

明日から使いたくなる Gradle

2015/11/20 Gradle 勉強会

アジェンダ

1. Gradle とは？
2. Gradle の基本的なこと
3. Gradle で便利なものを作ろう
4. ここまでできる Gradle
5. 導入事例のご紹介
6. 時間が許すかぎり小ネタ
7. おわりに



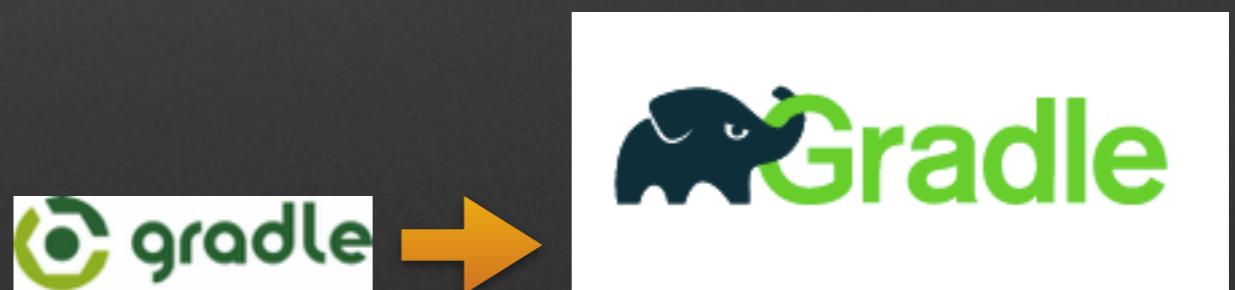
1. Gradle とは？

ぐれいどる

Gradle とは？

ぐれいどる

- 自動化ツール（ビルドツール）
- JVM（Java）上で動く
- Groovy（プログラミング言語）のスクリプト
- Ant や Maven の代わりに使うことができる



最近になって、ロゴが変わりました。

ビルドツールとは？

- Javaアプリケーション（およびその開発）において...
 - ソースコードをコンパイルする
 - コードの静的解析などの検証を行う
- Jar ファイルや War ファイルを作る
- 作った Jar/War をデプロイする

ビルドツールとは？

- 確かに、アプリケーション開発の成果物を生成するためのツールですが・・・
- 少し範囲を広げると、「作業をスクリプト化して、その作業を自動化できるようにするもの」と言えます

Groovy とは？

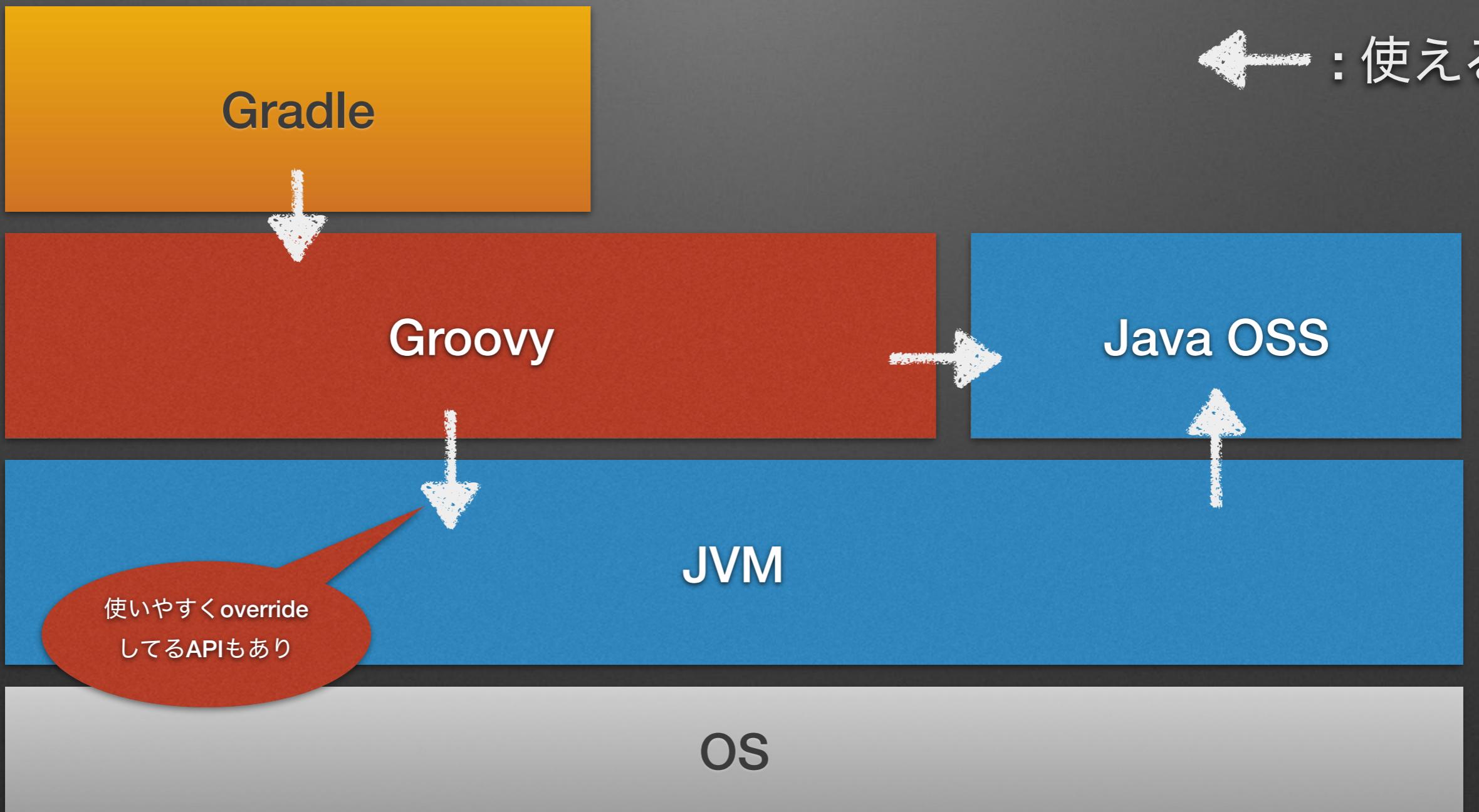
- JavaVM上で動くプログラミング言語（JVM言語）
- Javaとの互換性が高いので、とっつきやすい
- 植木さんが来年に Groovy の勉強会をやるらしいので
詳細はそちらで。Σ(ﾟ∀ﾟノノキヤー
- Gradleのスクリプトを書くということは
Groovy書いているということ。



Groovy わからないなら Java で

- Java ファイル (*.java) の拡張子を *.groovy に変えればそれはりっぱな Groovy です。
- 一部のレアケースを除いて、Java のコードはそのまま Groovy でも使用できます。
- Java で面倒なコーディングをらくにするためのライブラリだと思って大丈夫です。 (別のプログラミング言語と思う必要なし)

