Gradle布教活動

はじめに

Gradleとは?

Ant/Mavenに次ぐJavaの次世代ビルドツール

Ant/Mavenのイイトコ取り

Androidの公式ビルドツールとして使われている

他にもLinkedInとかSpringプロジェクトとかでも使われている

Gradleのイイトコ１

Groovy DSLによる簡潔で柔軟な記述

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="MavenSample" default="run" basedir="." xmlns:ivy="antlib:org.apache.ivy.ant">
    <property name="src.dir" value="src/main/java" />
    <property name="test.dir" value="src/test/java" />
    <property name="lib.dir" value="lib"/>
    <property name="build.dir" value="ant"/>
    <property name="compiler" value="/path/to/javac" />

    <path id="lib.path.id">
        <fileset dir="${lib.dir}"/>
    </path>
```

問題

```
    <target name="resolve">
        <ivy:retrieve/>
    </target>
```

さぁbuild.xml書くぞ！

```
    <target name="run" depends="resolve">
        <mkdir dir="${build.dir}" />
        <javac srcdir="${src.dir}" destdir="${build.dir}" classpathref="lib.path.id" executable="${compiler}"
encoding="UTF-8"/>
        <javac srcdir="${test.dir}" destdir="${build.dir}" classpathref="lib.path.id" executable="${compiler}"
encoding="UTF-8"/>
    </target>

</project>
```

XML地獄！

Gradleのイイトコ１

GroovyDSLによる簡潔で柔軟な記述

解決

さぁbuild.gradle書くぞ、
と心の中で思ったならッ!
その時スデに行動は終わっているんだッ!

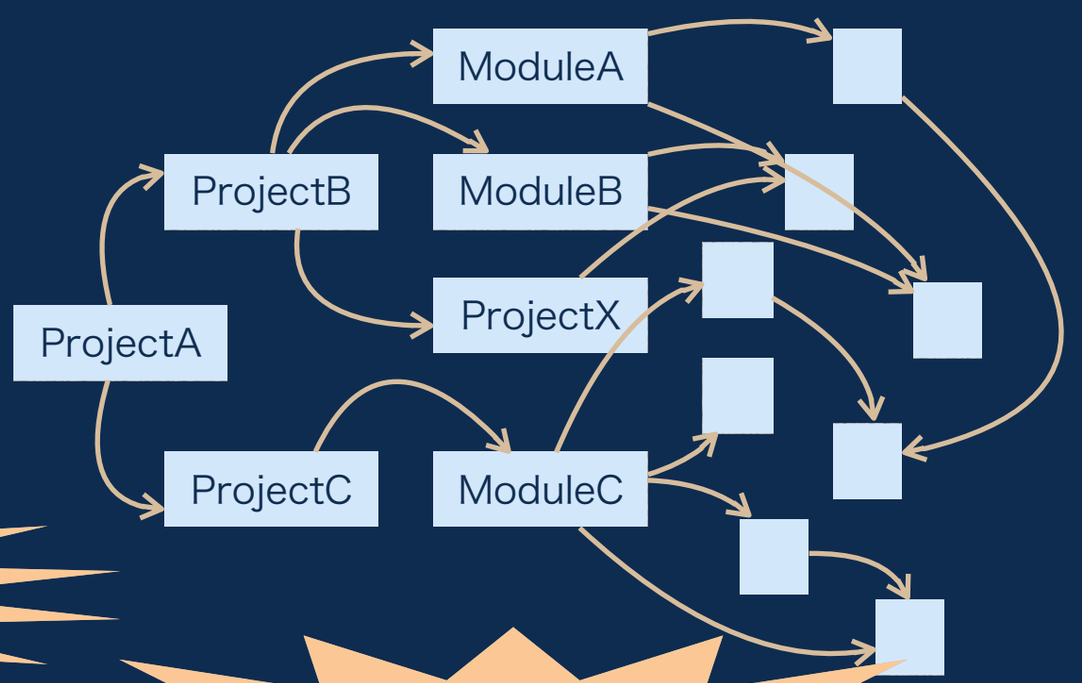
```
apply plugin: 'java'

