

# Maven2 プラグイン入門

第一回チキチキ  
『ant から maven2へ』  
～血があつい鉄道ならばこまるよね～

船戸 隆

tfunato@gmail.com

# 自己紹介

## ● 船戸 隆

- 株式会社エーティーエルシステムズ所属
- 日本一暑い町、岐阜県多治見市出身
- Javaプログラマ
- id:tfunato
- <http://www.canetrash.jp>
- Twitter <http://twitter.com/tfunato>



# アジェンダ

- Maven Pluginとは？
- どのように動いているか？
- プラグインの作り方
  - ファーストステップ
  - パラメータの受け渡し
  - 登録の方法
  - 起動の方法
- まとめ

The text is centered and surrounded by six circles of varying shades of yellow and green. Two circles are empty outlines, while the other four are solid. The circles are arranged in two rows: three in the top row and three in the bottom row.

MavenPluginとは

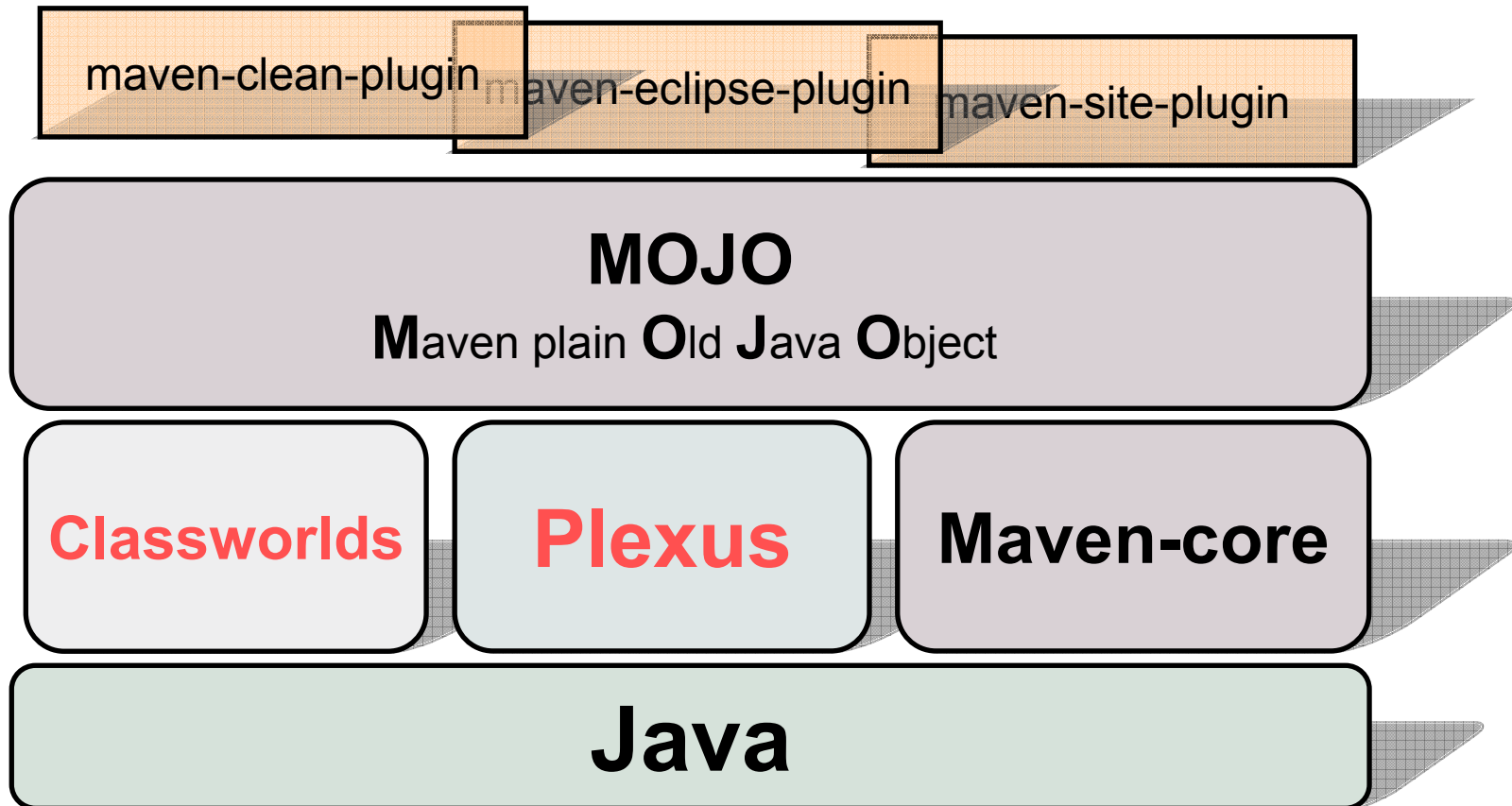
# MavenPluginとは

- プロジェクトのライフサイクルの各フェーズにおける、ゴールを実行する
  - ビルド、テスト、レポート出力...
- プロセスを簡素・共通化
  - コマンドの引数を変えるだけ
  - 誰がやっても同じ手順
- Javaでできることは何でもできる



どのように動いているか？

# アーキテクチャ概要



# 主要な使用ライブラリ

## ● **Classworlds** <http://classworlds.codehaus.org/>

- シンプルなクラスローディングライブラリ
- Mavenコアとプラグインのクラスローダを切り分ける

## ● **Plexus** <http://plexus.codehaus.org/>

- 軽量DIコンテナ
- Maven2の本体そのもの
- コンポーネントのライフサイクルのサポート
- ネストしたコンテナ
- コンポーネント単位のクラスローダーの分離