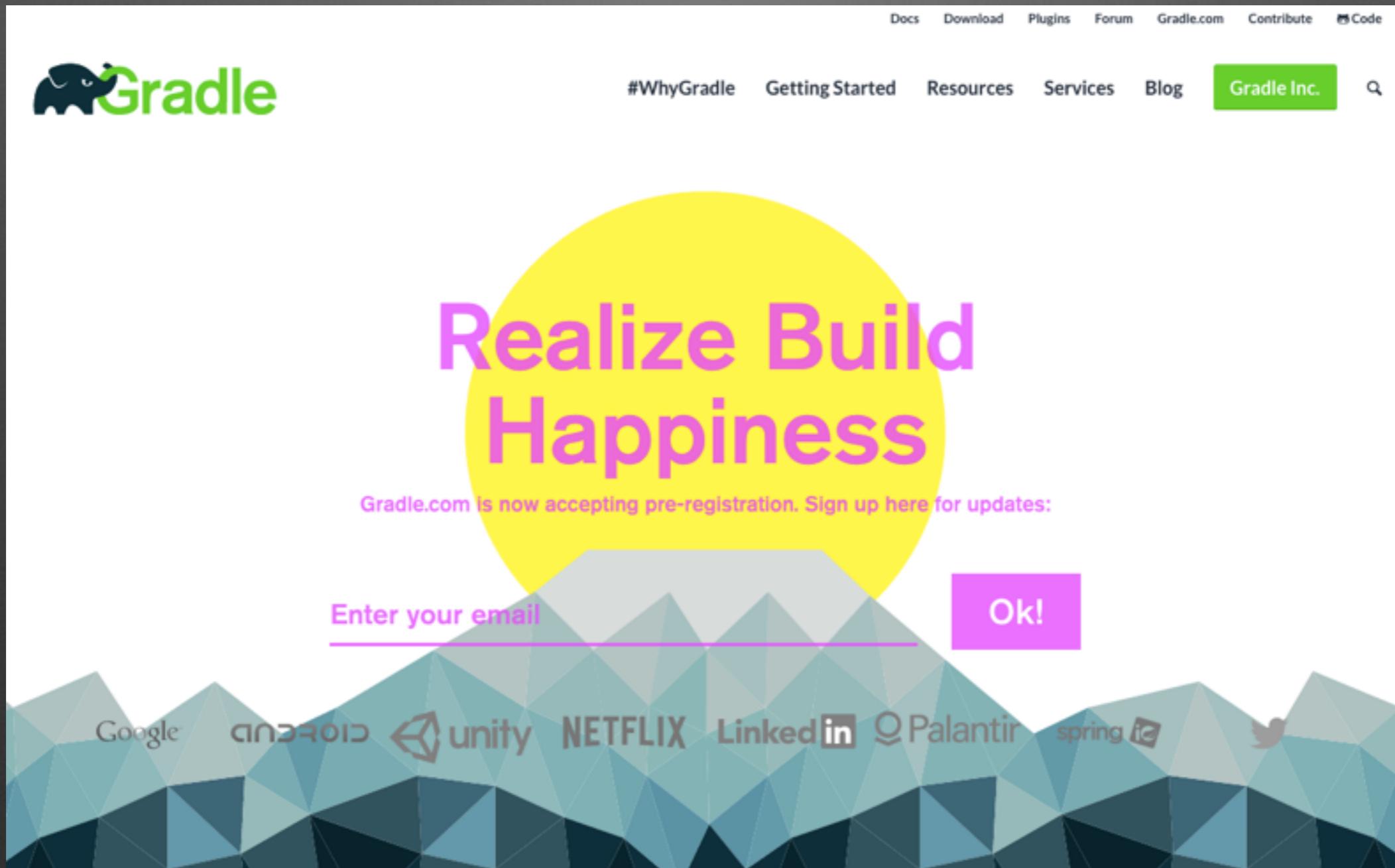
雑なイメージ



Gradle の説明にもどります

ぐれいどる

最近、見た目が変わりました...



<http://gradle.org/>

本家サイトから pickup

- Realize Build Happiness
(ハッピーなビルドを実現する)
- Gradle makes the impossible possible,
the possible easy, the easy elegant.
(不可能を可能に、可能を簡単に、簡単を上品に)

Why Gradle ?

 Gradle

#WhyGradle

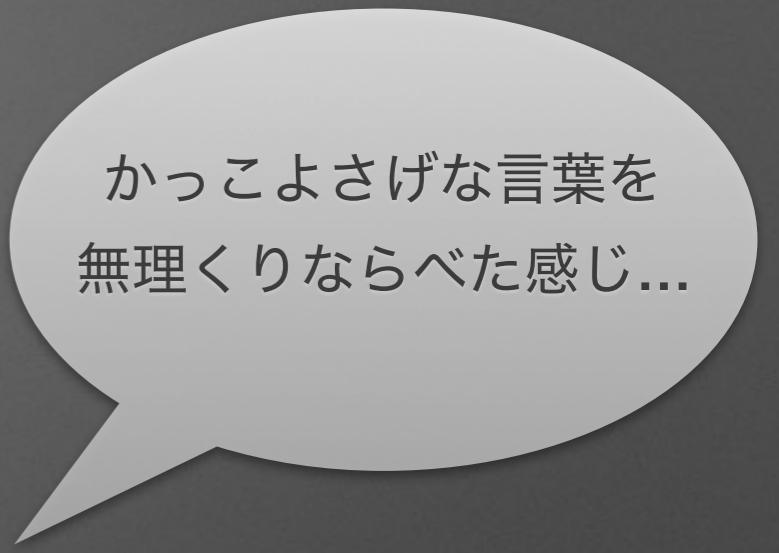
Gradle makes the impossible possible, the possible easy and the easy elegant.

 Polyglot Builds	 Tool Integrations	 Robust Dependency Management
LinkedIn uses Gradle to build 60 different programming languages including Java, Scala, Python, C/C++, Android, iOS and many many more. Find out why Gradle is the polyglot build tool.	Plugins and integrations with every imaginable tool in the DevOps Automation pipeline from IDEs like Eclipse, Android Studio and IntelliJ to Jenkins through to Chef, Puppet, Docker, and Ansible.	Out of the box, Gradle handles transitive dependencies that resolve across multiple repository types including Maven, Ivy, flat files.
 Powerful Yet Concise Logic	 High Performance Builds	 Build Reporting
The perfect blend of declarative and imperative. Gradle provides ultimate control and scriptability of Build Automation & Continuous Deployment pipelines. A concise maintainable build is why Gradle is becoming an Enterprise standard for build automation.	Incremental builds, build caching and parallelization drives the Gradle daemon to new heights of performance. Build only what has changed and see how performance is one of the reasons why Gradle is so broadly adopted.	Robust build analytic capabilities that allow build masters to see exactly what needs to be optimized and zero in on build problems. Gain insight into the efficiency of build automation including which modules are outperforming and which are lagging.

<http://gradle.org/whygradle-build-automation/>

Why Gradle ?

- Polygot Build
- Tool Integrations
- Robust Dependency Management
- Powerful Yet Concise Logic
- High Performance Builds
- Build Reporting



かっこよさげな言葉を
無理くりならべた感じ...

<http://gradle.org/whygradle-build-automation/>

今日のゴール

ねらい

- Gradle を使って、なにかを自動化してみたくなる

目的

- Gradle をすぐに実行できる環境、取っ掛かりを手に入れる
- ちょっとしたものであれば、その作業をコード化できるようになる

今日やること

- 簡単な Gradle タスクをゼロから作ってみる
- Gradle のサンプルをたくさん見てみる
- Gradle を色々と動かしてみる
- Gradle でできることをなんとなく理解する

今日やらないこと

- Java ソースコードのビルド／テスト／デプロイを自動化することについて深掘りする
- Gradle と Jenkins (CI) との連携をやってみる
- 過度な Maven 批判



2. Gradle の基本的なこと

Gradle をインストールする

- Java をインストールする
 - 環境変数に JAVA_HOME を追加する。
 - %JAVA_HOME%\bin にパスを通す。
 - \$ java -version
- Gradle をインストールする
 - 環境変数に GRADLE_HOME を追加する。
 - %GRADLE_HOME%\bin にパスを通す。 ← おそらく必須じゃない
 - \$ gradle -v

Gradle がインストールされたか確認する

- コマンドプロンプトを開いて、以下のコマンドを実行する

```
$ gradle -v
```

```
-----  
Gradle 2.7  
-----
```

← バージョン情報が表示される

```
Build time: 2015-09-14 07:26:16 UTC
```

```
Build number: none
```

```
Revision: c41505168da69fb0650f4e31c9e01b50ffc97893
```

```
Groovy: 2.3.10
```

```
Ant: Apache Ant(TM) version 1.9.3 compiled on December 23 2013
```

```
JVM: 1.8.0_05 (Oracle Corporation 25.5-b02) ← Gradle が参照している Java のバージョン
```

```
OS: Mac OS X 10.10.5 x86_64
```

プロキシ設定をする

ビルドツールでトラブルのは、だいたいプロキシがらみ！

1. [USER_HOME]¥.gradle ディレクトリに
gradle.properties ファイルを作成する (UTF-8で)
2. gradle.properties ファイルにプロキシ設定を書く

gradle.properties ファイル

```
systemProp.http.proxyHost=myproxy.co.jp  
systemProp.http.proxyPort=8080  
systemProp.http.proxyUser=*****  
systemProp.http.proxyPassword=*****
```

```
systemProp.https.proxyHost=myproxy.co.jp  
systemProp.https.proxyPort=8080  
systemProp.https.proxyUser=*****  
systemProp.https.proxyPassword=*****
```