repositories{
    mavenCentral()
}

dependencies{
    testCompile 'junit:junit:4.11'
}
```

Gradleのイイトコ2

Maven/Ivyによる依存関係解決

問題

ProjectAにパスを通して…と



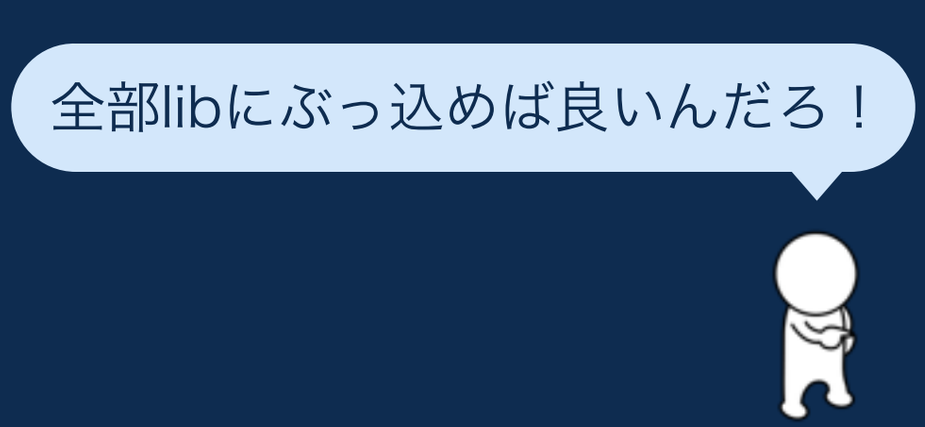
推移的な依存関係！

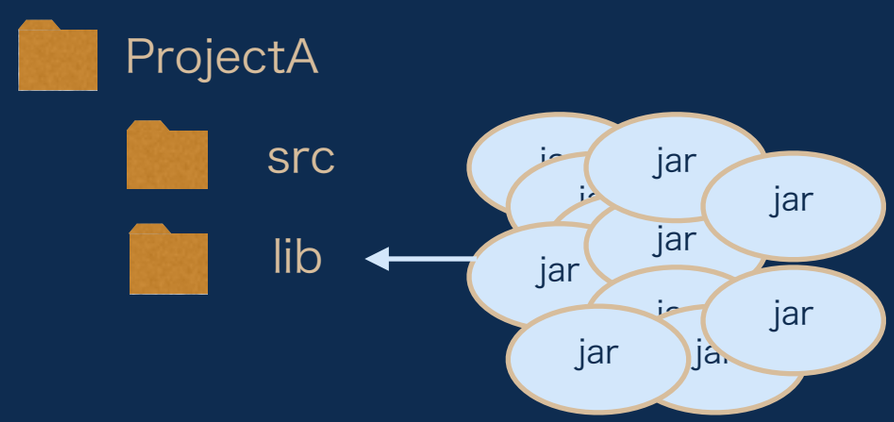
触手プレイ！

Gradleのイイトコ2

Maven/Ivyによる依存関係解決

問題





依存関係の隠蔽！

Gradleのイイトコ2

Maven/Ivyによる依存関係解決

解決

直接依存してるjarを指定すれば、あとはMavenリポジトリがヨロシクやってくれる

Mavenリポジトリ内のメタ情報で依存関係解決

Gradleのイイトコ3

設定より規約（COC）

問題

ソースはsrcに入れるべき

いや、src/main/javaだ

本質でない議論！

Gradleのイイトコ3

設定より規約（COC）

解決

Gradle(Maven)ではデフォルトで src/main/java

決まっているなら従おう

分かればよろしい

他にも様々な設定が存在する

Antだと全てを記述しなくてはいけない

COC = Corrosion Of Conformity


