグインを適用していない場合でも使えるヘルプタスク

タスク名	説明
tasks	タスク一覧を出力
dependencies	依存関係一覧を出力
projects	サブプロジェクト一覧を出力（マルチプロジェクトを使っている場合に使う）
properties	プロパティ一覧を出力

プラグインを適用して使えるようになるタスク

プラグイン	タスク名	説明
java	jar	JARファイルを作成
java	assemble	JAR（やWARやEAR）を作成
java	test	ソースコードをテスト
java	check	テストし、検証タスクを実行する。testタスクに依存
java	javadoc	JavaDocを生成
java	build	すべてのアーカイブ作成、テスト実行、検証タスクを実行
java	clean	プロジェクトのビルドディレクトリを削除
war	war	WARファイルを作成
ear	ear	EARファイルを作成
java & maven	install	アーティファクトをリポジトリに登録する

タスクプロパティによるタスクのカスタマイズ

```
// 個別に
fooTask {
  fooTaskProperty = "xxx"
  fooMethod "yyy"
}

// まとめて
[barTask, bazTask]*.barbazProperty = "zzz"
```

タスクにどのようなプロパティがあるかを調べるには、そのタスクのTask Typeを辿ればよい。
例1: javaプラグインで追加されるcleanタスクのTask TypeはDelete。
例2: warプラグインで追加されるwarタスクのTask TypeはWar。

よく使うプロパティ

プラグイン	プロパティ名	型	説明
標準	rootProject	Project	ルートプロジェクト
標準	rootDir	File	プロジェクトのルートディレクトリ
標準	buildDir	File	ビルドディレクトリ
java	sourceSets	SourceSetContainer	ソースセット（デフォルトでmainとjava）
java	sourceCompatibility	JavaVersion	コンパイル時に使用するJavaのバージョン(1.7など)
java	manifest	Manifest	マニフェスト

拡張プロパティ

```
// 全体
ext {
  springVersion = "3.1.0.RELEASE"
  emailNotification = "build@master.org"
}

// 他のオブジェクト
sourceSets.all { ext.purpose = null } // 1. プロパティを追加
sourceSets.main.purpose = "production" // 2. プロパティに値をセット
```

プロパティを追加せず、直接値をセットしても（1.6時点では）エラーにならないが、非推奨である（以下の警告が出力される）。拡張プロパティを追加する時は、extを使うことを推奨。

```
Deprecated dynamic property: "purpose" on "source set 'main'", value: "production".
```

よく使う処理

copy

```
// 方法1: Copyタスクを使う場合
task myCopy(type: Copy) << {
  from 'src/*.txt'
  into 'dest'
}

// 方法2: Project.copy() メソッドを使う場合
task myCopy2 << {
  copy {
    from 'src/*.txt'
    into buildDir
  }
}
```

単なるコピーだけならCopyタスクの方がよい。 Project.copy() メソッドは他のタスクの中に組み込んで使うことが多い。

mkdir

```
task myMkdir << {
  file('tmp').mkdir()
}
```

unzip

```
task myUnzip << {
  copy {
    from zipTree('aaa.zip')
    into buildDir
  }
}
```

tar

```
task myTar(type: Tar) {
  compression = Compression.GZIP // NONE/GZIP/BZIP2
  from 'content'

  destinationDir = file('dest') // default: project.distsDir = "build/distributions"
}
```

外部コマンド実行

```
// 方法1: Copyタスクを使う場合
task myExec(type: Exec) << {
  commandLine 'echo', 'hello'
}

// 方法2: groovy の execute() メソッドを使う場合
task myCopy << {
  ['echo', 'hello'].execute()
}
```

ファイルコレクション

```
# Using a relative path
File configFile = file('src/config.xml')

# Using a File object
configFile = file(new File('src/config.xml'))

# FileCollection
FileCollection collection = files('src/file1.txt', new File('src/file2.txt'), ['src/file3.txt'])
collection.each { File file -> println file.name }
String path = collection.asPath

# Create a file tree using path
```

```
FileTree tree = fileTree(dir: 'src/main').include('**/*.java')

# Create a file tree using closure
tree = fileTree('src') {
  include '**/*.java'
}

# Create a file tree using a map
tree = fileTree(dir: 'src', include: '**/*.java')
```

ファイルコレクションは、コピー元ファイル、ファイル依存関係などの指定で使われる。

タスクの定義

基本的な作り方

```
task hello << {
  println "hello!"
}

// 拡張タスクプロパティ
task myTask {
  ext.myProperty = "myValue"
}
```

タスク型Task typeを使う場合

```
task archive (type: Zip) {
  from "src"
  // "build/distributions/xxx.zip"
}
```

Task typeはCopy, Zip, Tar, JavaDocあたりが頻出。

その他

```
// 依存関係をつける
task taskX(dependsOn: 'taskY') << {
  println 'taskX'
}

// 置き換え
task taskZ(overwrite: true) << {
  println 'taskZ'
}
```

アーティファクト

```
artifacts {
  archives jar
}

install
  repositories {
    mavenInstaller {
      pom.groupId = 'com.github.tq-jappy'
      pom.version = '1.0.0-SNAPSHOT'
      pom.artifactId = 'example'
    }
  }
}
```