- 環境によっては proxyUser, proxyPassword が不要な場合も

日本語ドキュメントの存在

- ・新しいものを導入するにあたって、日本語のドキュメントがあるかどうかは、とてもとても重要なこと
→ 安心してください・・・売ってますよ。
3つ紹介します。

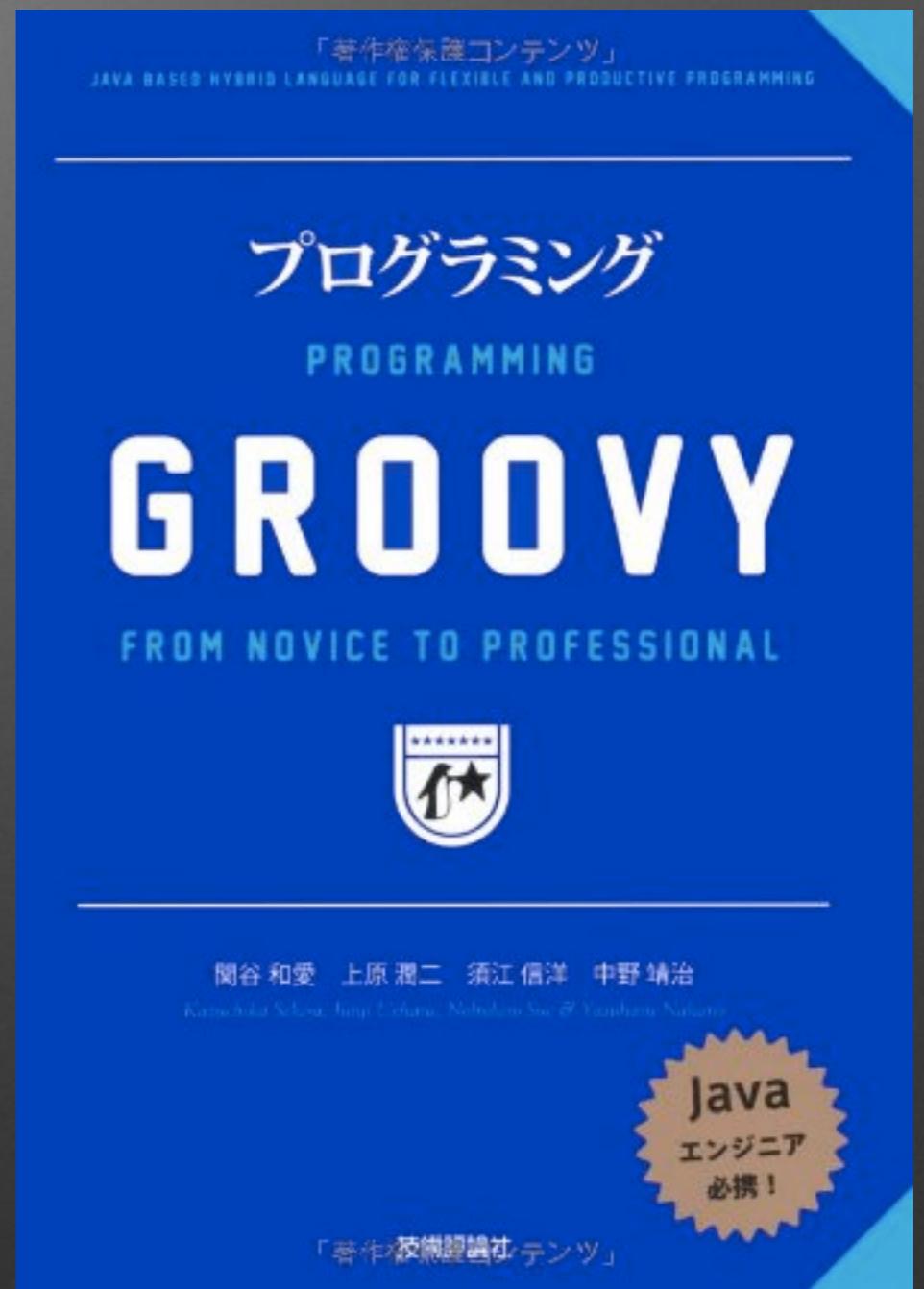
①『Gradle 徹底入門』

- 今から始めるなら、絶対これ買ってください。
- 2014年11月に出ました。
- これが出来たことで Gradle の敷居がぐっと下がりました。



②『プログラミング Groovy』

- Gradle は Groovy のスクリプトなので、必携です。
- Groovy ができることがたくさん紹介されています。
「Javaでやってたことがこんなに簡単にできるのか？！」
- 「逆引き○○」みたいにリファレンスとして末永く使えます。



③ Gradle日本語ドキュメント

- <http://gradle.monochromeroad.com/docs/>
- 『Gradle徹底入門』が出るまではこれがすべてだった。
手元に本を持ちあわせていないときにご覧ください。



他のビルドツールとの比較

→新しい

	Ant	Maven	Gradle
定義方法	XML	XML	スクリプト
依存関係の解決	できない	できる	できる
Eclipse との親和性	◎ 組み込み済み	○ プラグイン充実	△ プラグイン微妙
お隣との お付き合い	・・・	Ant 呼べる	Ant Maven 呼べる

- ・後発の優位性 → 既存ツールのいいところを取り込んでる
- ・定義方法が「スクリプト」というのがキモです

スクリプトベースのいいところ

- 複雑なところはコーディングすればいい。
正規の API を使えるとかっこいいけど、わからなければ
ガリガリ書いても大丈夫。自由です。
(Ant や Maven は定石や型を知らないと先に進めない)
- Gradle は Groovy、Groovy は Java の拡張。
Java で実装できるなら Java でコーディングしてOK。
Java のノウハウを移植可能。

ファイルコピー① Gradle っぽく

```
task copyByGradle(type: Copy) {  
    from "org/original1.txt"      ← コピー元のファイル  
    into "dest"                  ← コピー先のディレクトリ  
}
```

- ・ デフォルトタスクの Copy タスクを使用する
- ・ Gradle の文法を知っていると、複雑な制御も可能
- ・ ただし、ある程度の知識がないとつらい

ファイルコピー② Groovy で華麗に

```
task copyByGroovy << {
    new File("dest/original2.txt") << new File("org/original2.txt").readBytes()
}
```

シフト演算子で書き込み↑ ↑ファイルをバイトストリームにして

- Gradle 知識がなくても、ある程度簡略的に書ける。
- 条件分岐や繰り返しなどを駆使して、複雑なことをする場合は Groovy 主体で書いた方がうまくいくことも。

ファイルコピー③ Java でやむなく

```
buildscript {  
    repositories {  
        mavenCentral()  
    }  
    dependencies {  
        classpath 'commons-io:commons-io:2.4'      ← commons-io を依存関係に追加  
    }  
}  
  
import org.apache.commons.io.FileUtils           ← import 文  
  
task copyByJava << {  
    FileUtils.copyFile(new File("org/original3.txt"), new File("dest/original3.txt"))  
}          ↑ FileUtils#copyFile(コピー先, コピー元)
```

- ・ とはいえ、豊富な＆慣れている Java ノウハウを活用できる
ヘタに Gradle/Groovy 縛りでやるよりも効果的な場合もあり

結論

- ・文法やスマートなやりかたを知っていなくても
自分の知っているやり方でコーディングすればOK
- ・最低限覚えておかないといけないことは少ないので
安心してください。
- ・Groovy／Gradle に慣れないうちは Java

それでは、手を動かしましょう

build.gradle ファイル

- 任意の場所に build.gradle という名前のファイルを作ってください。
- Maven でいう pom.xml
Ant でいう build.xml
- このファイルにすべてを書いていきます。
(もちろんファイル分割もできます)

タスクを作る

- build.gradle ファイルに以下の3行を書いてください。

```
task firstTask << {
    println "Hello, Gradle."
}
```

※ `println "Hello"` は `System.out.println("Hello")` と同じ

タスクを実行する

- `build.gradle` が存在するフォルダでコマンドプロンプトを開いて、以下のコマンドを実行します。

```
$ gradle firstTask
```

```
:firstTask           ← firstTask タスクを実行開始する合図  
Hello, Gradle.      ← タスク内で標準出力した
```

- `$ gradle <タスク名>`
- `$ gradle <タスク名1> <タスク名2> ...`
という感じに複数のタスクを連続して実行できる

leftShift

「 << 」 ← これ重要

```
task print1 << {  
    println "test 1"  
}
```



```
task print2 {  
    println "test 2"  
}
```



- ・ この2つのタスクは似て非なるもの。
- ・ このケースでは、後者の書き方は間違いになる。

タスク実行してみると…

```
task print1 << {
    println "test 1"
}

task print2 {
    println "test 2"
}
```



```
$ gradle print1
test 2
:print1
test 1
```

```
$ gradle print2
test 2
:print2
```

- print2 タスクのなかで書いた「`println "test 2"`」がタスク実行とは関係なく実行されている

「<<」 有無の判断基準

- ・ <<あり：タスクで実行したい処理を実装する
- ・ <<なし：タスク（組み込みタスク）の定義やパラメータをセットする

```
task copyByGradle(type: Copy) {  
    from "org/original1.txt"      ← コピー元のファイルを引数っぽく指定  
    into "dest"                  ← コピー先のディレクトリを引数っぽく指定  
}
```

「<<」がないパターン

「<<」とのつきあいかた

- 「シフト演算子」 (Groovy では実は超便利アイテム)
- 慣れないうちはガリガリの有無で判断 (少し乱暴)
 - 自分でガリガリ書くときは「<<」あり
 - Copy や Zip などのすでにあるタスクに動的なパラメータを渡すときは「<<」なし

組み込みタスクの例

- Copy
- Zip
- 後述のプラグインを使うと、使えるタスクが増える
 - Javaプラグイン：jar, compileJava, clean
 - Warプラグイン：war