```
    *      *
  *      + うそです
    n ∧__∧ n
+ (∃(*´ ∀`)E)
    ∨      ∨    *
```

COC = Convension over Configuration

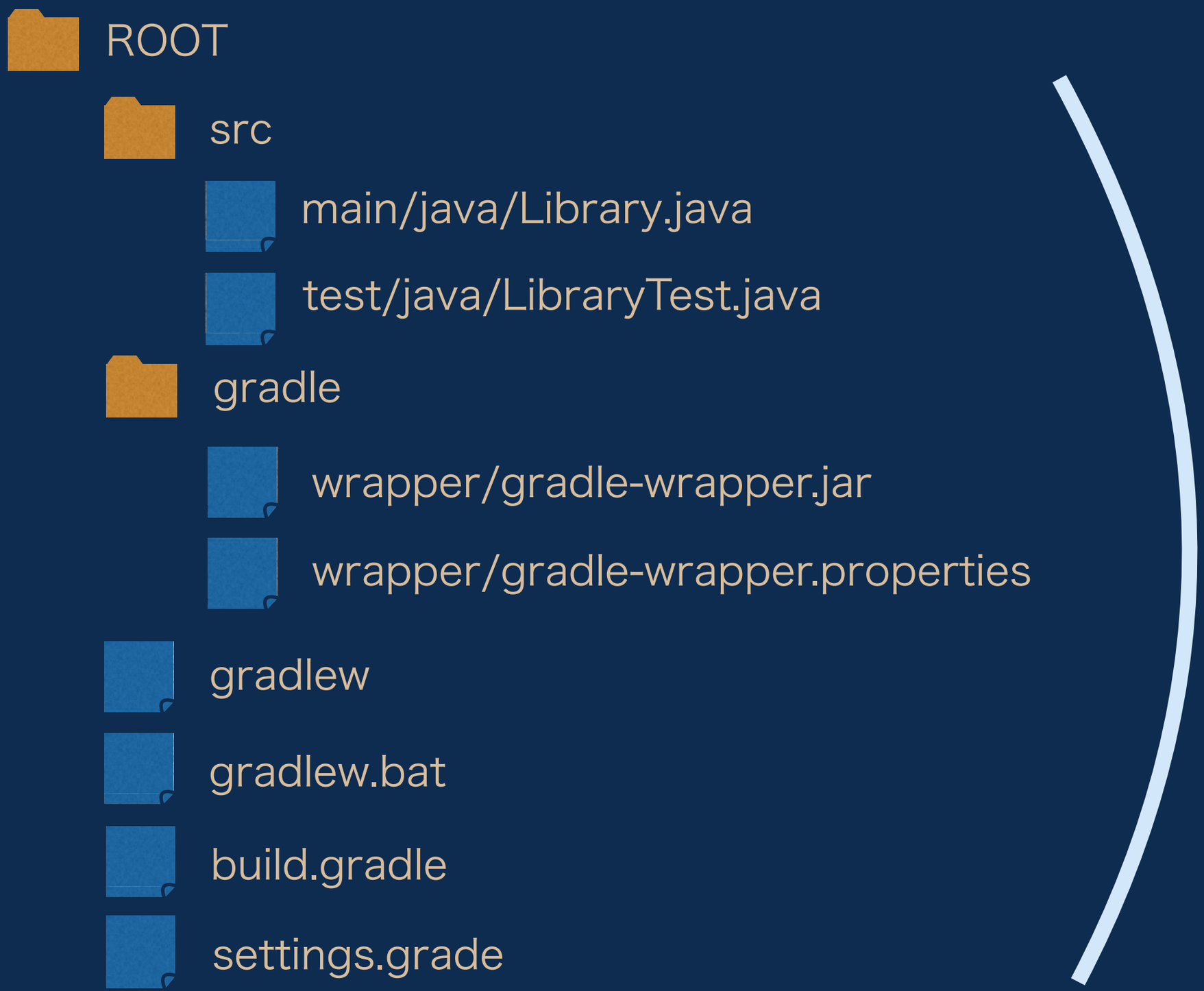
便利な機能

Build Init Plugin

サンプルプロジェクトをコマンド一発で作る

Gradleの機能を試したい時とかに便利

```
$ gradle init --type java-library
```



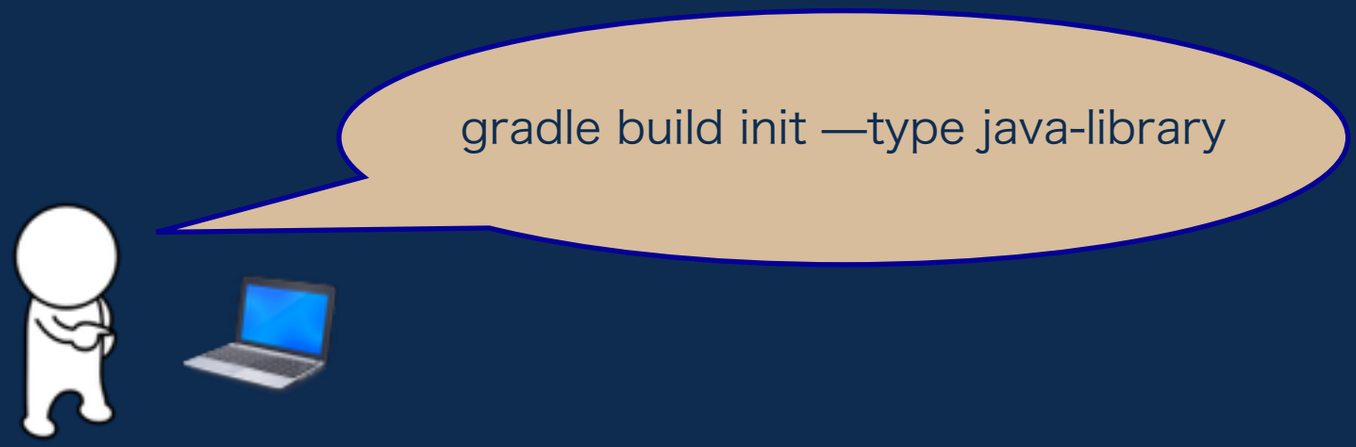
生成される

