Ant から Gant へ

Aug. 2008

宇野 るいも

(花井 志生)

Gant とは

- Apache Ant の Groovy 版
- ではなく、Rake の Groovy 版
- Rails に Rake が欠かせないように Grails には Gant が欠かせない。
- http://gant.codehaus.org/

なぜ Gant?

- Ant は面倒
  - XML 書くのは面倒
  - 条件分岐が面倒 (if, unless)
  - ちょっと複雑になると Ant Task の作成が必要で面倒
- Rake は便利だけど
  - Java のビルドには Ant の Task が欲しい（ディレクトリツリーのコピーでタイムスタンプ考慮とか）
  - Ant からの乗り換えは、基本的に書き直しが必要

簡単なサンプル

Groovy の Map 生成式

```
target(hello: 'Hello World') {
    println 'Hello World!'
}
```

クロージャ

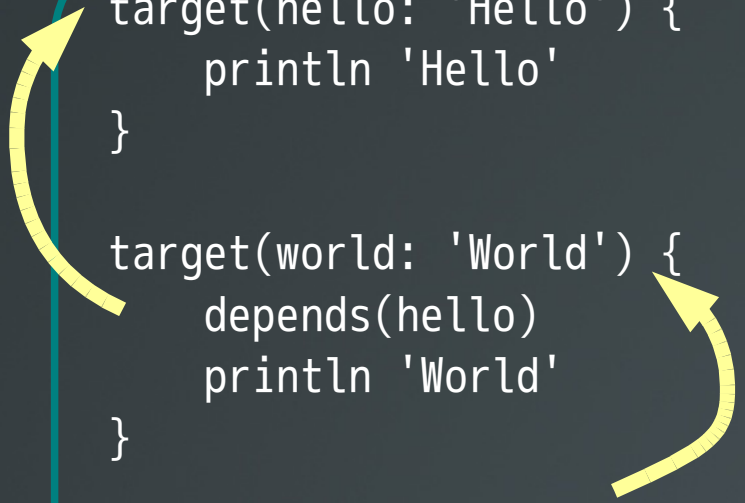
```
setDefaultTarget(hello)
```

デフォルトターゲットの指定

build.xml のかわりに build.gant を用意し、
ant のかわりに gant を起動する。
target() は、Map とクロージャを引数にとるメソッド。

依存関係

```
target(hello: 'Hello') {
    println 'Hello'
}

target(world: 'World') {
    depends(hello)
    println 'World'
}

setDefaultTarget(world)
```



Gant では、依存関係も手続き的に記述するのが特徴。

Ant の呼び出し

Ant 版

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="sample"
         default="compile"
         basedir=".">
  <target name="-prepare">
    <mkdir dir="classes"/>
  </target>

  <target name="compile"
          depends="-prepare"
          description="Compile stuff.">
    <javac destdir="classes" debug="on">
      <src path="src"/>
    </javac>
  </target>
</project>
```

Gant 版

```
target(prepare: 'Prepare') {
    Ant.mkdir(dir: 'classes')
}

target(compile: 'Compile stuff') {
    depends(prepare)
    Ant.javac(destdir: 'classes', debug: 'on') {
        src(path: 'src')
    }
}

setDefaultTarget(compile)
```

Ant 記述と 1 対 1 に対応するので、簡単に書き直せる。

XML 属性は、Map で表現。
ネストした要素はクロージャで表現。

GroovyMarkup の機能で実現されている。

標準ターゲット

- Clean ターゲット

```
includeTargets << gant.targets.Clean
cleanPattern << ['**/*~']
cleanDirectory << 'classes' << 'docs'
```

includeTargets を使用すると、簡単に機能追加できる。

cleanPattern で、消したいファイルのパターンを指定。
cleanDirectory に、消したいディレクトリを指定。

パス設定の外出し

- 文字列変数でも可能だが、File クラスを使用するのがお勧め。

