ビルドツールについて

# ビルドツールとは

ビルドとは、ソースコードのコンパイルやライブラリのリンクなどを行い、最終的な実行可能ファイルを作成すること。また、そのような作業によって生成されたソフトウェアの版。ビルドツールとは、ビルド作業を自動化、簡略化をしてくれるツールのこと。

今回は、Javaのビルドツールメインで話を進めます。

# ビルド作業

ビルド作業には主に以下の作業がある。これらの作業内容が、ビルドツールの自動化の対象になる。

* + **コンパイル**

Javaファイルを、実行可能のバイトコードファイルを作成する。

* + **依存ライブラリ(jar)の解決**

プログラムで使用しているライブラリをかき集めること。

* + **テスト**

動作確認。(Jnitなどのテストプログラムなどがある場合)

* + **設定ファイル書き換え**

配置先の環境に合わせて設定ファイルを書き換える。例えば、参照先のフォルダパスなど。

* + **パッケージング**

JarファイルやWarファイルに１つまたは複数のファイルに固める。

* + **リリース(デプロイ)**

実際に動作させる環境に配置する。

# ビルドツールの歴史

ビルドツールは、自動化範囲の拡大、ビルドスクリプトの簡素化、柔軟性に着目して進化をしていっている。

* + Make

元祖ビルドツールといえる存在。今でもC、C++で使われている。

ビルドというコンセプトを確立した。UNIX系のOSのみ対応。

実行ファイルはスクリプトで記述する。デフォルト名：Makefile

* + Ant

Apacheソフトウェア財団の開発したビルドツール。make のJava版といえるもの。今でも結構使われている。Javaなのでマルチプラットフォームに対応している。ライブラの依存関係を管理する仕組みを持っていない。(Ant 2.0からlvyが加わり解決)

実行ファイルはXMLで記述する。デフォルト名：build.xml

* + Maven

Apacheソフトウェア財団の開発したビルドツール。Apache Antに代わるものとして作られた。ライブラの依存関係を管理する仕組み(セントラルリポジトリ)を持つ。プロジェクト構造、ビルド手順に規約を持ち込み、開発者の作業を簡略化した。Mavenの規約に従うなら、作業は楽になるが、規約外のことをしようとすると難しくなる。

実行ファイルはXMLで記述する。デフォルト名：pom.xml

* + Gradle

実行ファイルをスクリプト言語にすることで、規約ベースながら、Mavenにない柔軟性を持たせた。Mavenのもっている、セントラルリポジトリを使うことができ、ライブラの依存関係を管理も可能。読み方：グレイドル

実行ファイルはGroovyというスクリプト言語で記述する。デフォルトは build.gradle

|  |  |  |
| --- | --- | --- |
| パラダイム  ビルド定義 | 手続き的 | 規約によるビルド |
| スクリプト | Make | Gradle |
| XML | Ant | Maven |

# Antの超基本

Antでは処理の内容を宣言的にxmlで記載する。

build.xml

<?xml version="1.0" encoding="UTF-8"?>

<project name="プロジェクト名" default="デフォルトターゲット" basedir="ベースディレクトリ">

<property name="プロパティ名" value="プロパティ値"/>

<target name="ターゲット名１" depends="依存するターゲット">

（このターゲットのタスクを記述する）

</target>

<target name="ターゲット名２" depends="依存するターゲット">

（このターゲットのタスクを記述する）

</target>

・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・

</project>

**超基本要素と属性**

* + **プロジェクト(project)**

XMLのルート要素

以下の属性を持つ

name･･･プロジェクト名(任意)

default･･･デフォルトターゲット (必須)

basedir･･･相対パスの基準となるディレクトリ (任意)

* + **ターゲット(target)**

Antに行わせたい作業を１つにまとめたもの

以下の属性を持つ(一部抜粋)

name･･･ターゲット名(必須)

depends･･･依存ターゲット　カンマ区切りで複数指定可能

* + **プロパティ(property)**

共通的な値や環境変数の定義

以下の属性を持つ(一部抜粋)