```
$ gradle init --type java-library
```

昔


新規クラス作って
新規プロジェクト作成して
フォルダ作って
新規ファイル作って
Eclipse立ち上げて
build.xml書いて

今


gradle build init —type java-library

Gradle Wrapper

Gradleをインストールしなくてもgradleタスクを実行できる

=> Gradle実行資産をダウンロードする

CI環境を作るときなどに便利

Gradleのバージョンアップもテキストを変えるだけ

$ gradle build

$ gradlew build

ROOT

gradle

wrapper/gradle-wrapper.jar

wrapper/gradle-wrapper.properties

gradlew

gradlew.bat

```
#Mon Jan 19 19:56:28 JST 2015
distributionBase=GRADLE_USER_HOME
distributionPath=wrapper/dists
zipStoreBase=GRADLE_USER_HOME
zipStorePath=wrapper/dists
distributionUrl
  =https://services.gradle.org/distributions/gradle-2.0-bin.zip
```

$ gradle wrapper

ROOT

build.grade

```
task wrapper(type: Wrapper) {
   gradleVersion = ‘2.2’
}
```

ROOT

gradle

wrapper/gradle-wrapper.jar

wrapper/gradle-wrapper.properties

gradlew

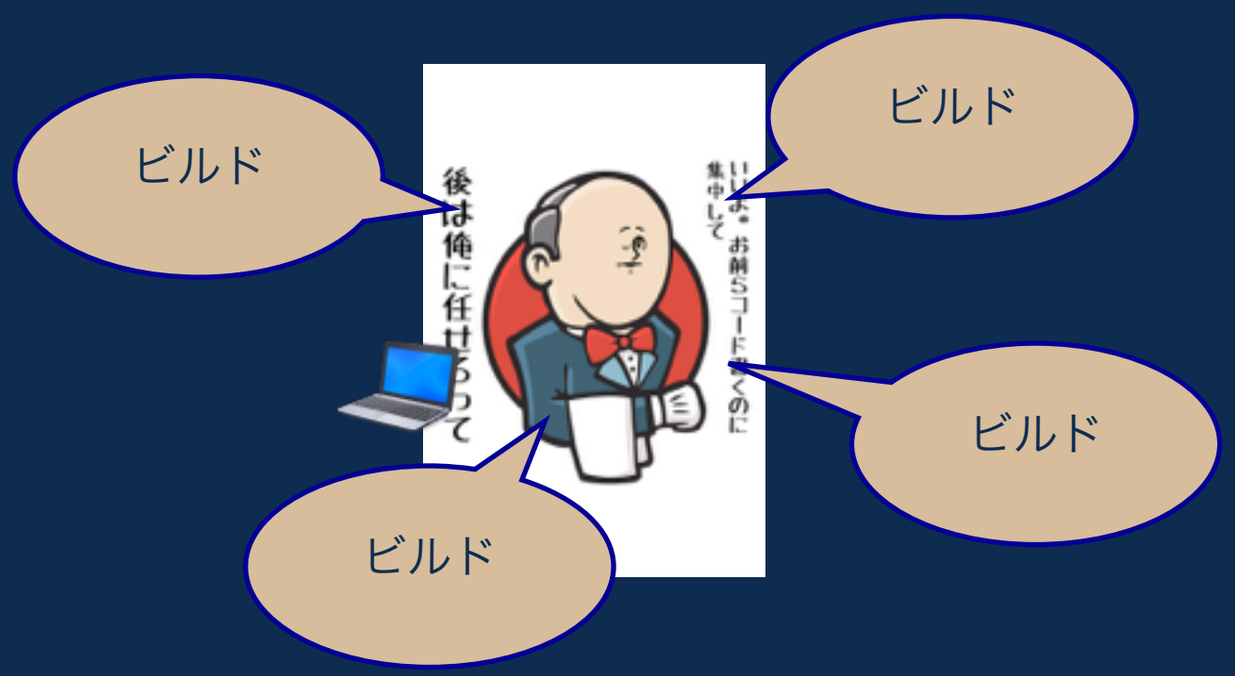
gradlew.bat

build.grade

昔


Ma
ven3.3インストール
ビルド
ビルド
Ant1.7イン
Ant1.8インストール
Ma
ven2.2インストール
ビルド
Gra
dle2.2インストール
ビルド

今


ビルド
ビルド
後は俺に任せろって
ビルド
ビルド

Ant/Maven取り込み

既存のスクリプトからの移行

build.xml(Ant)はそのまま使える

pom.xml(Maven)はbuild.gradleに変換する

Execute Ant Task

```
$ cat build.gradle
task helloAnt << {
  ant echo(message: “Hello Ant from Gradle!”)
}

$ gradle hA
:helloAnt
[ant:echo] Hello Ant from Gradle!