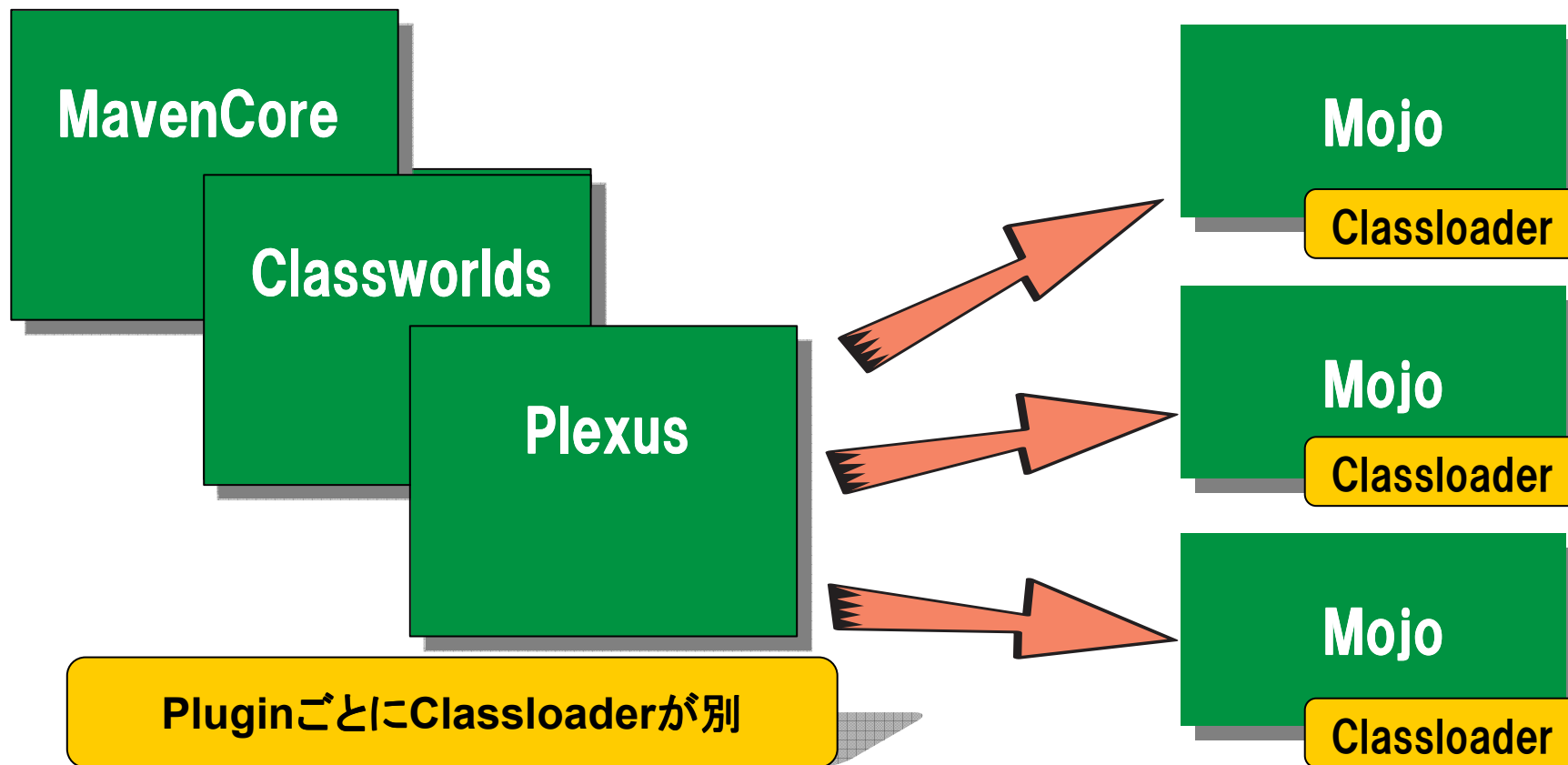
# Mojo

## ● Mojo

- Mavenプラグインインターフェース
- **M**aven **p**lain **O**ld **J**ava **O**bject
- **org.apache.maven.plugin.Mojo**インターフェース
  - 定義されてるのは
    - setLog(Log log)、getLog()、void execute()だけ！
- **org.apache.maven.plugin.AbstractMojo**
  - 継承してPluginを作る
  - void execute()を実装するだけ！

どのように動いているか？

● Maven2の仕組み



# BuildLifeCycleとは

- BuildLifeCycle

- MavenではコアコンセプトとしてBuildLifeCycleという考えに基づいている
- LifeCycleとしてビルド、配布、成果物作成、テストなどのプロセスを明確に分離して定義している
- その分離した単位をPhaseという
  - PhaseはPluginのまとまり
  - デフォルトのPhaseは  
`${M2_HOME}/lib/maven-core-2.0.X-uber.jar`のMETA-INF/plexus/components.xmlに定義

# BuildLifeCycle

## ● 定義済みLifeCycle

### ○ 大きく分けて3つ

#### ● Default

- プロジェクトのライフサイクル

#### ● Clean

- プロジェクトのクリーン
- 生成物のお掃除

#### ● Site

- プロジェクトのサイト作成
- ドキュメント、レポート作成
- 派生成果物

```
<configuration>
<lifecycle>
<id>default</id>
<phases>
<phase>validate</phase>
<phase>initialize</phase>
<phase>generate-sources</phase>
<phase>process-sources</phase>
<phase>generate-resources</phase>
<phase>process-resources</phase>
<phase>compile</phase>
<phase>process-classes</phase>
<phase>generate-test-sources</phase>
<phase>process-test-sources</phase>
<phase>generate-test-resources</phase>
<phase>process-test-resources</phase>
<phase>test-compile</phase>
<phase>process-test-classes</phase>
<phase>test</phase>
<phase>package</phase>
<phase>pre-integration-test</phase>
<phase>integration-test</phase>
<phase>post-integration-test</phase>
<phase>verify</phase>
<phase>install</phase>
<phase>deploy</phase>
</phases>
</lifecycle>
</configuration>
```