ライブラリを利用してみる

- buildscript ブロック
- repositories ブロック
- dependencies ブロック

複数のタスクに依存関係をもたせる

- `dependsOn`
- 組み込みタスクの中には、すでにタスク間の依存関係が設定されているものがある

gradle init

- Gradle を書き始める際の「ひな型」を作ってくれる。
- 任意の場所で以下のコマンドを実行してください。

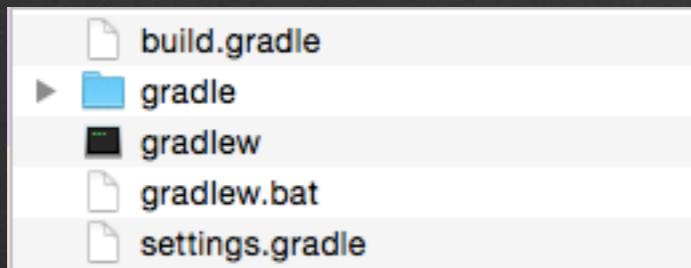
```
$ gradle init
```

```
:wrapper
```

```
:init
```

→wrapper タスクと init タスクが実行されている

```
BUILD SUCCESSFUL
```



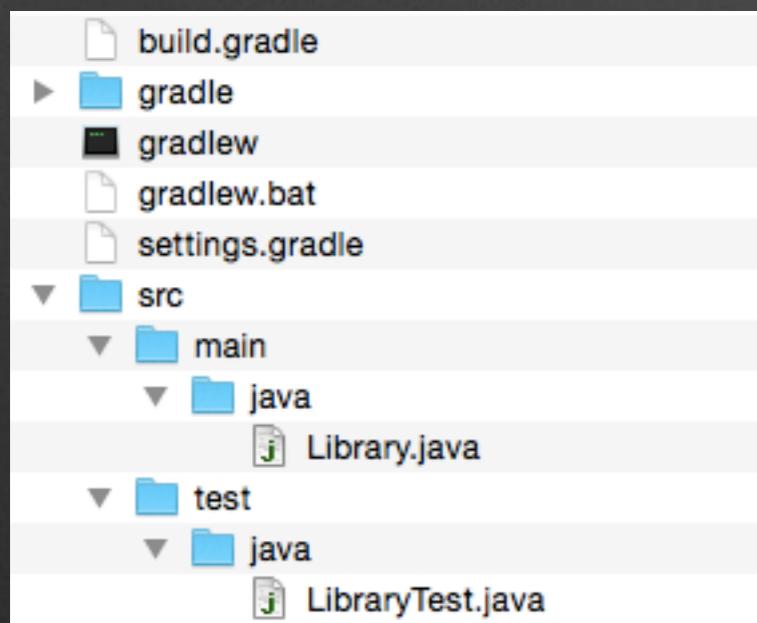
実用的なひな型もあります

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

↑オプションでひな型のパターンを指定する

```
:wrapper  
:init
```

BUILD SUCCESSFUL



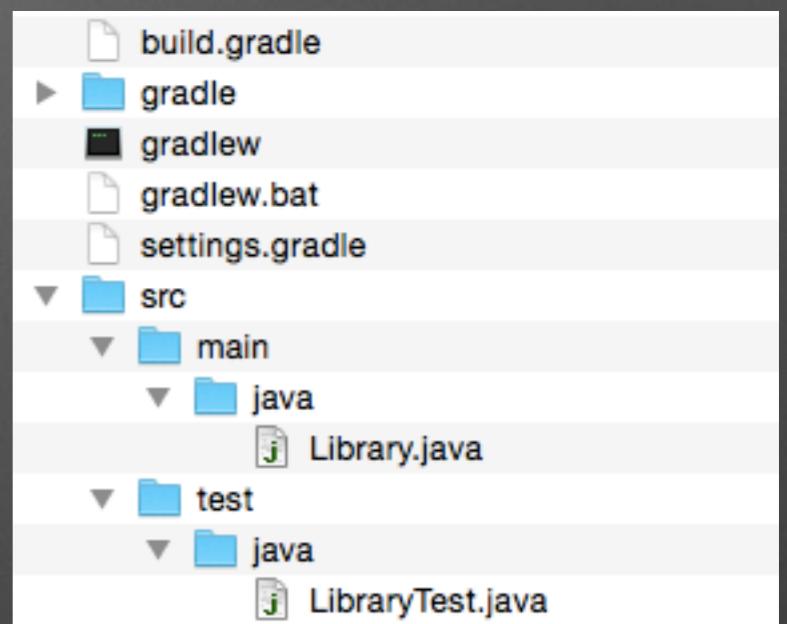
Java プロジェクトのひな型ができあがります。

他にも多数のひな型がありますといいたいところですが、
そんなにありません。

自分でゼロから作った方がいいかも。勉強になるし。

せっかくなので、Java のビルドやってみる

- さきほどの Java ひな型を使います
 - Java プロダクトコード (`Library.java`)
 - Java テストコード (`LibraryTest.java`)
- Java のコンパイル、テストコードの実行、jar ファイルを作成する...といったことができそうな予感



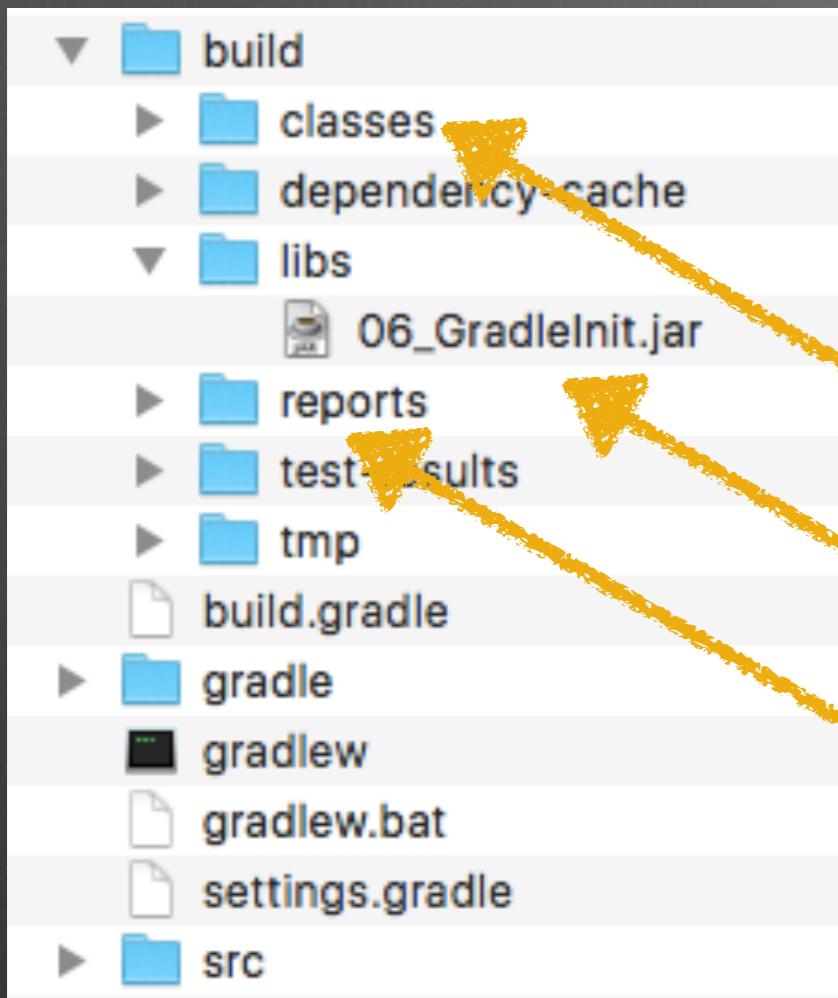
Java をビルドする

```
$ gradle build
```

```
:compileJava  
:processResources           ← この辺で Java コードをコンパイルしている  
:classes  
:jar                         ← Jar ファイルを作っている  
:assemble  
:compileTestJava  
:processTestResources  
:testClasses  
:test                          ← テストコードでテストしている  
:check                         ← ソースコードやここまで成果物の検証をしている  
:build
```

BUILD SUCCESSFUL

ビルド成果物



- Java プラグインは build ディレクトリ以下に成果物を作る
 - コンパイル結果のクラスファイル
 - jar ファイル
 - JUnit テスト結果のレポート

Java を使うための build.grade

```
apply plugin: 'java'      ← point① Java プラグインの使用  
  
repositories {  
    jcenter()  
}  
  
dependencies {  
    compile 'org.slf4j:slf4j-api:1.7.12'  
  
    testCompile 'junit:junit:4.12'  
}
```

