

Gradle のはなし

アジェンダ

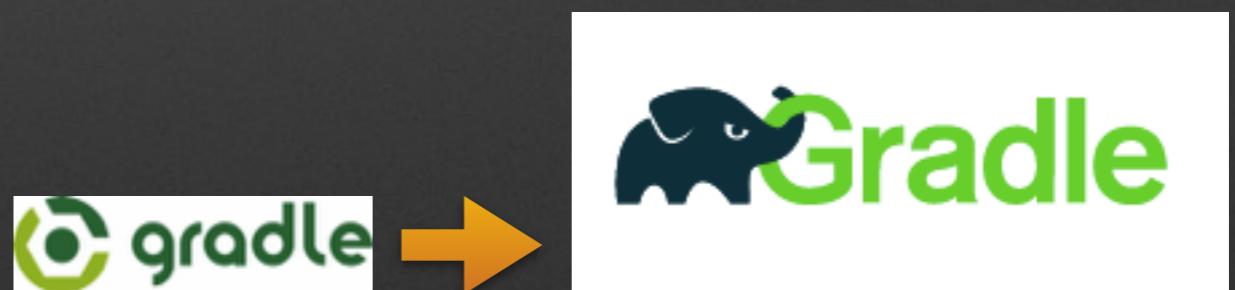
1. Gradle とは？
2. Gradle の基本的なこと
3. Gradle で便利なものを作ろう
4. ここまでできる Gradle
5. 導入事例のご紹介
6. 時間が許すかぎり小ネタ
7. おわりに