BUILD SUCCESSFUL
```

Import Ant Script

```
ant.importBuild ‘build.xml’
```

Convert pom.xml to build.gradle

```
$gradle init --type pom
```

昔

スムーズな移行

今

発展的な使い方

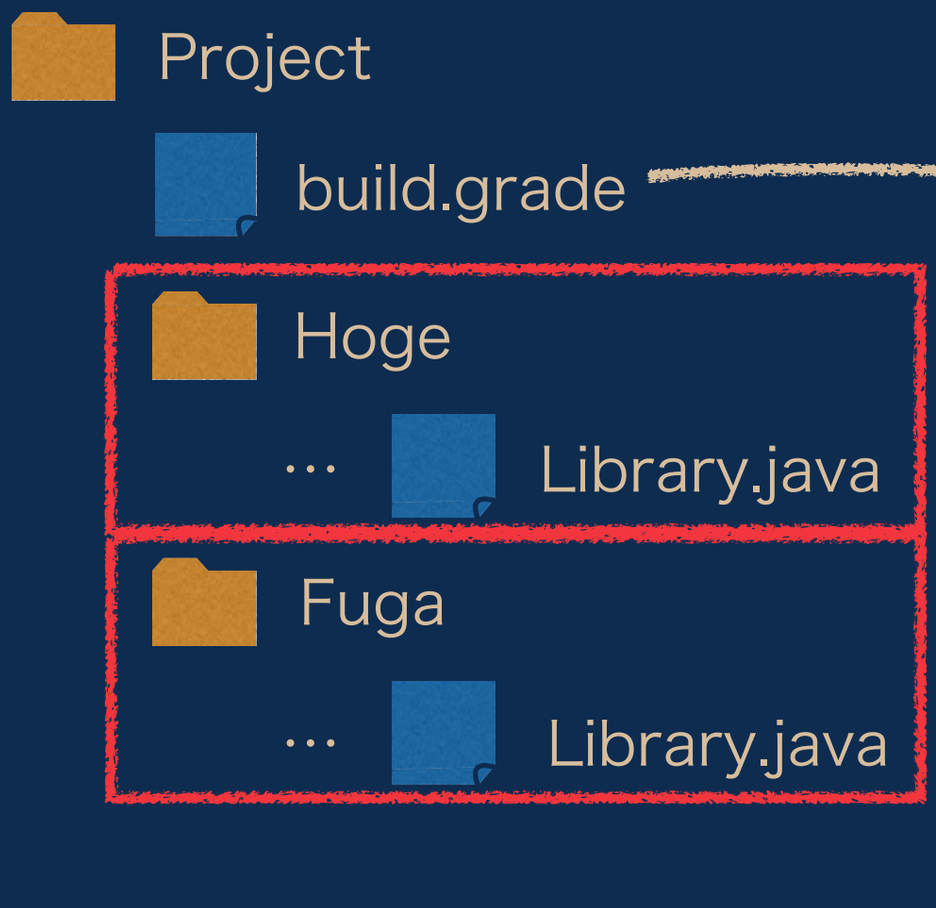
Build-compare Plugin

設定を変えてビルドを実行した際の差異を見る

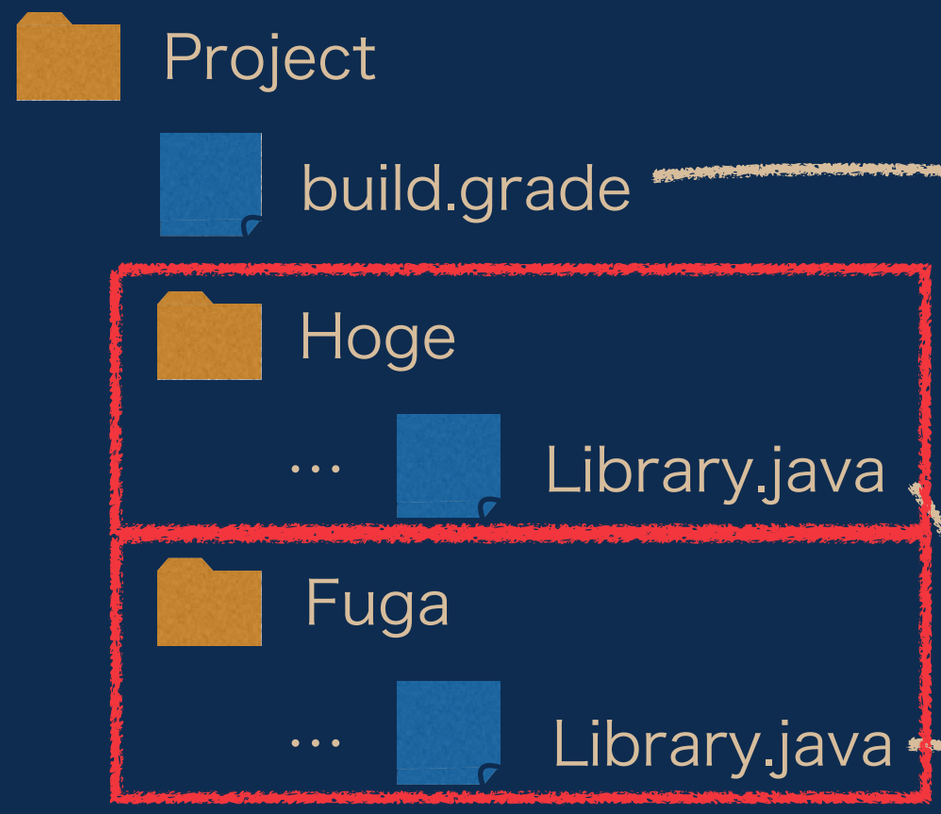
“gradle clean assemble” を実行した結果を比較

incubatingな機能

（= 〈計画・考えなどが〉生まれる，形をなす；具体化する.）