- 難しい話になるので、ちょっとだけ解説します

① Java プラグインの追加

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

- `apply plugin: '<Plugin ID>'` で、プラグインを追加できます。複数追加もちろん可。
- その他のプラグイン（たくさんあります）
 - ‘war’ : Java EE をビルドする。War ファイルを作る
 - ‘checkstyle’ : Java コード解析 CheckStyle を実行する
 - ‘eclipse’ : Eclipse プロジェクト化させる

① Java プラグインの追加

- ・ プラグインを追加すると、実行できるタスクが増える

```
$ gradle tasks
```

← 実行可能なタスクの一覧を確認できる

- ・ Java プラグインの場合、以下のようなタスクが増える

compileJava	Java プロダクトコードをコンパイルする
test	Java テストコードで JUnit テストを実行する
jar	Java プロダクトコードから jar ファイルを作る
build	jar ファイルを作る、テストを実行する

タスクは他のタスクに依存する場合も

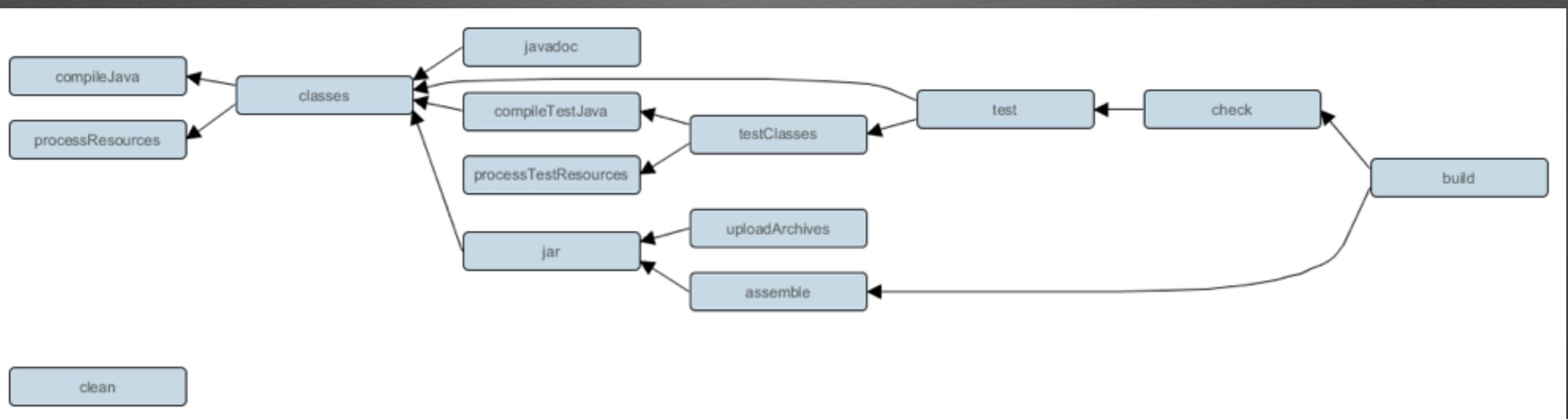
```
$ gradle build
```

```
:compileJava  
:processResources  
:classes  
:jar  
:assemble  
:compileTestJava  
:processTestResources  
:testClasses  
:test  
:check  
:build
```

←**build** タスクだけが実行されているわけではない

build タスクが依存するタスクも
いもづる式に実行される

Java プラグインタスク関係図



http://gradle.monochromeroad.com/docs/userguide/java_plugin.html

② リポジトリの指定

- 以下のどちらかを指定してもらったらいいです。

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

← jar ファイルを <http://jcenter.bintray.com>
から取ってくる

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

← jar ファイルを <http://repo1.maven.org/maven2>
から取ってくる

③ 依存関係の追加

```
dependencies {  
    compile 'org.slf4j:slf4j-api:1.7.12'  
  
    testCompile 'junit:junit:4.12'  
}
```

- `dependencies` ブロックに Java プログラムが参照する jar ファイル (とそのバージョン) を指定する

依存関係の書き方

```
dependencies {
```

```
    compile 'org.slf4j:slf4j-api:1.7.12'
```

コンフィグレーション
}

グループID

アーティファクトID

バージョン

↑は短縮形の書き方

短縮しない書き方↓

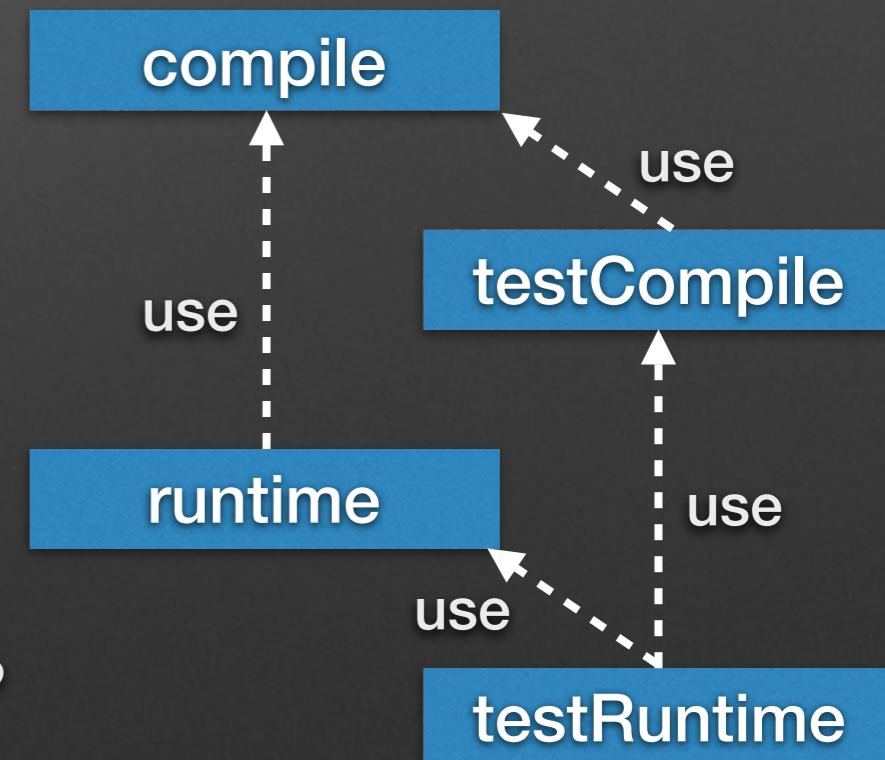
```
dependencies {  
    compile group: 'org.slf4j', name: 'slf4j-api', version: '1.7.12'  
}
```

コンフィグレーション？

`compile 'org.slf4j:slf4j-api:1.7.12'`

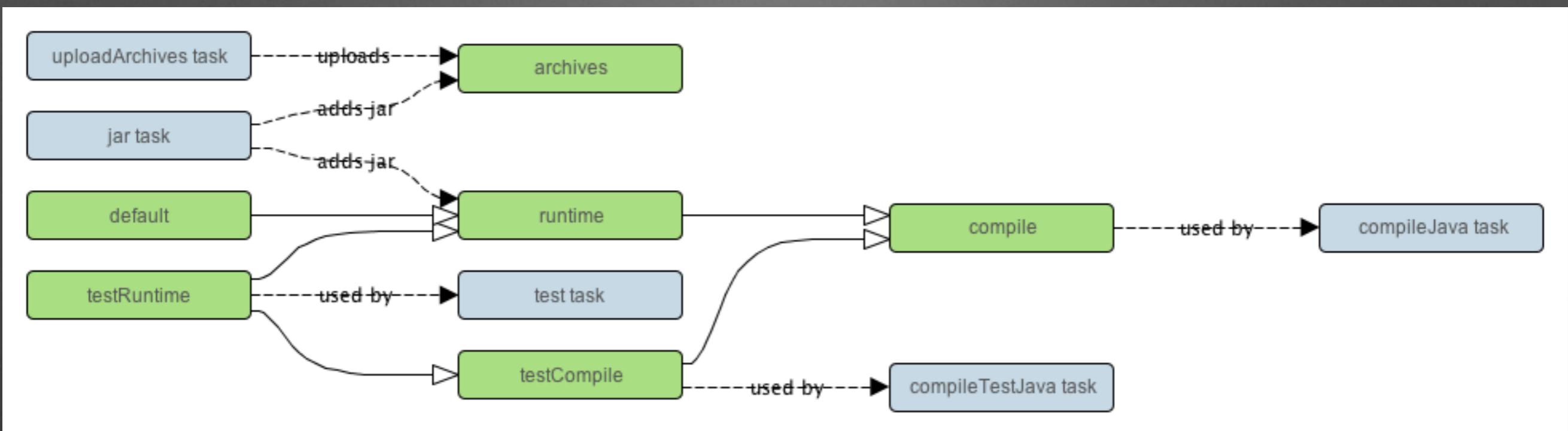
- Java プラグインの場合、以下のようなコンフィグレーション（どこでその jar が必要とされるか）がある

compile	プロダクトコードのコンパイルに使う
testCompile	テストコードのコンパイルに使う
runtime	プロダクトコードのクラスの実行に使う
testRuntime	テストコードのクラスの実行に使う



