

# G\*ワークショップZ Gradleハンズオン

2013.03.15(3.18更新)  
須江信洋(@nobusue)

<http://d.hatena.ne.jp/nobusue>  
<https://www.facebook.com/nobuhiro.sue>

# 自己紹介

- ▶ 須江 信洋(すえ のぶひろ)
  - ▶ Twitter: @nobusue
  - ▶ <https://www.facebook.com/nobuhiro.sue>
- ▶ 約10年ほどJavaEE関連の仕事をしています
- ▶ 最近はPhoneGap(Cordova)とかがメイン
- ▶ G\*(Groovy関連技術)との関わり
  - ▶ JGGUGサポートスタッフ
  - ▶ 「プログラミングGROOVY」執筆チーム
  - ▶ 「Groovy イン・アクション」翻訳チーム
  - ▶ Groovyで作ったBot飼ってます(@hatena\_groovy)



# agenda

---

- ▶ ~~Gradle概要~~-(thanks to 林さん)
- ▶ Gradleインストール
- ▶ Hello Gradle
- ▶ Gradle Tools
- ▶ Gradle Quickstart
- ▶ Gradle Build Language概要
- ▶ Ant integration
- ▶ IDE integration
- ▶ Jenkins integration
- ▶ 便利なGradleプラグイン紹介
- ▶ 参考情報

# 本日のサンプルコード

---

- ▶ GitHubから入手してください
  - ▶ <https://github.com/nobusue/GradleHandson>
  - ▶ git clone  
https://github.com/nobusue/GradleHandson
- ▶ より詳しい例はGradle公式サンプルを参照
  - ▶ <http://www.gradle.org/downloads>
  - ▶ gradle-1.4-all.zipの"samples"ディレクトリ

# Gradleインストール

---

## ▶ 前提

- ▶ JDK1.5以上 (“java -version”で確認)

## ▶ GVM利用

- ▶ `curl -s get.gvmtool.net | bash`
- ▶ `gvm install gradle`
- ▶ 詳細は <http://gvmtool.net/> 参照
- ▶ Windowsの場合はcygwinが必要

## ▶ ZIPを展開

- ▶ <http://www.gradle.org/downloads>
- ▶ `gradle-1.4-all.zip`か`gradle-1.4-bin.zip`
- ▶ 適当なディレクトリに展開 (`$GRADLE_HOME`)
- ▶ `$GRADLE_HOME/bin` にパスを通しておく

# 動作確認

## ▶ gradle -v

```
C:\Users\nobusue>gradle -v
```

```
-----  
Gradle 1.4  
-----
```

```
Gradle build time: 2013年1月28日 (月曜日) 3時42分46秒 UTC  
Groovy: 1.8.6  
Ant: Apache Ant(TM) version 1.8.4 compiled on May 22 2012  
Ivy: 2.2.0  
JVM: 1.6.0 (IBM Corporation 2.4)  
OS: Windows 7 6.1 build 7601 Service Pack 1 amd64
```

GVMでインストールした場合は ~/.gvm/gradle/1.4 以下に導入され、  
~/.gvm/gradle/current にシンボリックリンクが作成されます

# Hello Gradle

---

- ▶ 適当な作業ディレクトリを作成し、カレントディレクトリを移動します
- ▶ 以下の内容で“build.gradle”を作成します

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

- ▶ “gradle helloWorld” を実行します

```
C:\Users\nobusue\work\hello>gradle helloWorld  
:helloWorld  
Hello world.  
  
BUILD SUCCESSFUL  
  
Total time: 2.605 secs
```

# Hello Gradle解説

---

タスクの定義

タスクにクロージャを追加  
※ leftShift()の省略記法

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

Groovyのprintln文



# デフォルトGradleタスク

## ▶ gradle tasks

```
-----  
All tasks runnable from root project  
-----
```

```
Help tasks  
-----
```

```
dependencies - Displays all dependencies declared in root project 'hello'.  
dependencyInsight - Displays the insight into a specific dependency in root project 'hello'.  
help - Displays a help message  
projects - Displays the sub-projects of root project 'hello'.  
properties - Displays the properties of root project 'hello'.  
tasks - Displays the tasks runnable from root project 'hello' (some of the displayed tasks may belong to subprojects).
```

```
Other tasks  
-----
```

```
helloWorld
```

# Gradle Tools

---

- ▶ gradle command
- ▶ gradle daemon
- ▶ gradle-gui

# gradle command

## ► gradle --help

```
USAGE: gradle [option...] [task...]

-?, -h, --help            Shows this help message.
-a, --no-rebuild           Do not rebuild project dependencies.
-b, --build-file           Specifies the build file.
-C, --cache               Specifies how compiled build scripts should be cached. Possible values are: 'rebuild' and 'on'. Default value is 'on' [deprecated - Use '--rerun-tasks' or '--recompile-scripts' instead]
-c, --settings-file       Specifies the settings file.
--continue                Continues task execution after a task failure.
-D, --system-prop         Set system property of the JVM (e.g. -Dmyprop=myvalue).
-d, --debug               Log in debug mode (includes normal stacktrace).
--daemon                  Uses the Gradle daemon to run the build. Starts the daemon if not running.
--foreground              Starts the Gradle daemon in the foreground. [incubating]

-g, --gradle-user-home    Specifies the gradle user home directory.
--gui                     Launches the Gradle GUI.
-I, --init-script          Specifies an initialization script.
-i, --info                Set log level to info.
-m, --dry-run             Runs the builds with all task actions disabled.
--no-color                Do not use color in the console output.
--no-daemon               Do not use the Gradle daemon to run the build.
```

# gradle commandの重要オプション

---

## ▶ ログ関連

- ▶ --quiet(-q) ログ出力を抑制
- ▶ --info(-i) / --debug(-d)
- ▶ --stacktrace(-s) / --full-stacktrace(-S)

## ▶ ファイル/ディレクトリ指定

- ▶ --build-file(-b) build.gradle以外
- ▶ --project-dir(-p) カレントディレクトリ以外

## ▶ テスト

- ▶ --dry-run(-m) タスクの実行順序のみ確認

# 環境まわりのまとめ

---

## ▶ 環境変数

- ▶ JAVA\_OPTS ⇒ JVM全体に影響
- ▶ GRADLE\_OPTS ⇒ gradleのみ影響

## ▶ 初期化スクリプト(init scripts)

- ▶ ビルドの開始前に実行される
- ▶ ~/.gradle/init.gradle
- ▶ ~/.gradle/init.d/\*.gradle
- ▶ \$GRADLE\_HOME/init.d/\*.gradle
- ▶ --init-script(-I)で指定することも可能