```
JDK_HOME = new File('/usr/local/java/jdk/jdk')
JAVAC = JDK_HOME + 'bin' + 'javac'
SRC_DIR = new File('src')
CLASSES_DIR = new File('classes')

target(compile: 'Compile some stuff.') {
    Ant.mkdir(dir: CLASSES_DIR)
    Ant.javac(srcdir: SRC_DIR, destDir: CLASSES_DIR, executable: JAVAC,
          fork: 'yes', classpathref: 'compile.classpath', debug:'on')
}
```

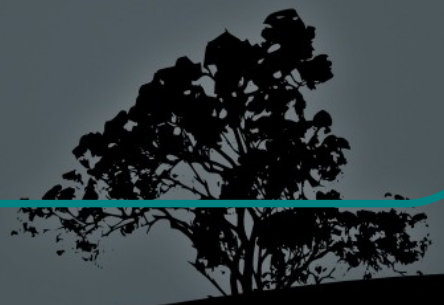
？？？
File オブジェクトに String を加算している。

MetaClass

- メタクラスを使用すると、既存のクラスの機能を変更できる。

```
File.metaClass.plus = {String child -> new File(delegate, child)}
```

これで File クラスに + 演算子のオーバーロードを追加。



別ファイルのインクルード

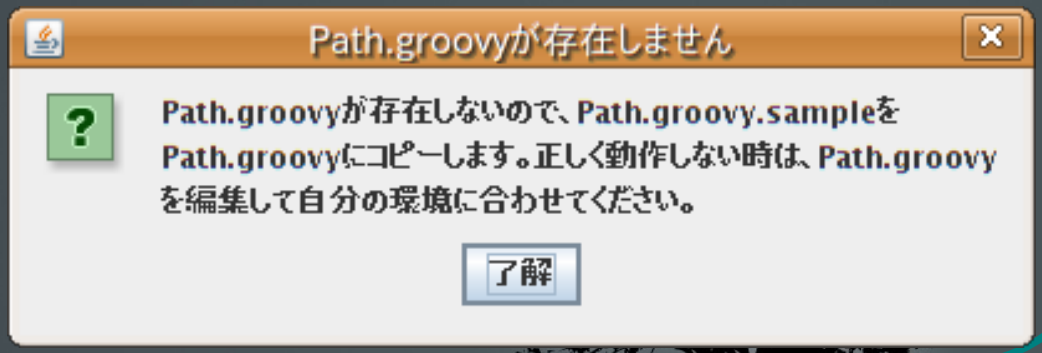
```
--- common.groovy ---
File.metaClass.plus = {String child -> new File(delegate, child)}

--- path.groovy ---
JDK_HOME = new File('/usr/local/java/jdk/jdk')
JAVAC = JDK_HOME + 'bin' + 'javac'
SRC_DIR = new File('src')
CLASSES_DIR = new File('classes')

--- build.gant ---
includeTargets <<
    new File('common.groovy') << new File('path.groovy')
```

includeTargets を File オブジェクトに対して使えば、
別ファイルをインクルードすることができる。

動的インクルード

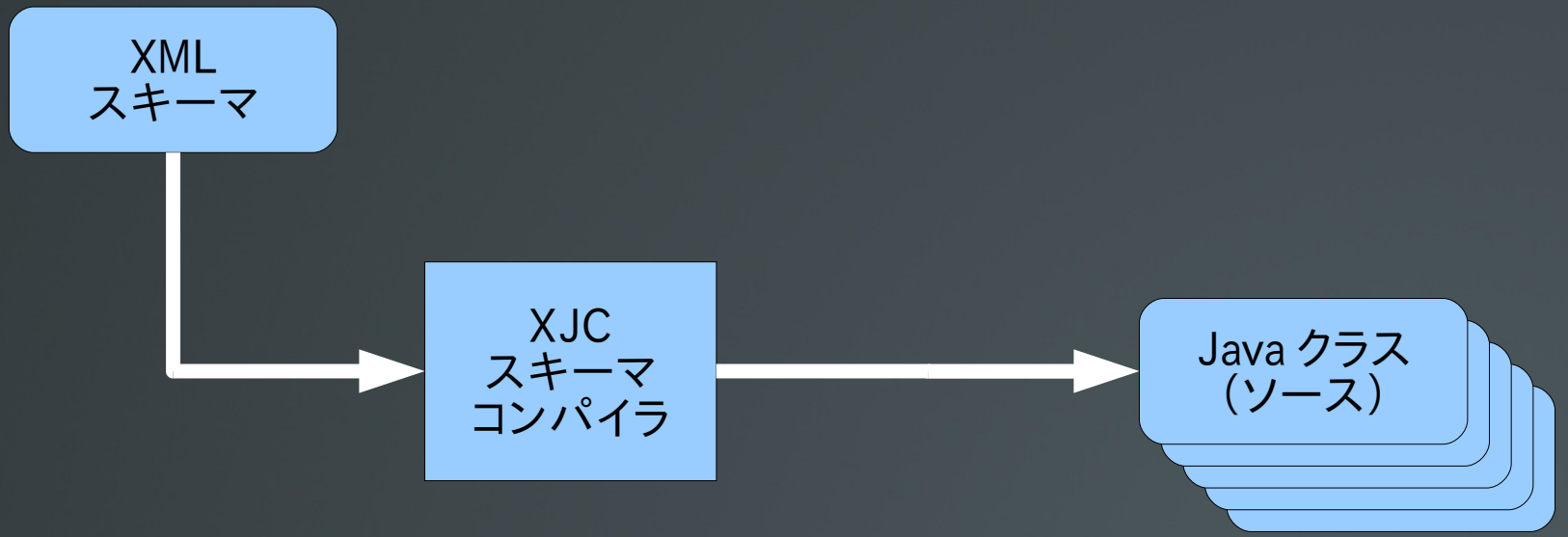
includeTargets は手続的に評価されるので、
特定の条件が成立した時にだけインクルードする
ことも可能。