- テストでしか使わない jar は testCompile に指定する
(たとえば、JUnit)

Java プラグインの コンフィグレーション全体像



http://gradle.monochromeroad.com/docs/userguide/java_plugin.html

使う jar を指定する？

```
compile 'org.slf4j:slf4j-api:1.7.12'
```

- ・ ログ出力ライブラリ 「slf4j-api-1.7.12.jar」 を使う ...と指定している。
- ・ この文字列をどうやって見つけたらいいのか？



「mvn repo」
「maven repository」
とかでもひっかかります

Maven レポジトリを検索するサービス

- <http://mvnrepository.com>

The screenshot shows the mvnrepository.com search interface. The search bar at the top contains the query "Apache POI". Below the search bar, a graph titled "Artifacts/Year" shows the growth of artifacts over time from 2004 to 2015, reaching approximately 1124k. To the right of the graph, the search results are displayed under the heading "Found 12771 results". The results list three entries related to Apache POI:

- 1. Apache POI (org.apache.poi > poi under Excel Libraries) - 313 usages
- 2. Apache POI (org.apache.poi > poi-ooxml) - 214 usages
- 3. Apache POI (org.apache.poi > poi-scratchpad) - 45 usages

Each result entry includes a thumbnail icon of a person holding a briefcase, the artifact name, its group ID, usage count, and a brief description. At the bottom of the search results, there is a Netflix advertisement for the series "Narcos". On the right side of the page, there are two sidebar sections: one featuring a doctor speaking about smoking and another featuring a photo of Prime Minister Abe of Japan.

http://mvnrepository.com

- キーワード入力すれば、たどっていけるはず

The screenshot shows the mvnrepository.com website interface. At the top, there's a navigation bar with icons for back, forward, search, and user profile. The URL in the address bar is `mvnrepository.com/artifact/org.apache.poi/poi/3.13`. Below the address bar, there's a chart titled "Artifacts / Year" showing the popularity of the Apache POI artifact over time, with a significant increase starting around 2010. To the right of the chart, the artifact details for "Apache POI » 3.13" are displayed. The details include:

- Artifact:** Apache POI
- POM File:** View
- Date:** (Sep 22, 2015)
- HomePage:** <http://poi.apache.org/>
- Organization:** Apache Software Foundation

Below these details, there are download links for various build tools: Maven (Download (JAR) 2.4 MB), Ivy, Grape, Gradle, Buildr, SBT, and Leiningen. A search bar at the bottom contains the query `'org.apache.poi:poi:3.13'`.

On the right side of the main content area, there's a sidebar with an advertisement for Amazon Basics products, featuring a backpack, batteries, and a USB cable. Below the sidebar, there's a small image of Muhammad Ali.



3. Gradle で便利なものを作ろう

おことわり m(_ _)m

- ・ ビルドツールの王道は、Javaなどのソースコードをビルド、デプロイすること。それを自動化すること。
 - ・ ビルド：コンパイルする、テストする、Warを作る etc
 - ・ デプロイ：ビルド成果物をWeb/Appサーバに反映する
- ・ 今日はあえて、そこにはあまり触れません。
Gradle は ”自動化ツール”。まずそこを実感ください。

ツールを作りたいと思ったことがありますか？

- 単純作業や繰り返し作業、プログラムで書こうかな...
- そういったとき、ツールを何で作りますか？
 - batファイル、シェルスクリプト
 - Excel 行関数 × 複数行をコピペ
 - Excel マクロ

なぜ Excel マクロなのか？

- ・ インプットとなるものが、Excel ファイルなことが多い
- ・ Excel は、もともと便利だ。コピペしやすい
- ・ 「マクロの記録」が最強すぎる
- ・ Excel マクロは Excel ファイルに含まれる
(Excelはプログラムコードと実行ファイルが一体化している)
- ・ ツールのGUIもExcelシート上で表現できる

なぜ Java で作らないのか？

- ・そもそもゼロから作るやり方を知らない
(かっちょいいEclipseプロジェクト名をつけたものの、手が止まる病)
- ・いちいちEclipse 立ち上げるのが面倒だ。
- ・どうやって配布するんだ？ jar? exe? Tomcat?
- ・ソースコード (.java) と実行ファイル (.jar) が分離する

ツール作りでの Want

1. 色々なことがしたい。組み合わせて作りたい
(Excel入出力、ファイル操作、DBアクセス etc)
2. 作った後はラクに配布したい
ツールのために、余計なものをインストールしたくない
3. 実行ファイルにソースコードを同梱したい
4. 自動実行させたい
(定時実行、コミットトリガーで実行)



Excel は
2 と 3 が得意

Groovy だからラクにできること

- Gradle は Groovy のスクリプトを拡張したもの。ビルドツールとはいえ、プログラミングができるようなもの。
- Groovy は便利な API が豊富なので、積極的に使おう。
 - ファイル入出力
 - HTTP リクエスト送信
 - SQL 実行
 - XML / JSON との入出力

CI との連携

- ビルドツールで色々なこと（ビルド・テストなど）を自動化していくと、**継続的インテグレーション (CI)** につながっていく。
- Gradle は CI サーバの代表格 Jenkins で簡単に動かせる（少なくとも Maven と同じくらいの容易さ）
- Jenkins で Excel マクロはそう簡単には動かせないよ。



4. ここまでできる Gradle

(ふつうはやらないかも)

① Excel 読み込み→ファイル出力

- Apache POI を使えば、Excel を入出力できる
(おそらく Excel マクロよりも簡単に読み込みできる)
- Excel 設計書からのファイル生成 →コード自動生成！

build.grade ファイル

```
buildscript {  
    repositories {  
        mavenCentral()  
    }  
    dependencies {  
        classpath 'org.apache.poi:poi:3.13'  
        classpath 'org.apache.poi:poi-ooxml:3.13'  
    }  
}  
  
import org.apache.poi.ss.usermodel.*
```



```
task generateMessageIdJava << {  
  
    Workbook workbook = WorkbookFactory.create(  
        new File("メッセージマスター.xls"))  
    Sheet sheet = workbook.getSheetAt(0)  
  
    new File("MessageId.java").withWriter("UTF-8") { writer ->  
  
        writer << "package sample.constants;\n\n"  
        writer << "public class MessageId {\n\n"  
  
        (3 .. sheet.getLastRowNum()).each { rnum ->  
            Row row = sheet.getRow(rnum)  
  
            String messageId = row.getCell(0).getStringCellValue()  
  
            // メッセージIDが記載されていない行は書き出さない  
            if (messageId?.length() == 0) return  
  
            String message = row.getCell(1).getStringCellValue()  
  
            writer << "    /** $message */\n"  
            writer << "    public static final String $messageId"  
            writer << "        = \"$messageId\";\n"  
            writer << "\n"  
        }  
  
        writer << "}"  
    }  
}
```

インプットとなるファイル（設計書）

	A	B	C	D	E
1	メッセージマスタ				
2					
3	メッセージID	メッセージ	引数1	引数2	引数3
4	MSG001	検索しました。検索結果は{0}件です。	検索件数		
5	MSG002	登録しました。			
6	MSG003	ユーザIDとパスワードを入力してログインしてください。			
7	MSG004	{0}は入力必須です。	項目名		
8	MSG005	{0}は{1}以下の数値で入力してください。	項目名	最大値	
9					
10					
11					
12					
13					
14					

出力されたファイル

```
package sample.constants;

public class MessageId {

    /** 検索しました。検索結果は{0}件です。 */
    public static final String MSG001 = "MSG001";

    /** 登録しました。 */
    public static final String MSG002 = "MSG002";

    /** ユーザIDとパスワードを入力してログインしてください。 */
    public static final String MSG003 = "MSG003";

    /** {0}は入力必須です。 */
    public static final String MSG004 = "MSG004";

    /** {0}は{1}以下の数値で入力してください。 */
    public static final String MSG005 = "MSG005";

}
```

(応用例) Jenkins × Gradle × SVN

1. 定期的にSVN上の設計書ファイル更新をチェックする
(Jenkins を使えば簡単にできるよ)
2. 更新があったら最新取得して、Gradle タスクを実行
3. 更新された設計書ファイルをインプットに、
Gradle タスクでソースコードファイルを生成する
4. 生成したソースコードファイルをSVNにコミットする

② Redmine から情報を抽出する

- Redmine には「REST API」という機能がある
実は色々な情報を引っ張ってこれる
http://www.redmine.org/projects/redmine/wiki/Rest_api
- Groovy は HTTP リクエストを送るのがとても簡単
- JSONやXMLでレスポンスが返ってくるけど、
Groovy なら余裕で解析可能 ドヤッ!!