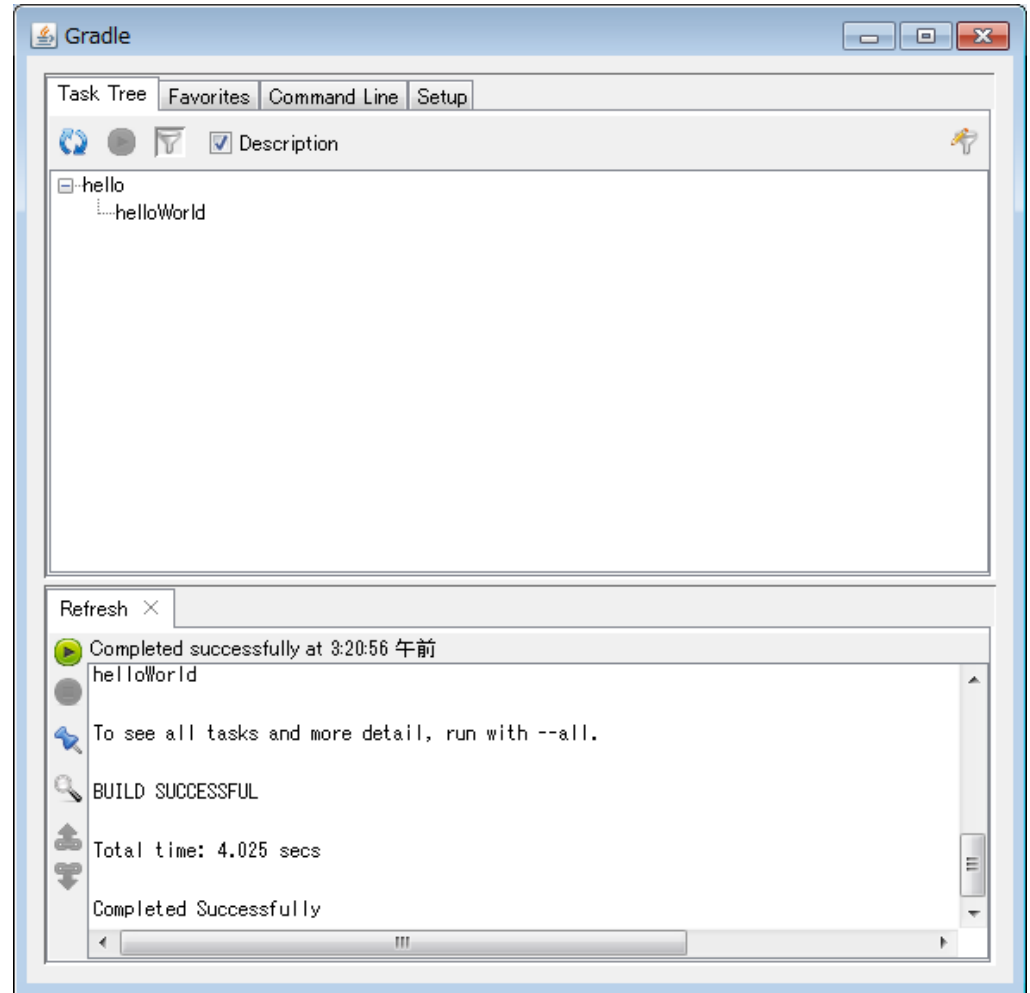
# gradle daemon

---

- ▶ gradleのプロセスを常駐して起動を高速化
- ▶ 起動
  - ▶ `gradle --daemon helloWorld`
- ▶ 停止
  - ▶ `gradle --stop`
- ▶ 起動済みのプロセスを使わない
  - ▶ `gradle -no-daemon helloWorld`
- ▶ 常にdaemonを使う場合は以下いずれか
  - ▶ `alias gradled='gradle --daemon'`
  - ▶ `export GRADLE_OPTS="-Dorg.gradle.daemon=true"`

# gradle-gui

## ▶ gradle --gui



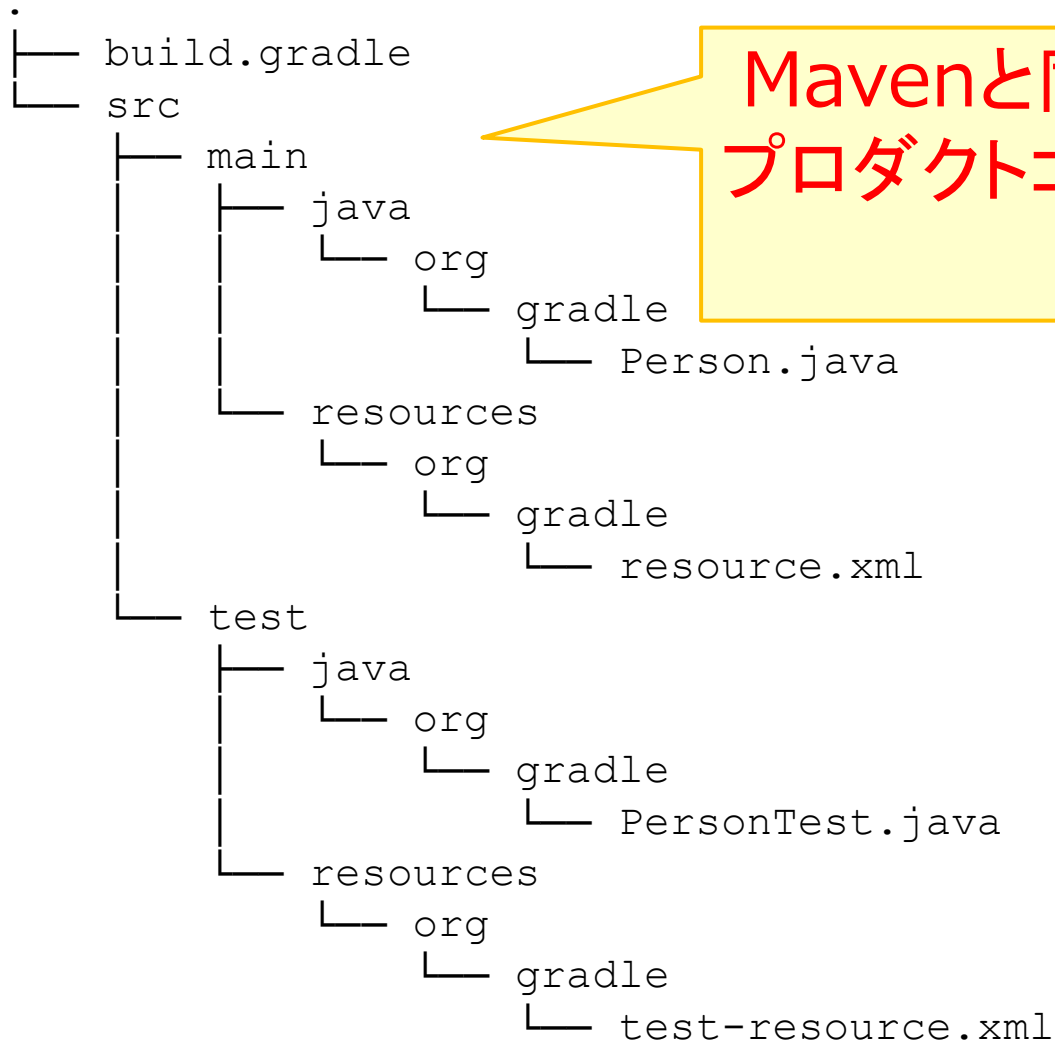
# Gradle Quickstart

---

- ▶ Java project
- ▶ Groovy project
- ▶ Web project



# Javaプロジェクトのレイアウト



Mavenと同様の規約に従って  
プロダクトコードとテストコードを  
配置

# Javaプロジェクトのビルドスクリプト(抜粋)

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

Javaプラグインを適用

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

依存性解決にMavenリポジトリを利用

```
dependencies {  
    compile(  
        group: 'commons-collections',  
        name: 'commons-collections',  
        version: '3.2')
```

プロダクトコードのコンパイル時の依存先

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

テストコードのコンパイル時の依存先

```
}
```

# Javaプラグインが追加するタスク

---

## Build tasks

-----

assemble - Assembles the outputs of this project.

build - Assembles and tests this project.

buildDependents - Assembles and tests this project and all projects that depend on it.

buildNeeded - Assembles and tests this project and all projects it depends on.

classes - Assembles the main classes.

clean - Deletes the build directory.

jar - Assembles a jar archive containing the main classes.

testClasses - Assembles the test classes.