# BuildLifeCycle

## ● 定義済みPhase – Default

Phase	概略
validate	POMの精査
initialize	ビルドプロセスの準備
generate-sources	ソースコード生成
process-sources	ソースコードの成形
generate-resources	リソース生成
process-resources	リソース成形
compile	コンパイル
process-classes	コンパイル後処理
generate-test-sources	テストコード生成
process-test-sources	テストコード成形
generate-test-resources	テストリソース生成
process-test-resources	テストリソース成形

Phase	概略
test-compile	テストコードコンパイル
process-test-classes	テストコードコンパイル後処理
test	テスト実行
package	成果物のパッケージング
pre-integration-test	インテグレーションテスト準備
integration-test	インテグレーションテスト
post-integration-test	インテグレーションテスト後処理
verify	パッケージング後チェック
install	成果物のインストール
deploy	成果物の配備

Via: Maven 2.0.8

# BuildLifeCycle

## ● 定義済みPhase – Clean

Phase	Mojo	Plugin	説明
pre-clean			cleanフェーズの前に実行されるフェーズ
clean	clean	maven-clean-plugin	生成物を削除するフェーズ
post-clean			cleanフェーズの後に実行されるフェーズ

# BuildLifeCycle

## ● 定義済みPhase – Site

Phase	Mojo	Plugin	説明
pre-site			siteフェーズの前に実行されるフェーズ
site	site	maven-site-plugin	JavaDoc、レポートなどを生成するフェーズ
post-site			siteフェーズの後に実行されるフェーズ
site-deploy	deploy	maven-site-plugin	siteフェーズで生成されたものをデプロイする

The image features six circles arranged in two rows. The top row consists of three circles: the leftmost is an outline, the middle and rightmost are solid light green. The bottom row consists of three circles: the leftmost and middle are solid light green, and the rightmost is an outline. The text 'プラグインの作り方' is centered over the top row of circles.

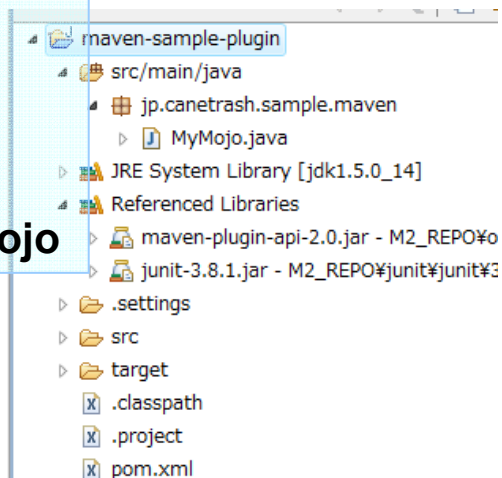
# プラグインの作り方

# ファーストステップ

- ArchetypePluginでプロジェクトを作る
  - Archetypeにmaven-archetype-mojoを指定
  - MavenPluginプロジェクトの雛形ができる

```
mvn archetype:create
  -DpackageName=任意のパッケージ
  -DgroupId=任意のグループID
  -DartifactId=maven-sample-plugin
  -DarchetypeArtifactId=maven-archetype-mojo
```

- **すでに実装コードが用意された状態**
- artifactIdのmaven-#####-pluginが**重要**
- <packaging>maven-plugin</packaging>



# Goal、Phase指定

- Goal、Phaseの指定

- Javadocで記述

```
/**
 * @goal sample
 * @phase compile
 */
public class MyMojo
    extends AbstractMojo
{
```

- 上記の例だと、ゴールはsampleでcompileのフェーズで実行される

# パラメータの受け渡し

- Pluginに外部からパラメータを渡す
  - JavaDocで記述 (Docletと同じ)
  - 渡す方法は3つ
    - 起動引数としてパラメータを渡す
      - -Dつきでパラメータを渡す
    - pom.xmlで定義
      - <configuration>で定義
    - settings.xmlで定義
      - プロファイルのプロパティに定義
  - 同時に定義された場合
    - **pom.xml > settings.xml > 起動引数の順で使われる**