Redmine REST API を有効にする

- [管理] > [設定] > [認証] タブ
[RESTによるWebサービスを有効にする] にチェック



Redmine

検索: プロジェクトへ移動...

設定

管理

- プロジェクト
- ユーザー
- グループ
- ロールと権限
- トラッカー
- チケットのステータス
- ワークフロー
- カスタムフィールド
- 列挙項目
- 設定
- LDAP認証
- プラグイン
- 情報

認証が必要

自動ログイン

ユーザーによるアカウント登録

ユーザーによるアカウント削除を許可

パスワードの最低必要文字数

パスワードの再発行

OpenIDによるログインと登録

RESTによるWebサービスを有効にする ▲

JSONPを有効にする

セッション有効期間

有効期間の最大値

無操作タイムアウト

警告: この設定を変更すると現在有効なセッションが失効する可能性があります。

保存

Redmine管理者でないと
この設定はできません

APIアクセスキーを取得する

- ・ [個人設定] メニューに「APIアクセスキー」が表示されるようになるので、キーをメモしておく

The screenshot shows the Redmine 'Personal Settings' page. At the top right, there is a yellow arrow pointing to the 'Personal Settings' link in the top navigation bar. On the right side of the page, there is another yellow arrow pointing to the 'API Access Keys' section under the '显示' heading. The 'API Access Keys' section displays a key ID: 720af9fb9107dffce2cb9ba3eaf25132fcdbd980, with a note indicating it was created 2 months ago and can be reset.

ホーム マイページ プロジェクト スクラム統計 管理 ヘルプ

Redmine ログイン中: 個人設定 ログアウト

検索: プロジェクトへ移動...

個人設定

情報

名前 *

苗字 *

メールアドレス *

言語 Japanese (日本語)

バックログ

Task color #AEC54E

保存

メール通知

ウォッチまたは関係している事柄のみ

自分自身による変更の通知は不要

設定

メールアドレスを隠す

タイムゾーン

コメントの表示順 古い順

データを保存せずにページから

移動するときに警告

個人設定

ログイン: 作成日: 2015/09/12 17:49

自分のアカウントを削除

Atomアクセスキー

Atomアクセスキーは2ヶ月前に作成されました (リセット)

APIアクセスキー

表示

720af9fb9107dffce2cb9ba3eaf25132fcdbd980

APIアクセスキーは2ヶ月前に作成されました (リセット)

ブラウザでAPIを試してみる

- JSON形式で、チケットの一覧を取得する
 - `http://<RedmineのURL>/issues.json?key=<さっきのAPIキー>`
 - XML形式でも
`http://<RedmineのURL>/issues.xml?key=<さっきのAPIキー>`



A screenshot of a web browser window displaying a JSON API response. The URL in the address bar is `192.168.56.2/issues.json?key=720af9fb9107dffce2cb9ba3eaf25132fc...`. The browser interface includes standard navigation buttons (back, forward, search) and a toolbar with icons for star, message, and more.

```
{"issues": [{"id": 4, "project": {"id": 1, "name": "\u30c6\u30b9\u30c8\u30d7\u30ed\u30b8\u30a7\u30af\u30c8"}, "tracker": {"id": 2, "name": "\u6a5f\u80fd"}, "status": {"id": 1, "name": "\u65b0\u898f"}, "priority": {"id": 2, "name": "\u901a\u5e38"}, "author": {"id": 5, "name": "\u5965\u91ce \u79c0\u6a39"}, "assigned_to": {"id": 5, "name": "\u5965\u91ce \u79c0\u6a39"}, "subject": "Gradle \u306e\u30b5\u30f3\u30d7\u30eb\u30b3\u30fc\u30c9\u3092\u4f5c\u308b", "description": "\u30b5\u30f3\u30d7\u30eb\u30b3\u30fc\u30c9\u3092\u7406\u89e3\u3057\u3084\u3059\u304f\u3059\u308b\u3002", "start_date": "2015-11-18", "due_date": "2015-11-19", "done_ratio": 0, "created_on": "2015-11-19T16:00:15Z", "updated_on": "2015-11-19T16:00:15Z", "story_points": null}, {"id": 3, "project": {"id": 1, "name": "\u30c6\u30b9\u30c8\u30d7\u30ed\u30b8\u30a7\u30af\u30c8"}, "tracker": {"id": 2, "name": "\u6a5f\u80fd"}, "status": {"id": 2, "name": "\u9032\u884c\u4e2d"}, "priority": {"id": 3, "name": "\u9ad8\u3081"}, "author": {"id": 5, "name": "\u5965\u91ce \u79c0\u6a39"}, "assigned_to": {"id": 5, "name": "\u5965\u91ce \u79c0\u6a39"}, "subject": "\u52c9\u5f37\u4f1a\u306e\u8cc7\u6599\u3092\u4f5c\u308b", "description": "\u308f\u304b\u308a\u3084\u3059\u3044\u8cc7\u6599\u3092\u4f5c\u308a\u305f\u3044\u3002", "start_date": "2015-11-15", "due_date": "2015-11-20", "done_ratio": 20, "created_on": "2015-11-19T15:59:31Z", "updated_on": "2015-11-19T15:59:31Z", "story_points": null}], "total_count": 2, "offset": 0, "limit": 25}
```

JSONだと Unicode エスケープ
されるけど、とりあえず放置

Groovy で HTTP リクエスト

```
new URL("アクセスするURL文字列").text
```

```
new URL("アクセスするURL文字列").getText("UTF-8")
```

↑レスポンスの文字コード指定する場合

- これだけで HTTP リクエスト (GET) を飛ばせます。
返り値にレスポンスを文字列で受け取れます。
- REST で情報を取得するような場合はこれで十分だったり

たとえば、こんな感じ

```
String yahooTopPage = new URL("http://yahoo.co.jp").text  
println yahooTopPage.contains("五郎丸")
```

- 無限ループをつくれば、何度もリクエストを投げることもできるので、Webアプリケーションの簡易な耐久テストに使える。 etc...

Remine REST API も簡単に

```
String apiKey = "720af9fb9107dffce2cb9ba3eaf25132fcdbd980"  
String requestUrl = "http://192.168.56.2/issues.json?key=${apiKey}"
```

```
String jsonstr = new URL(requestUrl).text
```

```
println jsonstr
```

Groovyは「\$変数名」で変数の値を文字列に挿入できる

```
[issues:[{assigned_to:[id:5, name:奥野 秀樹], author:[id:5, name:奥野 秀樹], created_on:2015-11-19T16:00:15Z, description:サンプルコードで理解しやすくする。, done_ratio:0, due_date:2015-11-19, id:4, priority:[id:2, name:通常], project:[id:1, name:テストプロジェクト], start_date:2015-11-18, status:[id:1, name:新規], story_points:null, subject:Gradle のサンプルコードを作る, tracker:[id:2, name:機能], updated_on:2015-11-19T16:00:15Z], [assigned_to:[id:5, name:奥野 秀樹], author:[id:5, name:奥野 秀樹], created_on:2015-11-19T15:59:31Z, description:わかりやすい資料を作りたい。図解もいれよう。, done_ratio:20, due_date:2015-11-20, id:3, priority:[id:3, name:高め], project:[id:1, name:テストプロジェクト], start_date:2015-11-15, status:[id:2, name:進行中], story_points:null, subject:勉強会の資料を作る, tracker:[id:2, name:機能], updated_on:2015-11-19T15:59:31Z]], limit:25, offset:0, total_count:2]
```

- あとは、レスポンスとして返ってきた JSON の解析

Groovy で JSON も簡単にあつかう

```
def jsonRoot =new groovy.json.JsonSlurper().parseText(jsonstr)

jsonRoot.issues.each { issue ->
    println "#${issue.id}:${issue.subject}(${issue.assigned_to.name})"
}
```



#4:Gradle のサンプルコードを作る(奥野 秀樹)
#3:勉強会の資料を作る(奥野 秀樹)

- **groovy.json.JsonSlurper** クラス
JSON形式の文字列を多階層のMapにオブジェクト変換
- **groovy.json.JsonBuilder** クラス
JSON形式で値を保持しているオブジェクトを文字列に変換

Groovy で JSON も簡単にあつかう

```
def jsonRoot =new groovy.json.JsonSlurper().parseText(jsonstr)

println root

root.issues.each { issue ->
    println '#' + issue.id + ':' + issue.subject + '(' + issue.assigned_to.name + ')'
}
```

- **groovy.json.JsonSlurper** クラス
JSON形式の文字列を多階層のMapにオブジェクト変換
- **groovy.json.JsonBuilder** クラス
JSON形式で値を保持しているオブジェクトを文字列に変換

REST API は身边にあるかも

- ・ クラウド、サービスとか言ってるものは API が提供されていることが多い。
- ・ Jenkins も API あるよ。
- ・ 複雑なリクエストを投げたい場合は URL クラスでは力不足です。Apache HttpComponents を使うとか。