## Documentation tasks

-----

javadoc - Generates Javadoc API documentation for the main source code.

# Javaプラグインが追加するタスク

---

## Upload tasks

-----

uploadArchives - Uploads all artifacts belonging to configuration ':archives'

## Verification tasks

-----

check - Runs all checks.

test - Runs the unit tests.

## Rules

-----

Pattern: build<ConfigurationName>: Assembles the artifacts of a configuration.

Pattern: upload<ConfigurationName>: Assembles and uploads the artifacts belonging to a configuration.

Pattern: clean<TaskName>: Cleans the output files of a task.

# Groovyプロジェクトのレイアウト

---

```
.
├── build.gradle
├── src
│   ├── main
│   │   ├── groovy
│   │   │   ├── org
│   │   │   │   └── gradle
│   │   │   │       └── Person.groovy
│   │   └── resources
│   │       ├── resource.txt
│   │       └── script.groovy
│   └── test
│       ├── groovy
│       │   ├── org
│       │   │   └── gradle
│       │   │       └── PersonSpec.groovy
│       └── resources
│           ├── testResource.txt
│           └── testScript.groovy
```

# Groovyプロジェクトのビルドスクリプト

```
apply plugin: 'groovy'
```

Groovyプラグインを適用

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

依存性解決にMavenリポジトリを利用

```
dependencies {  
    compile 'org.codehaus.groovy:groovy-all:2.0.5'  
    testCompile "org.spockframework:spock-core:0.7-groovy-2.0"  
}
```

プロダクトコードのコンパイル時の依存先

テストコードのコンパイル時の依存先

# Groovyプラグインが追加するタスク

---

## Documentation tasks

-----

javadoc - Generates Javadoc API documentation for the main source code.

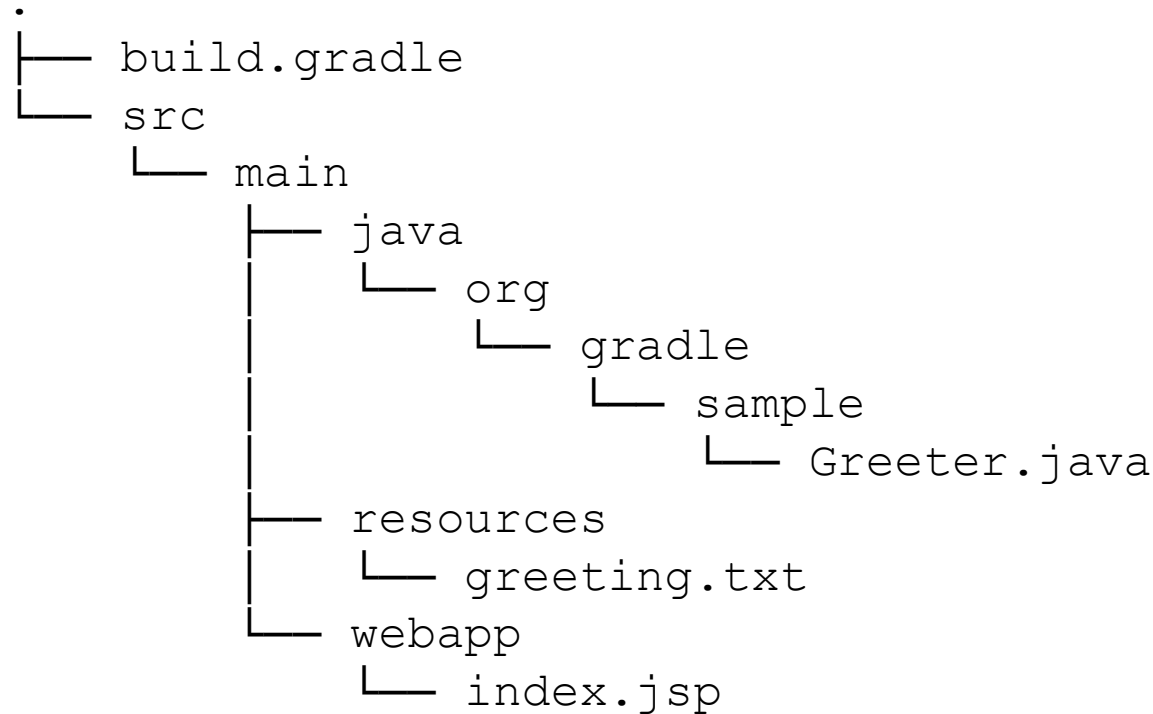
## Documentation tasks

**groovydoc - Generates Groovydoc API documentation for the main source code.**

GroovyプラグインはJavaプラグインを拡張して作られているため、Javaプラグインが提供するタスクはそのまま利用可能  
(ただし、一部のタスクはGroovy用に機能拡張されている)

# Webappプロジェクトのレイアウト

---





# Webappプロジェクトのビルドスクリプト

**apply** plugin: 'war'

Warプラグインを適用

**apply** plugin: 'jetty'

Jettyプラグインを利用して  
ビルドスクリプト上でwarを実行可能

**repositories** {  
 mavenCentral()  
}

依存性解決にMavenリポジトリを利用

**dependencies** {  
 compile group: 'commons-io', name: 'commons-io', version: '1.4'  
 compile group: 'log4j', name: 'log4j', version: '1.2.15', ext: 'jar'  
}

プロダクトコードのコンパイル時の依存先

httpPort = 8080  
stopPort = 9451  
stopKey = 'foo'

Jettyプラグインのパラメータ

# Webappプラグインが追加するタスク

---

## Build tasks

-----

war - Generates a war archive with all the compiled classes, the web-app content and the libraries.

## Web application tasks

-----

jettyRun - Uses your files as and where they are and deploys them to Jetty.

jettyRunWar - Assembles the webapp into a war and deploys it to Jetty.

jettyStop - Stops Jetty.

WebappプラグインはJavaプラグインを拡張して作られているため、Javaプラグインが提供するタスクはそのまま利用可能  
(ただし、一部のタスクはWebapp用に機能拡張されている)

# Gradle Build Language概要

---

- ▶ Gradle DSL
- ▶ Gradle Domain Objects

# Gradle DSLのエントリーポイント

## ▶ Gradleビルド言語リファレンス

- ▶ <http://gradle.monochromeroad.com/docs/dsl/index.html>

### Home

はじめに  
いくつかの基本  
ビルドスクリプトの構造  
核となる型  
コンテナ型  
補助タスク型  
タスク型  
Eclipse/IDEAモデル型  
Eclipse/IDEAタスク型

### ビルドスクリプトのブロック

#### Build script blocks

```
allprojects { }  
artifacts { }  
buildscript { }  
configurations { }
```

## Gradleビルド言語リファレンス

Version 1.5-20130308082157+0000

### はじめに

このリファレンスガイドは、Gradleビルド言語、DSLを構成する、様々な型について説明するものです。

### いくつかの基本

あなたが理解すべきなのは少々の基本構想で、それらはあなたがGradleスクリプトを記述する際に助けになるでしょう。

# タスク定義

```
task('hello') << {  
    println "hello"  
}
```

```
task('copy', type: Copy) {  
    from(file('srcDir'))  
    into(buildDir)  
}
```

```
task(hello) << {  
    println "hello"  
}
```

```
task(copy, type: Copy) {  
    from(file('srcDir'))  
    into(buildDir)  
}
```

```
task copy(type: Copy) {  
    description = 'Copies the resource directory to the target directory.'  
    from 'resources'  
    into 'target'  
    include('**/*.txt', '**/*.xml', '**/*.properties')  
}
```