実行可能Jar

```
jar {
    copy {
        from configurations.compile
        into "build/distribution/lib"
    }
    def manifestClasspath = configurations.compile.collect{ 'lib/' + it.getName() }.join(' ')
    manifest {
        attributes "Main-Class" : "foo.bar.Main"
        attributes 'Class-Path': manifestClasspath
    }
    from (configurations.compile.resolve().collect { it.isDirectory() ? it : fileTree(it) }) {
        exclude 'META-INF/MANIFEST.MF'
        exclude 'META-INF/*.SF'
        exclude 'META-INF/*.DSA'
        exclude 'META-INF/*.RSA'
        exclude '**/*.jar'
    }
    destinationDir = file("build/distribution")
}
```

実行可能FatJar

```
jar {
  from configurations.compile.collect { it.isDirectory() ? it : zipTree(it) }
  manifest.mainAttributes("Main-Class" : "foo.bar.Main")
}
```

War

```
apply plugin: "war"

configurations {
  moreLibs
}

war {
  from 'src/main/webapp'
  classPath configurations.moreLibs
}
```

Ear

```
apply plugin: "ear"

dependencies {
  deploy project(':war')

  earlib 'log4j:log4j:1.2.15@jar'
}

war {
  appDirName 'src/main/app'
}
```

マルチプロジェクト

階層

```
+ parent
  + sub1
  + sub2
  + sub3
```

ルートプロジェクトの parent/build.gradle

```
allprojects {
  task hello << {task -> println "I'm $task.project.name" }
}
subprojects {
  hello << {println "- I depend on water"}
}
project(':sub1').hello << {
  println "- I'm the largest animal that has ever lived on this planet."
}
```

ルートプロジェクトの parent/settings.gradle

```
include "sub1", "sub2", "sub3"
```

サブプロジェクト間の依存関係

```
dependencies {
  compile project(':sub1')
  compile project(path: ':sub2', configuration: 'abc')
}
```

サブプロジェクトのタスクを実行

```
:sub1:build
```

フラット

```
+ parent
+ sub1
+ sub2
+ sub3
```

ルートプロジェクトの parent/settings.gradle

```
includeFlat 'sub1', 'sub2', 'sub3'
```

静的解析

build.xml

```
apply plugin: "checkstyle"
apply plugin: "findbugs"

buildscript {
  apply from: 'https://github.com/valkolovos/gradle_cobertura/raw/master/repo/'
            + 'gradle_cobertura/gradle_cobertura/1.2.1/coberturainit.gradle'
}

test.jvmArgs '-XX:-UseSplitVerifier'
[checkstyleMain, checkstyleTest, findbugsMain, findbugsTest]*.ignoreFailures = true
[checkstyleTest, findbugsTest]*.excludes = ['**/*']
// checkStyleMain {
//   configFile = file('config/checkstyle/checkstyle.xml')
// }
```

タスク

- check
- coberturaMain

レポート

- build/reports/checkstyle/main.xml
- build/reports/findbugs/main.xml
- build/reports/cobertura/coverage.xml

依存関係の管理

```
dependencies {
  compile 'org.springframework:spring-core:2.5'
  // アーティファクトオンリー記法
  compile 'org.gradle.test.classifiers:service:1.0:jdk15@jar'
  // 推移的な依存関係の除外
  compile 'org.hibernate:hibernate:3.0.5') {
     transitive = true
  }

  // ローカルのファイル依存関係
  compile fileTree(dir: 'libs', include: '*.jar')
}
```

依存関係のキャッシュ

dependencies に記述した依存するサードパーティのアーティファクト（依存モジュール）は ${GRADLE_USER_HOME} > ${USER_HOME}/.gradle/cache 以下にキャッシュされる。 Mavenキャッシュと管理方法が異なるので、そのまま Maven リポジトリとして公開はできない、

警告

Jenkins Gradle Plugin 1.22 では GRADLE_USER_HOME は Jenkins の ワークスペース(例えば /var/lib/jenkins/workspace/job1)がセットされる。
Workspace Cleanup Plugin などを使ってビルド前にワークスペースをクリーンしていると、 毎回ローカルキャッシュも削除されてしまい、ビルドの度に ライブラリを毎回ダウンロードすることになってしまうので注意（最新の1.23では解消されており、 /var/lib/jenkins/workspace/.gradle にキャッシュされる）

Ant

```
task hello1 << {
  ant.echo(message: 'hello1')
}

task hello2 << {
  ant {
    echo(message: 'hello1')
  }
}
```

Wrapper

```
task wrapper(type: Wrapper) {
  gradleVersion = '1.6'
}
```

gradlew で実行。
1.7以降はタスクを作る必要がなくなる（予定らしい）

gradle.properties

```
# proxy settings
systemProp.http.proxyHost=http://proxy:8080/
systemProp.http.proxyPort=http://proxy:8080/
systemProp.https.proxyHost=http://proxy:8080/
systemProp.https.proxyPort=http://proxy:8080/
```