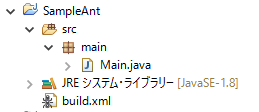
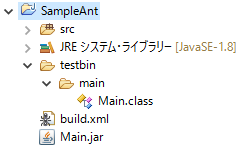
name･･･プロパティ名　プロジェクト全体で${name}で参照

value･･･プロパティ値

**例：プロジェクトをコンパイルし、実行可能Jarを作るサンプル**

プロジェクト構成

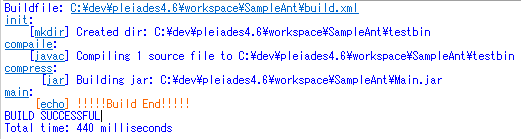
実行前→実行後

build.xml



ANT実行



# Mavenの超基本

Mavenは実行するとアーティファクトとプロジェクトサイトを作り出す。

アーティファクト･･･プロジェクトの成果物のファイル

プロジェクトサイト･･･成果物に関するHTMLファイル

アーティファクトとプロジェクトサイトを作り出すためのメタ情報をPOM(Project Object Model)と呼んでおり。POMをXML(pom.xml)で記載したものが実行ファイルとなる。

pom.xml



**超基本要素**

* + **グループID(groupId)**

アーティファクトを作成した組織のID。組織としてユニークである必要がある。

* + **アーティファクトID(artifactId)**

グループIDで示した組織が作成したアーティファクトを一意に識別するID

* + **バージョン(version)**

アーティファクトのバージョン。

* + **パッケージング(packaging)**

成果物のパッケージング

* + **名前(name)**

レポート等に出る表示名称。

* + **URL(url)**

サイトのURLです。

* + **依存関係(dependencies)**

プロジェクトの依存しているライブラリ

scopeは、その依存のスコープを定義する。

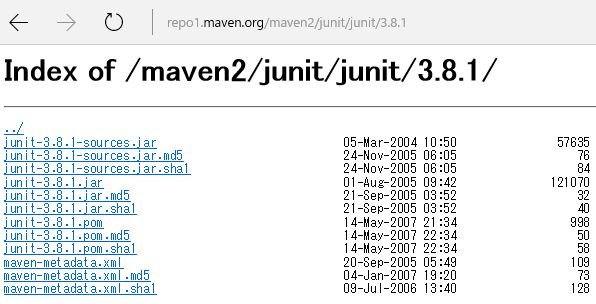
compile,provided,runtime,test,systemが定義されている。

testはテスト用という意味で、テストのコンパイルとテストの実行時に使用される。

**セントラルリポジトリ**

Mavenはあらかじめ無数のソフトウェアが配置されたリモートリポジトリ(**セントラルリポジトリ**)をもっており、ライブラリの依存関係も記録されている。以下のようにリポジトリの階層を管理することにより、pom.xmlの依存関係(dependencies)から目的のjarを探しあてることができる。

http://repo1.maven.org/maven2/[グループID]/[アーティファクトID]/[バージョン]



また、同階層内にあるmaven-metadata.xmlに、そのjarが依存するライブラリ情報が記録されており、そこからjarの依存するjarを探しあてることができる。

**ビルドライフサイクルとフェーズ**

Mavenではビルドのプロセスの大まかなステップを予め定義している。この各ステップのことをフェーズ、複数の作業ステップを纏めたものをビルドライフサイクルと呼んでいる。Mavenを実行するときは、pom.xmlと共にフェーズの指定が必要となる。

代表的なビルドライフサイクルとフェーズ

|  |  |  |
| --- | --- | --- |
| ビルドライフサイクル | フェーズ | 説明 |
| デフォルト | compile | ソースコードのコンパイル |
| test | テストの実施 |
| package | jarなどへのパッケージング |
| install | アーティファクトのローカルリポジトリへの配置 |
| deploy | アーティファクトのリモートリポジトリへの配置 |
| サイト | site | プロジェクトサイト生成 |
| site- deploy | プロジェクトサイトを公開用ディレクトへ配置 |
| クリーン | clean | Maven成果物の削除 |