# パラメータの設定

## ● 実際のコード例

```
/**  
 * @parameter  
 */  
private String param;
```

## ○ フィールドに@parameterを記述

- 例ではparamという名前で定義される
  - POMに定義されてるもののみ設定される
  - 起動時パラメータを受け取るにはexpressionを使う

# 指定できるパラメータの種類

- java.lang.String
- java.lang.Integer、Double、Long
- boolean
- java.util.Date
- java.io.File
- java.util.URL
- java.util.Map
- Collections
- 配列
- その他

# 登録の仕方

- mvn installするだけ
  - ローカルのリポジトリにインストールされる
  - `${user.home}/.m2/repository`

## mvn install

- packagingされるときにplugin.xmlが作成される
  - Pluginのパラメータやgoalが定義されたもの

# 起動の仕方

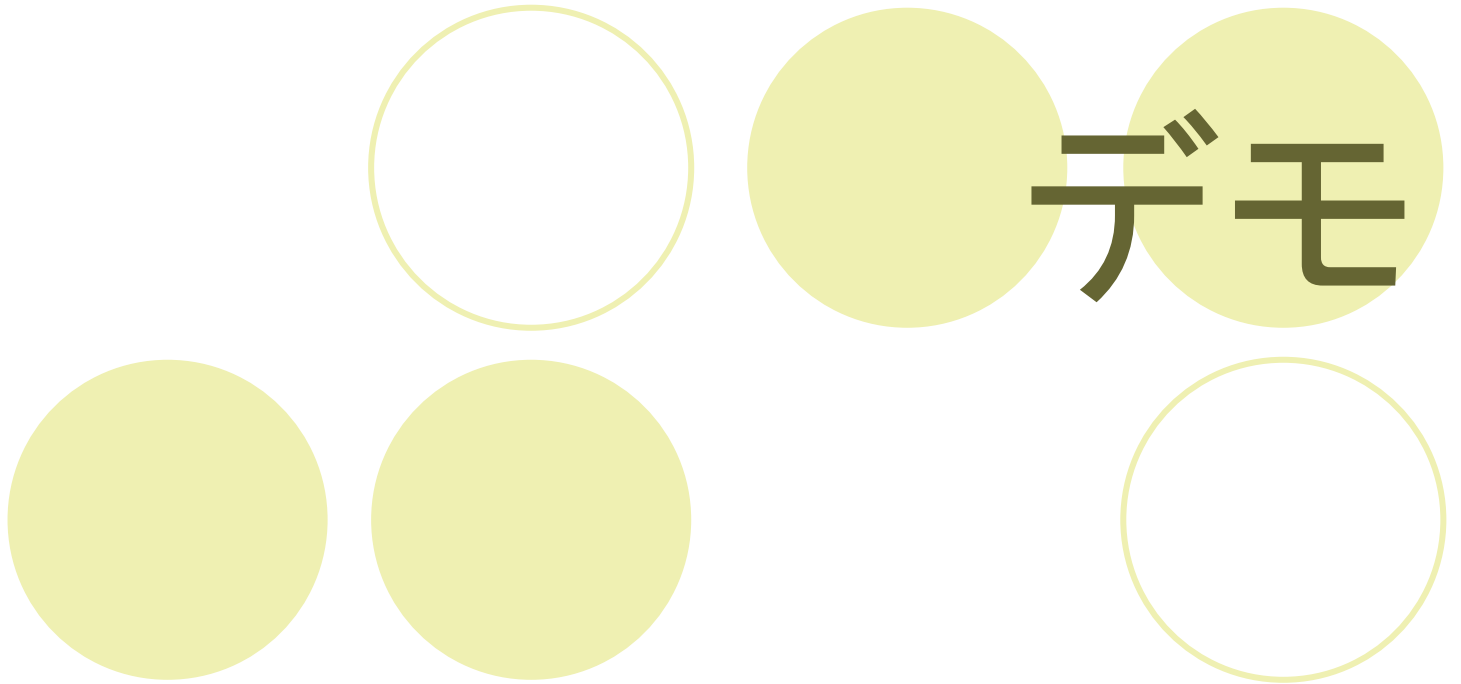
- `mvn {groupId}:{artifactID}:{version}:{goal}`
  - このうち `groupId` と `version` は省略できる
    - POM に `plugin` が定義されていれば `groupId` は省略可
    - `Version` は最新のものが使われる
    - `groupId` が `org.apache.maven.plugin` は暗黙のうちに省略可
    - `artifactID` が `maven-XXXXX-plugin` であれば `maven` と `plugin` が省略できる。