# デフォルトタスク

---

```
defaultTasks 'clean', 'run'
```

```
task clean << {  
    println 'Default Cleaning!'  
}
```

```
task run << {  
    println 'Default Running!'  
}
```

```
task other << {  
    println "I'm not a default task!"  
}
```

# タスク依存関係(task利用)

```
task taskX << {  
    println 'taskX'  
}
```

```
task taskY << {  
    println 'taskY'  
}
```

```
taskX.dependsOn taskY
```

タスク名でtaskオブジェクトを参照

# タスク依存関係(クロージャ利用)

```
task taskX << {  
  println 'taskX'  
}  
taskX.dependsOn {  
  tasks.findAll { task -> task.name.startsWith('lib') }  
}  
  
task lib1 << {  
  println 'lib1'  
}  
task lib2 << {  
  println 'lib2'  
}  
task notALib << {  
  println 'notALib'  
}
```

taskのコレクションを返すクロージャ



# description / グループ化

```
def taskGroup = 'base'
task first(description: 'Base task', group: taskGroup) << {
    println "I am first task"
}
```

groupを指定

```
task second(dependsOn: first, description: 'Secondary task',
group: taskGroup) << {
    println "I am second task"
}
```

```
-----
All tasks runnable from root project
-----
```

```
Base tasks
```

```
-----
```

```
first - Base task
```

```
second - Secondary task
```

```
Help tasks
```

```
-----
```

```
dependencies - Displays all dependencies declared in root project
```

# 動的タスク定義

```
4.times { counter ->
  task "task${counter}" << {
    println "${counter+1}番目の動的タスクです"
  }
}
```

```
Other tasks
-----
task0
task1
task2
task3
```

```
C:\work\gradle\dynamic>gradle task1
:task1
2番目の動的タスクです

BUILD SUCCESSFUL
```

# 条件分岐

```
task "OsDependTask" << {  
    def os = System.getProperty("os.name")  
    if(os.contains("Windows")) {  
        println "Windows用の処理" }  
    else if(os.contains("Mac OS")) {  
        println "Mac OS用の処理" }  
    else { println "Linux/Unix用の処理" }  
}
```

```
C:¥work¥gradle¥condition>gradle OsDependTask  
:OsDependTask  
Windows用の処理  
  
BUILD SUCCESSFUL
```

# Gradleの代表オブジェクト

---

- ▶ `org.gradle.api.Project`
  - ▶ ビルドスクリプト(`build.gradle`)に対応
  - ▶ 中核となるオブジェクト
  - ▶ ビルドスクリプト内では暗黙的に、もしくは`project`プロパティで参照

# Gradleの代表オブジェクト

---

- ▶ `org.gradle.api.invocation.Gradle`
  - ▶ 実行中のビルドエンジンに対応
  - ▶ 初期化時の情報や、環境情報を保持
  - ▶ `Project.getGradle()` で取得可能

# Gradleの代表オブジェクト

---

- ▶ `org.gradle.api.initialization.Settings`
  - ▶ `settings.gradle`に対応
  - ▶ マルチプロジェクト構成時にプロジェクト階層の情報を保持

# 便利メソッド/プロパティ

---

- ▶ `file()`
  - ▶ 相対/絶対パス、Fileオブジェクト、URLなど
  - ▶ PathValidationでいろいろ判定可能
- ▶ `files()`
  - ▶ ファイルのコレクション(filter可能)
  - ▶ 引数としてファイルを返すtaskを渡せる
- ▶ `fileTree()`
  - ▶ ファイルのツリー階層をトラバース(visit)可能
  - ▶ Antのpathelement式でinclude/exclude可能
- ▶ `logger`
  - ▶ SLF4J Loggerのインスタンス
  - ▶ 標準出力への出力はQUIETレベルにリダイレクト

# 拡張プロパティ

---

- ▶ Gradleのドメインモデルにプロパティを追加する際には、extプロパティ(extブロック)を使うこと
  - ▶ Gradle-1.0M9からこちらが強く推奨されています
  - ▶ 現在は互換性のため未定義のプロパティがあってもエラーにならないが、警告が出ます
- ▶ ローカル変数(def)と異なり、ドメインモデルのライフサイクル全体で利用できます
- ▶ 詳細はこちらの「13.4.2. 拡張プロパティ」を参照
  - ▶ [http://gradle.monochromeroad.com/docs/userguide/writing\\_build\\_scripts.html](http://gradle.monochromeroad.com/docs/userguide/writing_build_scripts.html)



# Ant integration

---

- ▶ Antタスクの利用
- ▶ Antビルド定義の利用

# AntからGradleへ

---

- ▶ Gradleは既存のAnt資産を活用できる
  - ▶ Antのbuild.xmlをそのまま読み込んで実行可能
    - ▶ Antターゲット=Gradleタスク
  - ▶ AntタスクをGradleから直接利用可能
    - ▶ GroovyのAntBuilderが組み込まれている
  - ▶ AntタスクとGradleタスクを共存することも可能
    - ▶ 相互に依存するタスクも定義できる
    - ▶ AntタスクをGradleから拡張することもできる
- ▶ Gradleは「Better Ant」としても使える
  - ▶ Mavenとの大きな違い
  - ▶ Antから段階的にGradleへ移行できる

# GradleでAntタスクを利用

---

```
task hello << {  
    ant.echo('Antタスクの実行')  
}
```

```
Other tasks  
-----  
hello
```

```
C:\work\gradle\anttask>gradle hello  
:hello  
[ant:echo] Antタスクの実行  
  
BUILD SUCCESSFUL
```

# GradleでAntのビルド定義を利用

[build.gradle]

```
ant.importBuild 'build.xml'
```

[build.xml]

```
<project>
  <target name="hello">
    <echo>Antターゲットの実行</echo>
  </target>
</project>
```

```
Other tasks
-----
hello
```

```
C:\work\gradle\ant\import>gradle hello
:hello
[ant:echo] Antターゲットの実行

BUILD SUCCESSFUL
```