**Mavenの標準ディレクトリレイアウト**

Mavenではプロジェクトのディレクトリ構成にある程度のルールが決まっている。

Mavenのコマンドによってこれらの階層は自動生成される。

ルートディレクトリ/

├pom.xml …pom.xml

├src/ …入力ファイル

│├main/ …プロジェクトのアーティファクトのためのソースファイル

││├java/ …Javaソースファイルを格納

││└resources/ …リソースファイルを格納

│├site/ …プロジェクトサイトを生成するためのファイルを格納

│└test/ …テストに関するファイル

│　├java/ …テストのソースコード

│　└resources/ …テスト用のリソースファイル

└target/ …出力ファイル

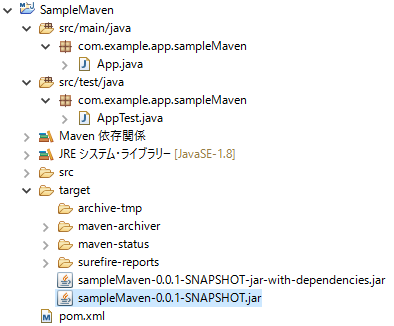
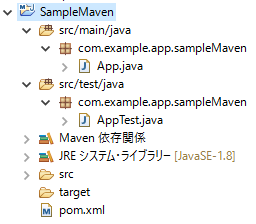
**Mavenの柔軟性**

Mavenでは、pom.xmlを実行した際に、pom.xmlの内容に応じて、各フォーズにMavenプラグインがマッピングされて実行されます。例えば、packagingがjarの時と、warの時では、それぞれmaven-jar-plugin、maven-war-pluginとういう異なるプラグインが真ピングされて実行されます。このマッピングするプラグインをpom.xmlで指定することによって、任意の拡張が可能になります。様々な、プラグインがセントラルリポジトリに登録されています。