```
apply plugin: 'compare-gradle-builds'

compareGradleBuilds {
    sourceBuild {
        projectDir "Hoge"
    }
    targetBuild {
        projectDir "Fuga"
    }
}
```




```
apply plugin: 'compare-gradle-builds'

compareGradleBuilds {
    sourceBuild {
        projectDir "Hoge"
    }
    targetBuild {
        projectDir "Fuga"
    }
}
```

```
/*
 * This Java source file was auto generated
 *
 * @author kaakaa_hoe, @date 15/01/08 20:27
 */
public class Library {
    public boolean someLibraryMethod() {
        return true;
    }
}
```

```
/*
 * This Java source file was auto generated
 * @author kaakaa_hoe, @date 15/01/08 20:27
 */
public class Library {
    public boolean someLibraryMethod() {
        return true;
    }
    public String Hello(String name){
        return "Hello " + name + "!";
    }
}
```

$ gradlew compareGradleBuild

Compared builds

	Source Build	Target Build
Project	/Users/kaakaa_hoe/Documents/Learning/java/multi_project/compare-sample/Hoge	/Users/kaakaa_hoe/Documents/Learning/java/multi_project/compare-sample/Fuga
Gradle version	2.0	2.0
Tasks	clean assemble	clean assemble
Arguments		

Compared build outcomes

Compared build outcomes are outcomes that have been identified as being intended to be the same between the target and source build.

1. :jar

:jar

	Origi	
Source	build/libs/test1.jar	files/source/_jar/test1.jar
Target	build/libs/test1.jar	files/target/_jar/test1.jar

There are differences within the archive.

Entry Differences

Path	Difference
Library.class	entry in the source build is 330 bytes - in the target build it is 662 bytes (+332)

Path	Difference
Library.class	entry in the source build is 330 bytes - in the target build it is 662 bytes (+332)

Change Default Task

```
apply plugin: 'compare-gradle-builds'