# IDE integration

---

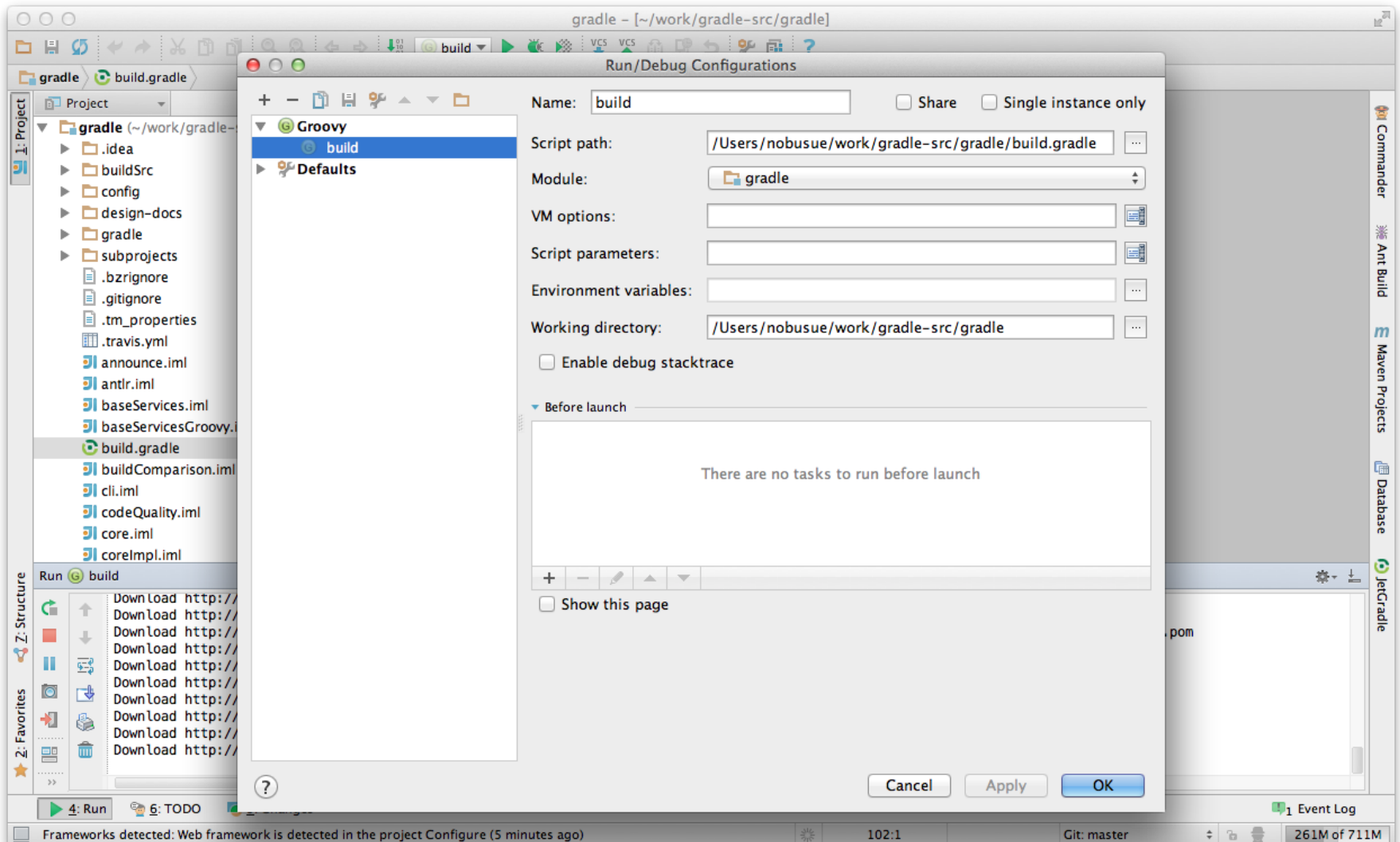
- ▶ IntelliJ IDEA
- ▶ Eclipse

# IntelliJ IDEA

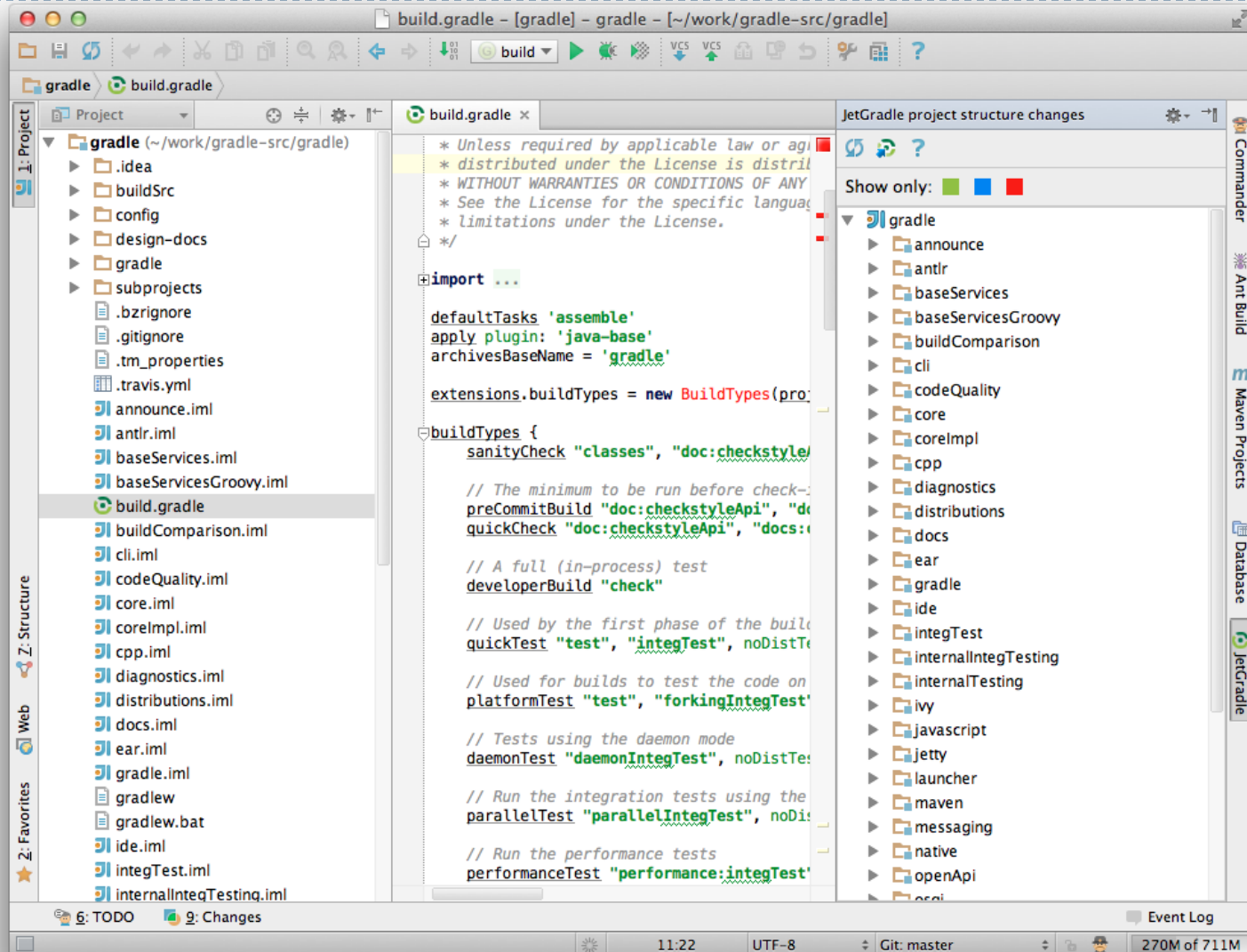
---

- ▶ Gradle -> IntelliJ
  - ▶ gradle IntelliJ pluginを利用
    - ▶ apply plugin: 'idea'
  - ▶ IDEA固有リソースを生成
- ▶ IntelliJ Gradle plugin(JetGradle)
  - ▶ Preferences -> Plugins -> Gradle
  - ▶ ウィンドウ右端の「JetGradle」を開いてみよう！