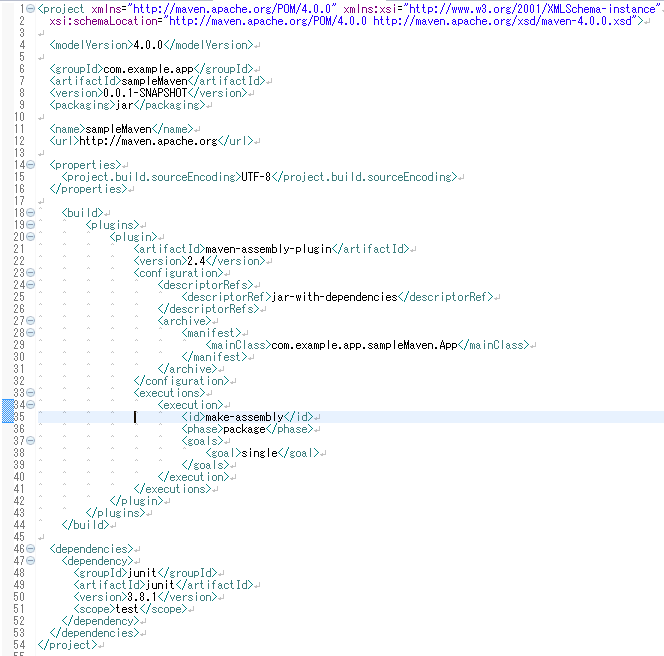
**例：プロジェクトをコンパイルし、実行可能Jarを作るサンプル**

プロジェクト構成

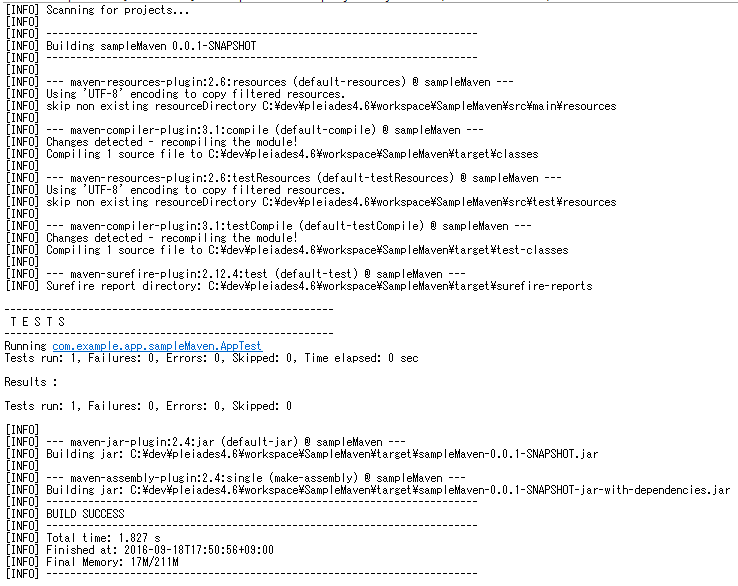
実行前→実行後



pom.xml



Maven実行

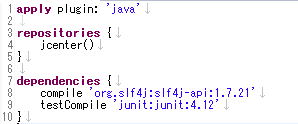


# Gradleの超基本

Mavenのもつ、規約によるビルドと依存性解決の機能を持ちつつ、Antのような手続きてきな記載も可能になっている。ビルドファイルがGroovyベースのDSLを採用している。

ビルドファイルがスクリプト言語のため、柔軟なスクリプトファイルの構築可能となっている。規約によるビルドの考えはMavenと似ている。

**build.gradle**



* + **apply plugin(1行目)**

Javaプラグインを使用する宣言。JavaプラグインはJavaプロジェクトのビルドに必要なタスクを追加する。

* + **repositories(3行目)**

依存関係解決のためリポジトリにJCenter Mavenリポジトリを使用するという定義。

* + **dependencies(7行目)**

依存ライブラリの宣言。compileはコンパイル用。testCompileはテスト用。

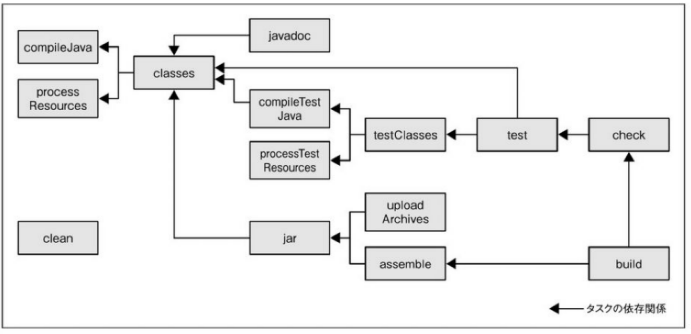
**タスク**

　Mavenのビルドライフサイクルとフェーズに該当するもの。Javaプラグインに定義されている。

代表的なタスク

|  |  |
| --- | --- |
| タスク | 説明 |
| compileJava | javaのコンパイルを行う。拡張子が.javaが対象。 |
| test | テストコードを実行する。 |
| build | プロジェクトのビルドを行う。 |
| clean | ビルド出力を削除する。 |

タスクの依存関係



Gradle**の標準ディレクトリレイアウト**

ルートディレクトリ/

├build.gradle …build.gradle

├gradlew …Gradleラッパー起動用シェルスクリプト

├gradlew.bat …Gradleラッパー起動用バッチファイル

├settings.gradle …設定スクリプト

├gradle / …Gradle用のフォルダ

│└wrapper/ …Gradleラッパー用のフォルダ

│ ├gradle-wrapper.jar …Gradleラッパーのブートストラップ

│ └gradle-wrapper.properties …Gradleラッパーの設定ファイル

└src/ …入力ファイル

├main/ …ソースファイル

│└java/ …Javaソースファイルを格納

└test/ …テストに関するファイル

└java/ …テスト用のJavaソースファイルを格納

※Gradleラッパー

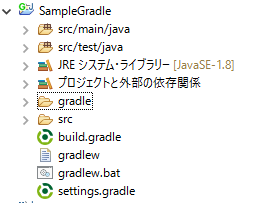
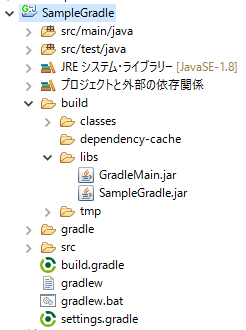
スタンドアロンで動作可能な実行ファイルを簡単に配布可能にする機能。これにより、各開発者が自分の環境にツールのインストールする必要がなくなる。

メンバーやビルド環境間でのビルドツールのバージョンを揃えられる。

**例：プロジェクトをコンパイルし、実行可能Jarを作るサンプル**

プロジェクト構成

実行前→実行後

**build.gradle**



Gradle実行