compareGradleBuilds {
    sourceBuild {
        projectDir "Hoge"
        tasks = ["projects"]
    }
    targetBuild {
        projectDir "Fuga"
        tasks = ["init", "help"]
    }
}
```

Change Checkstyle Conf File

checkstyleの設定ファイル変えた時にどれだけ警告数が変わるか試して見ようとしたけど、現在(ver 2.2.1)ではZip形式のファイル比較しか出来ないっぽかった

http://www.gradle.org/docs/current/userguide/comparing_builds.html

=> 64.2.2. Supported build outcomes

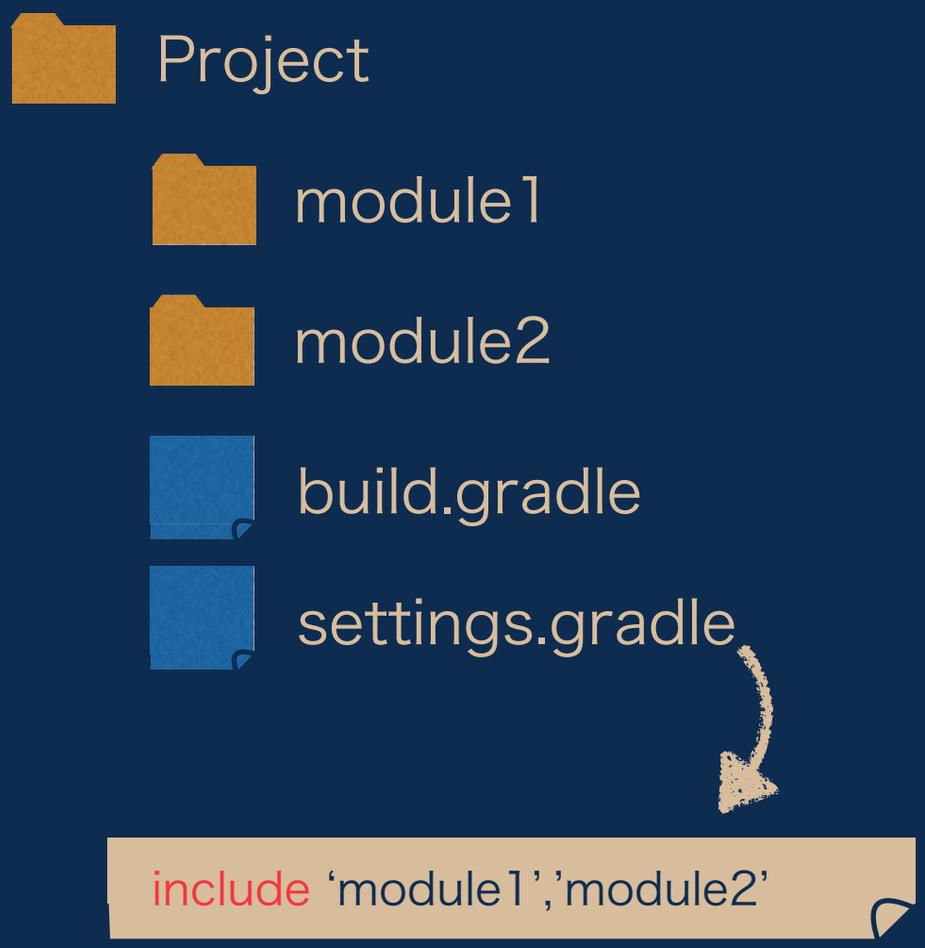
マルチプロジェクトビルド

複数のJavaモジュールから成るプロジェクト

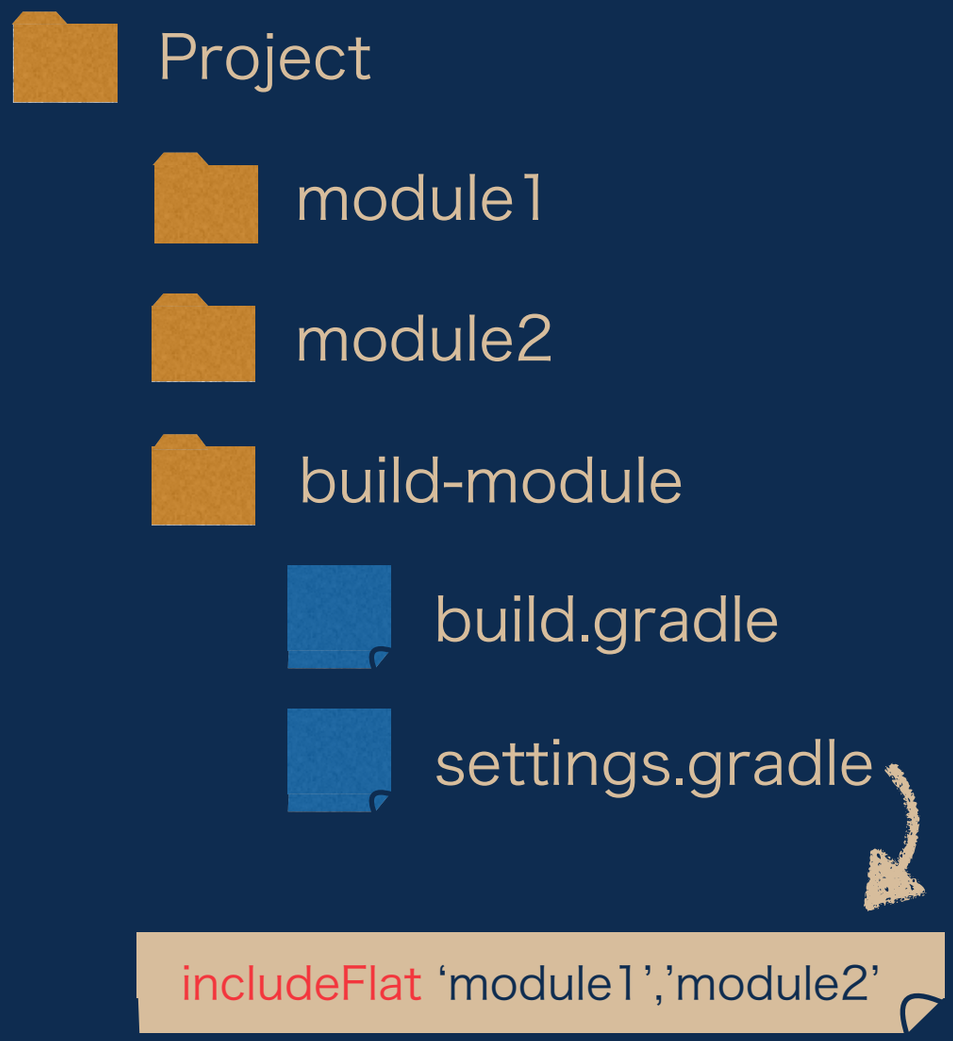
プロジェクト参照が多用されてる時に便利

どんなプロジェクト構成でも適用できる

Hierarchical layouts

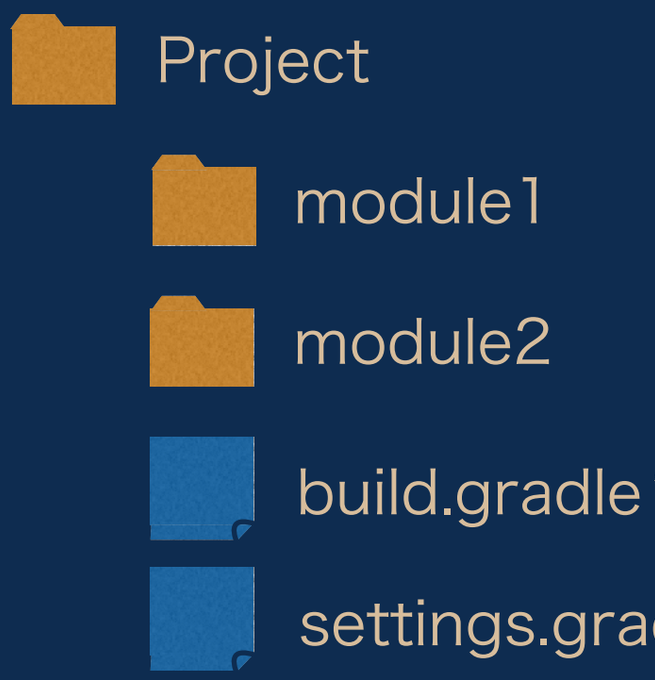


Flat layouts



ビルドスクリプト（build.gradle）はルートプロジェクト内に全てのモジュールのタスクを記述しても良いし、各モジュール毎にbuild.gradleを用意しても良い

Hierarchical layouts