## フル指定

```
mvn jp.canetrash.maven.plugin:maven-sample-plugin:1.0:sample
```

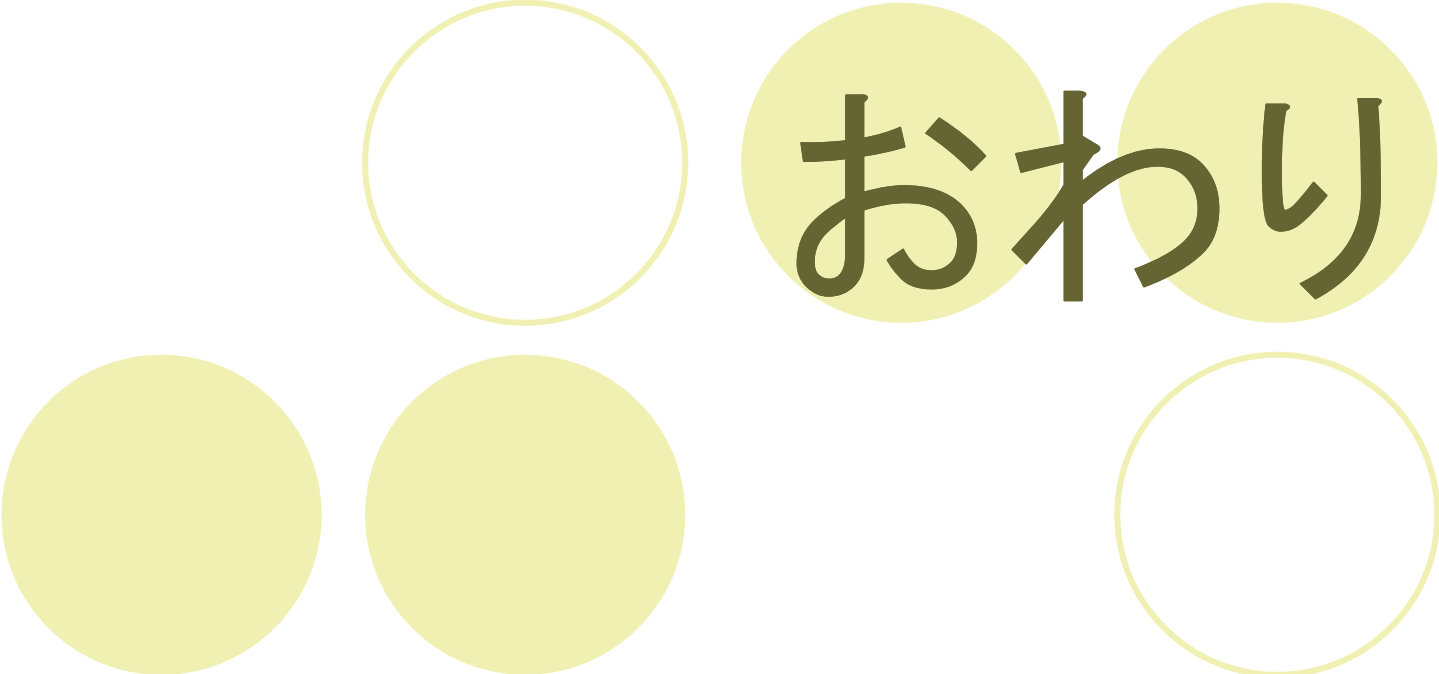
## 省略形

```
mvn sample:sample
```



# まとめ

- MavenPlugin難しくない！
- アイデアしだいでいろんなことができる
- 3rdParty製のMavenPluginもたくさんある
- Mavenをうまく使ってハッピーに



おわり