1. Gradle とは？

Gradle とは？

- 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
- Maven の代わりに使うことができる



ビルドツール

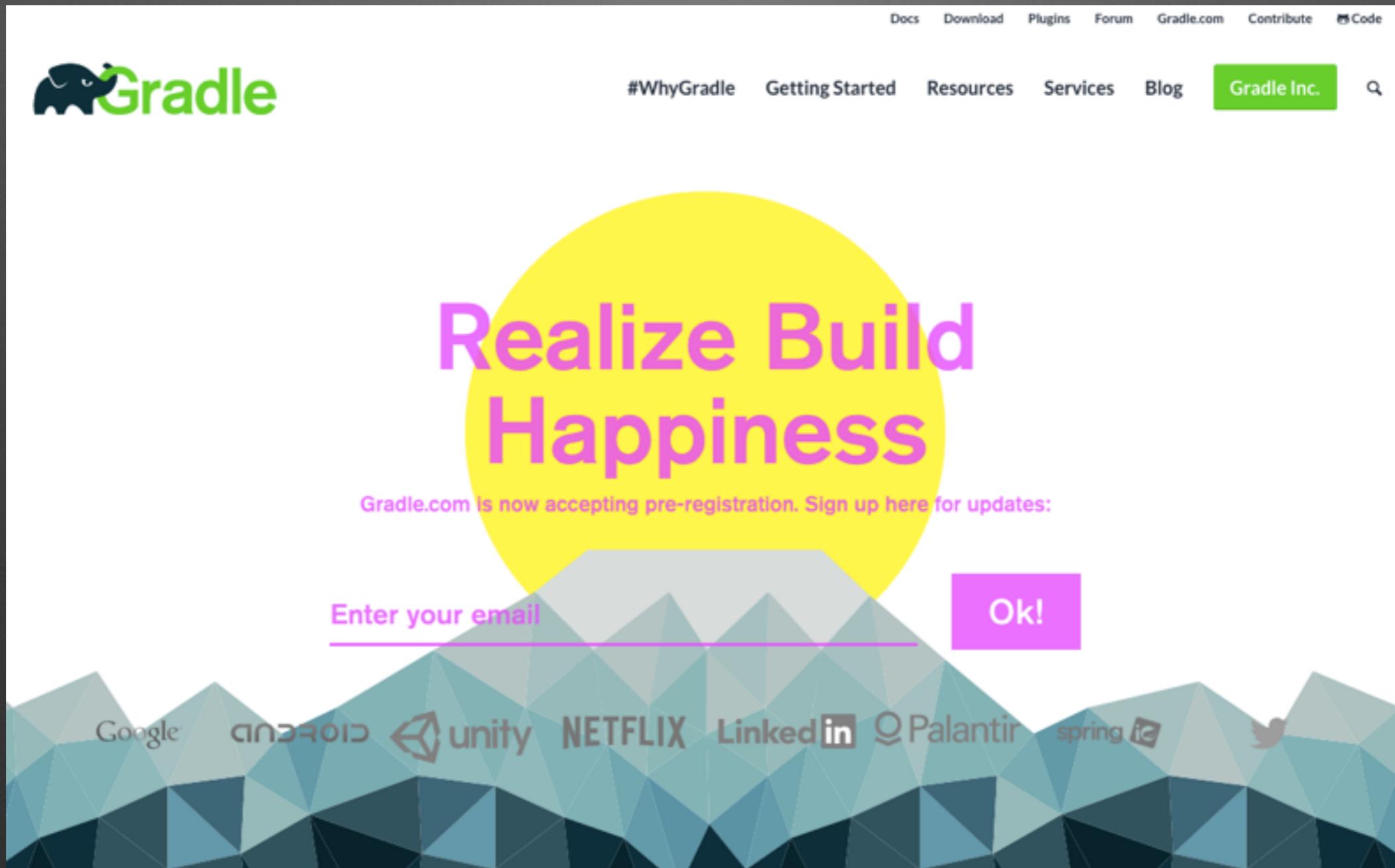
Groovy とは？

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。

Gradle の説明にもどります

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 ?