```
include ‘module1’,’module2’
```

```
allprojects {
   …
}
subprojects {
   apply plugin: ‘java’
   …
}

project(‘:’) {
   …
}
project(‘:module1’) {
   …
}
```

---

allprojects … Project / module1 / module2

subprojects … module1 / module2

project(‘:’) … Project

project(‘:module1’) … module1

Hierarchical layouts

- Project
  - module1
  - module2
  - build.gradle
  - settings.grade

```
// collect child project names
def root = /./ as file
def filter = { it.isDirectory()
                 && !it.name.startsWith('.') }
               as FileFilter
def children = root.listFiles(filter)
                   .collect{it.name}.toArray()

// set children to multi project
include children
```

ビルドスクリプトにGroovyを直接書くことも出来るので、マルチプロジェクトの柔軟な設定も可能

おわりに

Gradle良いよね

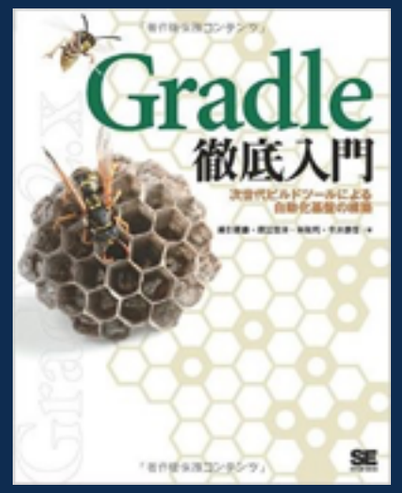
既存のビルドツールの機能を包括してる

Groovyが使えるので、とても柔軟

ビルドが定義でなくプログラミングに近くなる

情報源

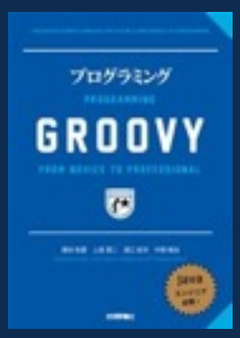
公式リファレンスだけでも十分



書籍なら「Gradle徹底入門」一択

英語読めるなら「Gradle In Action」でも◯

Groovyも知っておくと◎



Gradleはじめませんか