# IntelliJ IDEA: Gradle build confing



# IntelliJ IDEA: JetGradle





# Eclipse

---

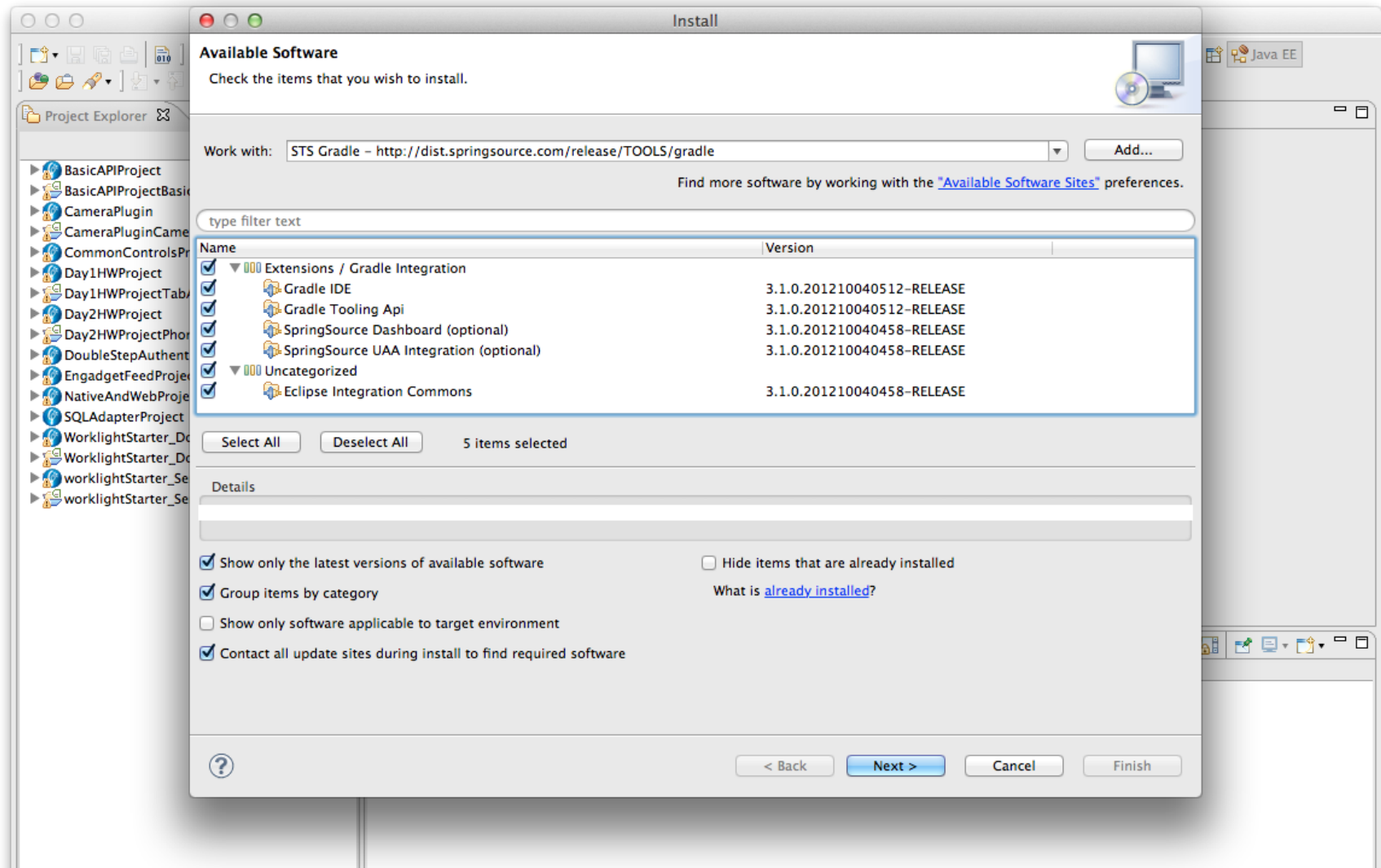
- ▶ Gradle -> Eclipse

- ▶ gradle Eclipse pluginを利用
  - ▶ apply plugin: 'eclipse'
- ▶ Eclipse固有リソースを生成
  - ▶ ただし、使いこなすにはいろいろ癖が...

- ▶ Eclipse Gradle plugin

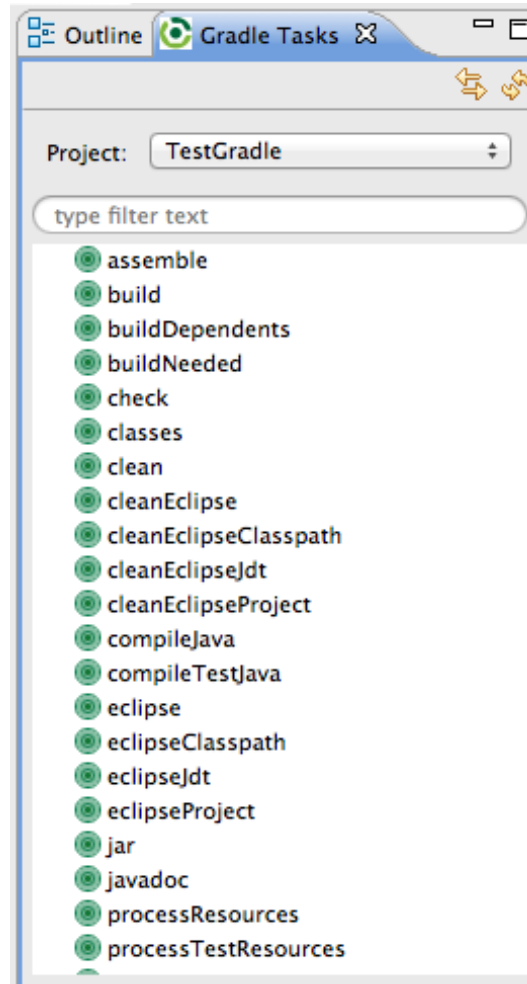
- ▶ STSの一部として提供
  - ▶ <http://dist.springsource.com/release/TOOLS/gradle>

# Eclipse: installing Gradle plugin



# Eclipse: Gradle Tasks view

---



# Jenkins integration

---

- ▶ Gradle wrapper
- ▶ Jenkins Gradle plugin

# Jenkinsとの統合 - Gradle

---

- ▶ Gradle wrapperを利用
  - ▶ JDKさえ導入されていれば、Gradleを自動でインストールして実行してくれる
  - ▶ Jenkins上では汎用コマンドとして実行すればよい
- ▶ Gradle wrapperは、CI用途に限らず、開発者の環境構築負荷軽減にも有用

# Gradle wrapper

[build.gradle]

```
task wrapper(type: Wrapper) {  
    gradleVersion = '1.4'  
}
```

Gradle導入済の環境で"gradle wrapper"を実行すると、Gradleのブートストラップ(gradlew)が生成される

```
build.gradle  
gradlew  
gradlew.bat  
  
└─ gradle  
    └─ wrapper  
        gradle-wrapper.jar  
        gradle-wrapper.properties
```



# Jenkinsのビルド設定

ジョブ名

☒ **フリー スタイル・プロジェクトのビルド**

もっとも汎用性の高いJenkinsの中核機能です。任意のSCMからソースコードをチェックアウトし、任意のビルドシステムでプロジェクトがビルドできます。往々にして、ソフトウェアのビルド以外にも様々な仕事の自動化に利用することができます。

☐ **Maven2/3プロジェクトのビルド**

Maven2/3のプロジェクトをビルドします。JenkinsはPOMファイルから必要な情報を読み取るので、設定が必要な項目はごくわずかです。

☐ **外部ジョブの監視**

Jenkinsの外(含む別のマシン上)で実行されるプロセスの実行をJenkinsに記録します。これにより、既存の自動化システムの動作をJenkinsを使って監視できます。[詳しくはこのドキュメント](#)をご覧ください。

☐ **マルチ構成プロジェクトのビルド**

複数の環境でのテストや、プラットフォームごとのビルドなどといった、多数の異なる構成が必要なプロジェクトに適しています。

## ビルド

### Windowsバッチコマンドの実行

コマンド

[ビルドから利用可能な環境変数の一覧](#)