- Polygot Build
- Tool Integrations
- Robust Dependency Management

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・・・売っていますよ。
3つ紹介します。

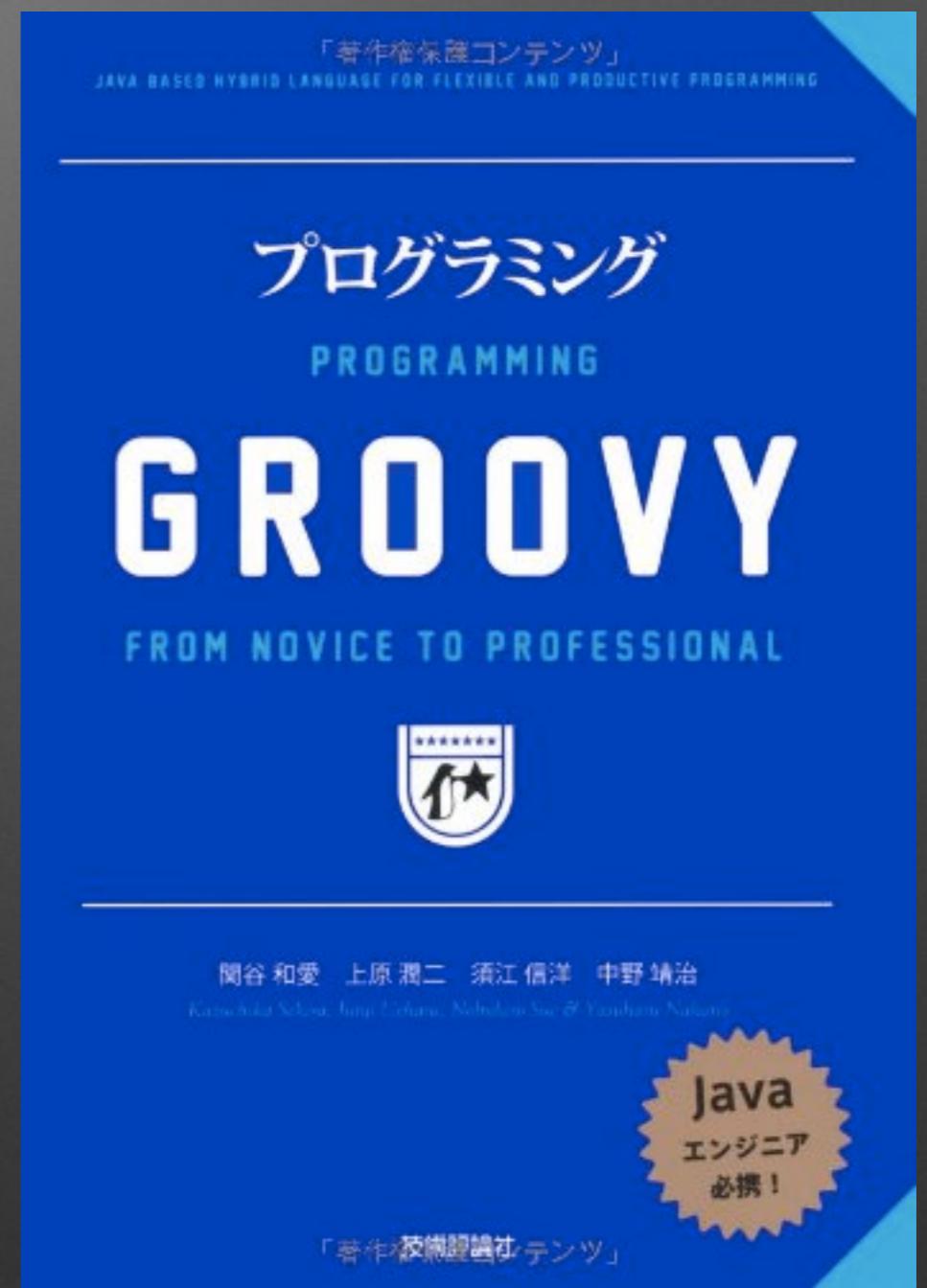
①『Gradle 徹底入門』

- 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。

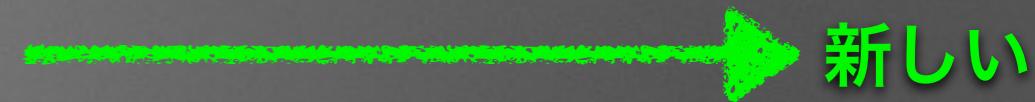


③ Gradle日本語ドキュメント

- <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



	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'  
    }  
}  
  
import org.apache.commons.io.FileUtils  
  
task copyByJava << {  
    FileUtils.copyFile(new File("org/original3.txt"), new File("dest/original3.txt"))  
}  
↑ FileUtils#copyFile(コピー先, コピー元)
```

←commons-io を依存関係に追加

←import 文

- ・ 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

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

「<<」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

組み込みタスクを使う

- Copy
- Zip

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。

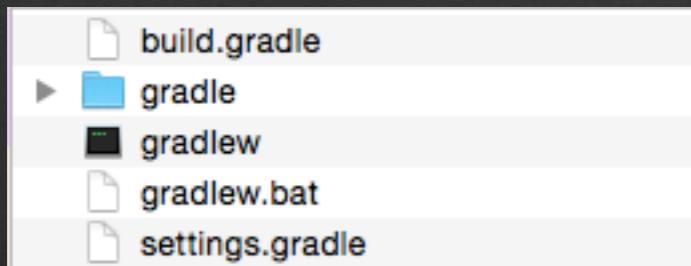
```
$ gradle init
```

```
:wrapper
```

```
:init
```

→wrapper タスクと init タスクが実行されている

```
BUILD SUCCESSFUL
```



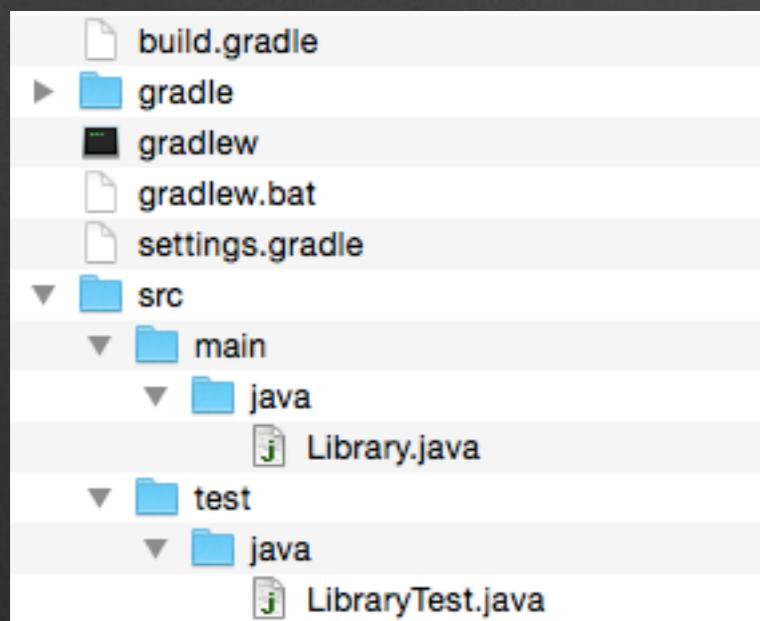
実用的なひな型もあります