③ DB アクセスしてみる

- Groovy は簡単に SQL を発行できる。
- JDBCドライバーを依存関係に追加するところ、注意。
- SQL 発行するためのインプット情報は CSV なり、Excel なり読み込んで取得しましょう。

Groovy でのDBアクセス

- `groovy.sql.Sql` クラスを使用する
最初にDBコネクションを取得したインスタンスを経由して
SQL文を発行することができる



```
def sql = Sql.newInstance("jdbc:h2:mem:", "org.h2.Driver")
```

DB接続文字列 JDBCドライバークラス

```
String selectSql = "SELECT E.EMP_NO, E.EMP_NAME, D.DEPT_NAME FROM EMP E, DEPT D WHERE E.DEPT_NO =  
D.DEPT_NO"
```

```
sql.eachRow(selectSql) { row ->           ←eachRow はSELECT結果を1行ずつ処理するメソッド  
    println "${row.emp_no} : ${row.emp_name} ( ${row.dept_name} )"  
}
```

※SELECT だけでなく、INSERTなども当然できます。DDL (CREATE TABLEなど) もできます。

④ JavaFX で GUI なツールを

- ・間に合わなかつた・・・



5. 導入事例のご紹介

(もしかしたら、飛ばす？)

詳細は別ファイルにて紹介

導入事例から学ぶこと

- ・自動化は最初からやる。一度でも手でやつたら負け
- ・できるのであれば、自動化するための工数は確保したい
- ・ビルド環境として、PC 1 台用意してもらう
最低でもそのコストの10倍はリターンできるはず
- ・自動化できないところがボトルネックになる

導入事例での効果

- 安定的なビルド・デプロイの実現（終盤は超重要）
- 「だれでもワンクリックで○○できる」ことで、無駄な待機や残業から解放される。（これも超重要）
- Jenkins の認知度UP → 将来的な ALM への足がかり
(ただし、なにをやるものかは正確に把握していない)
(Jenkins の裏で Gradle は動くので、Gradle はまったく認知されず)

導入事例の失敗から学ぶこと

- Maven の XML 地獄より簡潔とはいえ、成長しきった Gradle のスクリプトも初心者から見れば...完全な魔法。
→ 序盤戦からメンテできる人を増やしていく努力
- 魔法すぎて、拒否反応を示される。
せっかく自動化したのに元に戻す／手でやる方に退化
→ やっていることの説明や見える化を



6. 時間が許すかぎり小ネタ

(たぶん、スライドだけ用意してしゃべらない)

デーモン起動で speed up

- Gradle のプロセスを常駐させておくことで、2回目以降の Gradle 実行を速くする。結構速くなる

- 方法①

```
gradle --daemon <タスク名>
```

- 方法②

<ユーザホームディレクトリ>/.gradle/.gradle.properties
に1行追加する「org.gradle.daemon=true」

Gradle Wrapper

- Gradle をインストールしていない環境でも Gradle を実行できるようにできる
- ビルドスクリプトを配布するのにピッタリな機能
- とはいえ、プロキシ設定が...
- Default Task と組み合わせると、gradlew を叩くだけ
- Gradle Wrapper × JavaFX で Excel マクロを超えられるかも...

IntelliJ IDEA で Gradle

あいであ

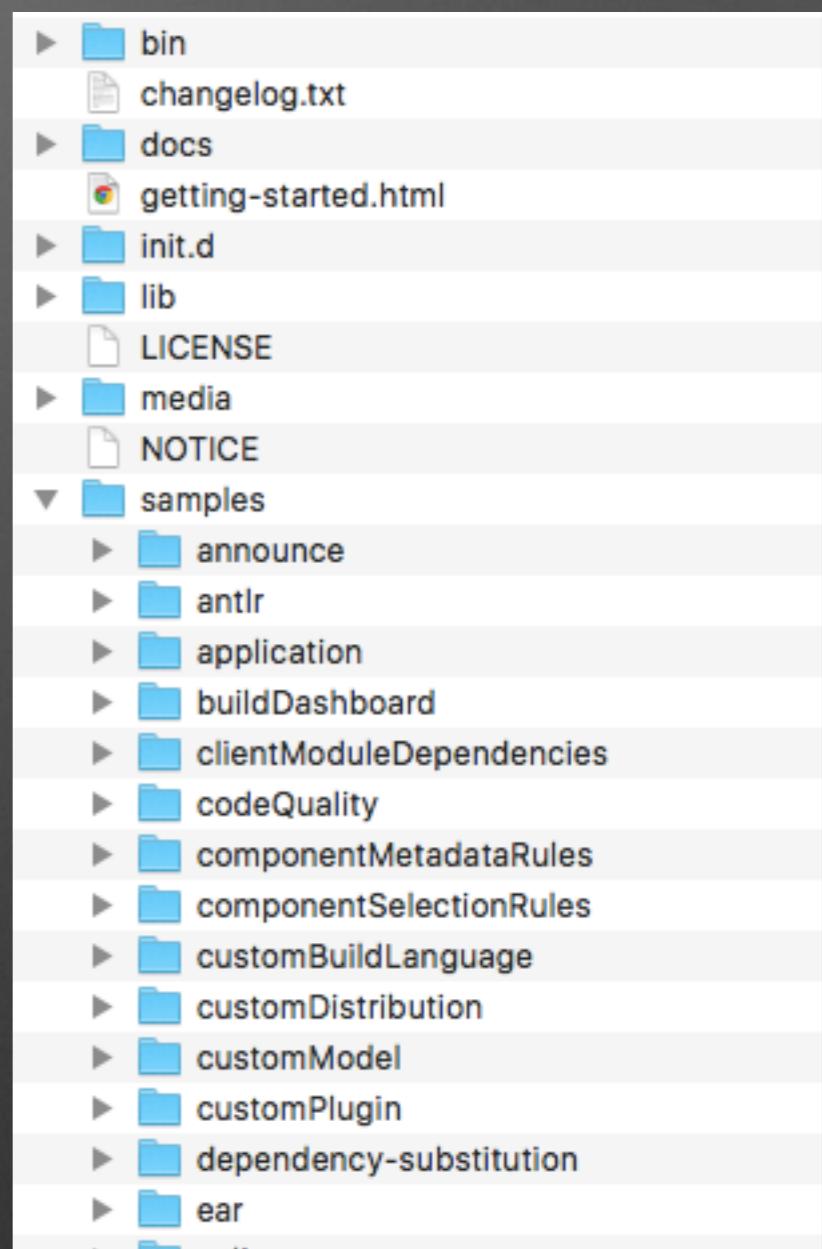
- Gradle や Groovy 向けの Eclipse プラグインはだいぶ充実してきたが、まだまだ使いづらい
- IntelliJ IDEA (<https://www.jetbrains.com/idea/>) はデフォルトで、Gradle, Groovy をサポートしている
- Ultimate Edition (有償) でなくとも、Community Edition (無償) で Gradle 使える

マルチプロジェクト構成

- 大きなシステム開発でモジュール分割して（≒ Eclipse プロジェクトを分割して）作るときなどに有効
- Eclipse プロジェクトと共に存させるためにはひと工夫が必要
- 気を抜くとカオス状態になる

Gradle の公式サンプル

- [Gradle インストールディレクトリ]
¥sample
- 初心者が見るには敷居が高いけれど意外とバラエティに富んでいる
- 困ったときに見ると助かるかも
(私はマルチプロジェクト作るときに助けられました)



Gradle でできないこと

- (思いつかない。思いつかないと信憑性が)



7. おわりに

今日のゴール (のおさらい)

ねらい

- Gradle を使って、なにかを自動化してみたくなる

目的

- Gradle をすぐに実行できる環境、取っ掛かりを手に入れる
- ちょっとしたものであれば、その作業をコード化できるようになる

今日おぼえて帰る言葉

- Gradle
- プロキシくたばれ
- Groovy
- 『Gradle 徹底入門』
- build.grade ファイル
- \$ gradle <タスク名>

今日忘れて帰る言葉

- Excel マクロ
- Maven

今日買って帰るもの



- 千里中央駅
田村書店 4階で売っています
- 淀屋橋駅
odona 2階の本屋にあるかも
- 豊洲駅
・・・本屋ある？

お伝えしたいこと

- ・ ビルドツールは取っ掛かりがなくて、なにから手を付けていいのか分からぬことが多いです。
- ・ **Gradle** はこんなことができるのだという紹介と合わせて、取っ掛かりとなるサンプルをお渡しできたはずです。
- ・ **Jenkins** と **Gradle** を組み合わせれば、開発プロジェクトにまつわる雑用（多くの場合、とても重要な雑用）を自動化して、低成本で安定して開発を回すことができます。
そして、雑用から解放されます。