削除

ビルド手順の追加 ▼

# Jenkinsのテストレポート設定

## ▶ JUnitと同様

ビルド後の処理

☒ JUnitテスト結果の集計 

テスト結果XML

[Antのファイルセットincludes属性の書式](#)に従ってビルドから生成されるXMLレポートファイル群を指定します(例: myproject/target/test-reports/\*.xml)。ワークスペースルートからの相対パスです。

☐ Retain long standard output/error 



# Jenkins Gradle plugin

---

- ▶ Jenkins Gradle pluginの機能
  - ▶ Gradleランタイムの管理
    - ▶ Gradleの自動インストール/バージョン指定
    - ▶ Gradle wrapperも指定可能
  - ▶ ビルド手順「Invoke Gradle script」の提供
  - ▶ コンソールに「実行されたGradleタスク」一覧のリンクを表示⇒ログの該当箇所にジャンプ

# Gradle plugin: install



アップデートセンター [Jenkins]

localhost:8081/pluginManager/installed

Jenkins プラグインマネージャー

ダッシュボードへ戻る  
Jenkinsの管理

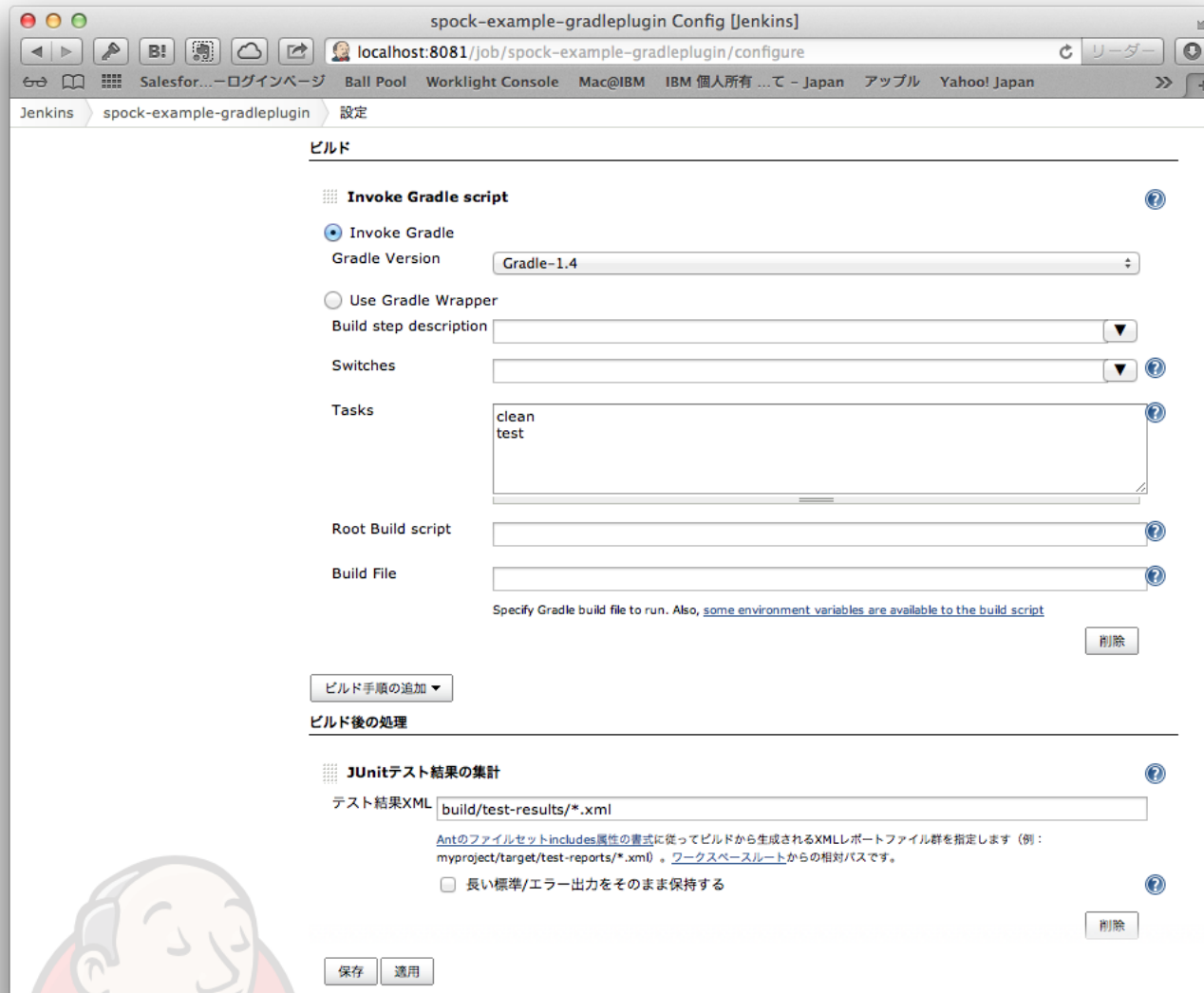
アップデート 利用可能 インストール済み 高度な設定

有効化	名前 ↓	バージョン	前回バージョン	ピン	アンインストール
<input checked="" type="checkbox"/>	<a href="#">Jenkins Mailer Plugin</a>	1.4			
<input checked="" type="checkbox"/>	<a href="#">External Monitor Job Type Plugin</a>	1.1			
<input checked="" type="checkbox"/>	<a href="#">LDAP Plugin</a>	1.2	1.1 にダウングレード	解除 ?	
<input checked="" type="checkbox"/>	<a href="#">pam-auth</a>	1.0			
<input checked="" type="checkbox"/>	<a href="#">Ant Plugin</a>	1.2	1.1 にダウングレード	解除 ?	
<input checked="" type="checkbox"/>	<a href="#">Javadoc Plugin</a>	1.1	1.0 にダウングレード	解除 ?	
<input checked="" type="checkbox"/>	<a href="#">Jenkins CVS Plug-in</a> Integrates Jenkins with CVS version control system using a modified version of the Netbeans cvsclient.	2.8	2.3 にダウングレード	解除 ?	
<input checked="" type="checkbox"/>	<a href="#">Jenkins GIT client plugin</a>	1.0.4			アンインストール
<input checked="" type="checkbox"/>	<a href="#">Jenkins GIT plugin</a> This plugin integrates <a href="#">GIT</a> with Jenkins.	1.3.0	1.1.18 にダウングレード		アンインストール
<input checked="" type="checkbox"/>	<a href="#">Jenkins Gradle plugin</a> This plugin allows Jenkins to invoke <a href="#">Gradle</a> build scripts directly.	1.21	1.16 にダウングレード		アンインストール
<input checked="" type="checkbox"/>	<a href="#">Maven Integration plugin</a>	1.505			
<input checked="" type="checkbox"/>	<a href="#">Jenkins SSH Slaves plugin</a>	0.22			
<input checked="" type="checkbox"/>	<a href="#">Jenkins Subversion Plug-in</a>	1.45	1.40 にダウングレード	解除 ?	
<input checked="" type="checkbox"/>	<a href="#">Jenkins Translation Assistance plugin</a>	1.10	1.9 にダウングレード	解除 ?	