```
$ 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
```

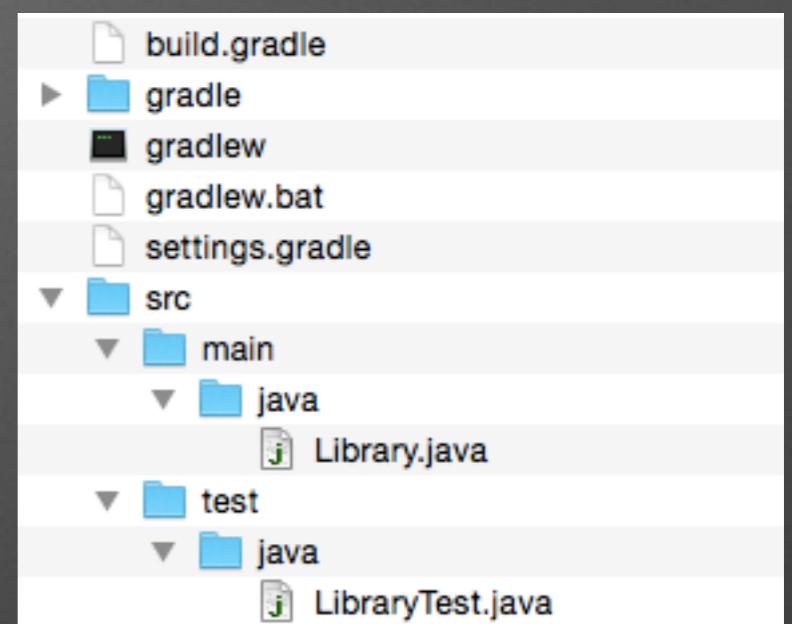


Java プロジェクトのひな型ができあがります。

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

① Java プラグインの追加

```
$ gradle build
```

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

←build タスクが依存するタスクが
いもづる式に実行される

② リポジトリの指定

- 以下のどちらかを指定してもらつたらいいです。

```
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



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 や VB 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

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）



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 JavaFX で GUI なツールを

③ 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!!

④ DB にデータ投入してみる

- 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。



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
- gradle --daemon <タスク名>
- <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」

jarを見つける方法

- 「maven repository」で検索

The screenshot shows the Maven Repository search results for 'Apache POI'. The search bar at the top contains 'Apache POI'. The results section displays three entries for Apache POI, each with a thumbnail icon, the name, the artifact ID, the number of usages, and a brief description. To the right of the results, there is a sidebar with a graph showing the growth of artifacts over time, a list of popular categories, and two news/trending sections featuring a doctor speaking about smoking and a photo of Prime Minister Abe.

mvnrepository.com/search?q=Apache+POI

MVNREPOSITORY

Artifacts/Year

Found 12771 results

1. Apache POI
org.apache.poi > poi under Excel Libraries
Apache POI - Java API To Access Microsoft Format Files

2. Apache POI
org.apache.poi > poi-ooxml
Apache POI - Java API To Access Microsoft Format Files

3. Apache POI
org.apache.poi > poi-scratchpad
Apache POI - Java API To Access Microsoft Format Files

313 usages

214 usages

45 usages

Categories | Popular | Contact Us

Web site developed by @frodriguez

タバコと余命

余命が平均的に10年短くなると言われています。

参照：英国在住の男性医師34,439人の50年間にわたる調査1900-1930年の間に産まれた吸煙者と非吸煙者の35歳以降の平均寿命の差。Doll, R. et al. BMJ 328(7455):1519.2004

NETFLIX ORIGINAL SERIES
NETFLIX ナルコス NARCOS
コカインで世界を手にいれた男
アトナリス
今すぐはじめよう

Japanese media self-censorship grows in PM Abe's

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. The URL in the address bar is `mvnrepository.com/artifact/org.apache.poi/poi/3.13`. The main content is titled "Apache POI » 3.13". It features a graph showing the number of artifacts over time from 2004 to 2015, reaching approximately 1124k. Below the graph, there's a section for "Apache POI" with a logo, the version "3.13", and "313 usages". A description states "Apache POI - Java API To Access Microsoft Format Files". A table provides details about the artifact:

Artifact	Download (JAR) (2.4 MB)
POM File	View
Date	(Sep 22, 2015)
HomePage	http://poi.apache.org/
Organization	Apache Software Foundation

Below the table are links for Maven, Ivy, Grape, Gradle, Buildr, SBT, and Leiningen. A search bar contains the query "'org.apache.poi:poi:3.13'". To the right of the main content, there's an advertisement for "amazon basics" featuring a backpack, batteries, and a USB cable. Below the ad is a small image of Muhammad Ali.

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なる
- ビルドスクリプトを配布するのにピッタリな機能
- とはいっても、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

- 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 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。