Help us localize this page

更新: 2013/03/18 0:17:29 [REST API](#) [Jenkins ver. 1.505](#)

# Gradle plugin: プロジェクト設定



spock-example-gradleplugin Config [Jenkins]

localhost:8081/job/spock-example-gradleplugin/configure

Jenkins spock-example-gradleplugin 設定

### ビルド

**Invoke Gradle script**

☒ Invoke Gradle

Gradle Version:

☐ Use Gradle Wrapper

Build step description:

Switches:

Tasks:

Root Build script:

Build File:

Specify Gradle build file to run. Also, [some environment variables are available to the build script](#)

### ビルド後の処理

**JUnitテスト結果の集計**

テスト結果XML:

[Antのファイルセットincludes属性の書式](#)に従ってビルドから生成されるXMLレポートファイル群を指定します (例: myproject/target/test-reports/\*.xml)。ワークスペースルートからの相対パスです。

☐ 長い標準/エラー出力をそのまま保持する

# Gradle plugin: コンソール



The screenshot shows the Jenkins web interface for a build named 'spock-example-gradleplugin #5'. The console output displays the following information:

- Build Information:** Checkout: workspace / /Users/nobusue/.jenkins/jobs/spock-example-gradleplugin/workspace - hudson.remoting.LocalChannel@25708786. Using strategy: Default.
- Git Repository:** Fetching changes from 1 remote Git repository. Commencing build of Revision 1c26a489f166368e3d94a27ef6f51cfb66f34b16 (origin/master, origin/HEAD).
- Build Steps:** Checking out Revision 1c26a489f166368e3d94a27ef6f51cfb66f34b16 (origin/master, origin/HEAD). No change to record in branch origin/master. No change to record in branch origin/HEAD.
- Gradle Build:** [Gradle] - Launching build. Unpacking <http://services.gradle.org/distributions/gradle-1.4-bin.zip> to /Users/nobusue/.jenkins/tools/hudson.plugins.gradle.GradleInstallation/Gradle-1.4 on Jenkins [workspace].
- Gradle Tasks:** \$ /Users/nobusue/.jenkins/tools/hudson.plugins.gradle.GradleInstallation/Gradle-1.4/bin/gradle clean test. The tasks executed are: clean UP-TO-DATE, compileJava UP-TO-DATE, compileGroovy UP-TO-DATE, processResources UP-TO-DATE, classes UP-TO-DATE, compileTestJava UP-TO-DATE, compileTestGroovy, Download <http://oss.sonatype.org/content/repositories/snapshots/org/spockframework/spock-core/1.0-groovy-2.0-SNAPSHOT/spock-core-1.0-groovy-2.0-20130225.163903-18.pom>, Download <http://repo1.maven.org/maven2/org/codehaus/groovy/groovy-all/2.0.6/groovy-all-2.0.6.pom>, Download <http://oss.sonatype.org/content/repositories/snapshots/org/spockframework/spock-core/1.0-groovy-2.0-SNAPSHOT/spock-core-1.0-groovy-2.0-20130225.163903-18.jar>, Download <http://repo1.maven.org/maven2/org/codehaus/groovy/groovy-all/2.0.6/groovy-all-2.0.6.jar>, processTestResources, testClasses, test, Download <http://repo1.maven.org/maven2/com/h2database/h2/1.3.168/h2-1.3.168.pom>, Download <http://repo1.maven.org/maven2/com/h2database/h2/1.3.168/h2-1.3.168.jar>.
- Build Result:** BUILD SUCCESSFUL. Total time: 25.404 secs. Build step 'Invoke Gradle script' changed build result to SUCCESS.
- Finished:** SUCCESS.

The left sidebar shows the build history and console output. The bottom status bar indicates the build was updated on 2013/03/18 0:23:23, with a REST API link and Jenkins version 1.505.

# Gradleの便利プラグイン

---

## ▶ プロジェクトレポート

- ▶ [http://gradle.monochromeroad.com/docs/userguide/project\\_reports\\_plugin.html](http://gradle.monochromeroad.com/docs/userguide/project_reports_plugin.html)

## ▶ 通知

- ▶ [http://gradle.monochromeroad.com/docs/userguide/announce\\_plugin.html](http://gradle.monochromeroad.com/docs/userguide/announce_plugin.html)

## ▶ Build Dashboard

- ▶ [http://gradle.monochromeroad.com/docs/userguide/buildDashboard\\_plugin.html](http://gradle.monochromeroad.com/docs/userguide/buildDashboard_plugin.html)

## ▶ Release

- ▶ <https://github.com/townsfolk/gradle-release>

# Gradleプラグイン、順調に増殖中

- 26 Ivyxml Plugin
- 27 Javascript Library Plugin
- 28 JavaFx Plugin
- 29 JavaPropFile Plugin
- 30 JAXB Plugin
- 31 jDocbook Plugin
- 32 JDepend Plugin
- 33 Launch4J Plugin
- 34 MacAppBundle Plugin
- 35 PMD Plugin
- 36 Protocol Buffers Plugin
- 37 OneJar plugin
- 38 Release Plugin
- 39 SvnKit Plugin
- 40 Templates Plugin
- 41 Tomcat Plugin
- 42 Xslt Plugin
- 43 Workspace Plugin
- 44 PGP Plugin
- 45 org.linkedin plugins
  - 45.1.1 a. org.linkedin.userConfig
  - 45.1.2 b. org.linkedin.spec
  - 45.1.3 c. org.linkedin.repository
  - 45.1.4 d. org.linkedin.release
  - 45.1.5 e. org.linkedin.cmdline
- 46 JSLint Plugin
- 47 Ubuntu Packager Plugin
- 48 RPM Plugin
- 49 Writing Custom Plugins
  - 49.1 Example of a simple custom task within buildSrc
  - 49.2 Example of a simple custom plugin within buildSrc
  - 49.3 Building an external plugin (outside buildSrc)

[http://wiki.  
gradle.org  
/display/G  
RADLE/Plu  
gins](http://wiki.gradle.org/display/GRADLE/Plugins)

# Gradleの情報源

---

- ▶ Gradle 日本語ドキュメント(@literaliceさん)
  - ▶ <http://gradle.monochromeroad.com/docs/index.html>
- ▶ ビルドツールGradle スタートアップガイドの紹介
  - ▶ [http://www.ntts.co.jp/publish/column/tec/java\\_03/index.html](http://www.ntts.co.jp/publish/column/tec/java_03/index.html)
- ▶ GVM:the Groovy enVironment Manager
  - ▶ <http://gvmtool.net/>
  - ▶ groovy/grails/gradle/griffon/vert.xに対応
- ▶ Gradleプロジェクトのソースコード
  - ▶ <https://github.com/gradle/gradle>
  - ▶ Gradleのdownloadにあるsrc.zipはビルドスクリプトがないので、ソースコードリーディングにはこちらを参照するべし

# 文字エンコーディング指定

---

- ▶ やり方はいろいろありますが、ビルドスクリプト中で指定するならこれがエレガントな方法
  - ▶ <http://mrhaki.blogspot.jp/2012/06/gradle-goodness-set-java-compiler.html>
  - ▶ タスクごとに指定
    - ▶ `compileJava.options.encoding = 'UTF-8'`
  - ▶ タスクのタイプでまとめて指定
    - ▶ `tasks.withType(Compile){  
    options.encoding = 'UTF-8' }`



# おすすめ書籍

---

- ▶ “Gradle Effective Implementation Guide” Hubert Klein Ikkink (mrhaki)
- ▶ <http://mrhaki.blogspot.de/search/label/Gradle%3AGoodness> の人の本です



# ありがとうございました

---



powered by NekoFont

<http://nekofont.upat.jp/>