```
import javax.swing.JOptionPane

File PATH_SETTING = new File('Path.groovy')
File PATH_SETTING_SAMPLE = new File('Path.groovy.sample')

if (! PATH_SETTING.exists()) {
    JOptionPane.showConfirmDialog(null,
"""${PATH_SETTING} が存在しないので、${PATH_SETTING_SAMPLE} を
${PATH_SETTING} にコピーします。正しく動作しない時は、${PATH_SETTING}
を編集して自分の環境に合わせてください。""",
        "${PATH_SETTING} が存在しません",
        JOptionPane.DEFAULT_OPTION);
    Ant.copy(file: PATH_SETTING_SAMPLE, tofile: PATH_SETTING)
}

includeTargets << PATH_SETTING
```



条件分岐の例

- jaxb でスキーマから Java クラスを生成する例。



スキーマが更新された時にだけ、XJC を実行したい。

Ant を使うと

```
<target name="-prepare">
  <mkdir dir="${gen_dir}"/>
</target>

<target name="-checkUpToDate" depends="-prepare">
  <uptodate property="needGen" targetfile="${schema}">
    <srcfiles dir= "${gen_dir}" includes="**/*"/>
  </uptodate>
</target>

<target name="all" depends="-checkUpToDate" if="needGen">
  <delete dir="${gen_dir}"/>
  <mkdir dir="${gen_dir}"/>
  <java jar="${xjc_lib}" fork="true" resultproperty="rc">
      <sysproperty key="file.encoding" value="${encoding}"/>
      <arg line="-d ${gen_dir} -p ${package_name} ${schema} -classpath $class_dir
  </java>
</target>
```

既に、なんだか良く分からなくなりつつある。

実はまだ足りない。
XJC が異常終了したら、生成ファイルを消したい。

Gant を使うと

タイムスタンプ比較

```
target(generate: 'Convert XML Schema into Java Source') {
    if (GEN_DIR.lastModified() < SCHEMA.lastModified()) {
        Ant.delete(dir: GEN_DIR)
        Ant.mkdir(dir: GEN_DIR)
        Ant.java(jar: XJC_LIB, fork: true, resultproperty: 'rc') {
            sysproperty(key: 'file.encoding', value: ENCODING)
            arg(line: "-d ${GEN_DIR} -p ${PACKAGE_NAME} ${SCHEMA}"
                        + " -classpath ${CLASS_DIR}")
        }
        if (Ant.project.properties.rc.toInteger() != 0) {
            Ant.delete(dir: GEN_DIR)
            System.exit(1)
        }
    }
}
```

スキーマコンパイラが失敗したら、ファイルを削除

タイムスタンプ比較や、プロパティの値チェックが簡単にできる！

マクロ定義

- プログラミング言語なので普通にメソッドを作れば良い。

```
def jarDirectory(List<File> dirs, File dest) {
    Ant.jar(destfile: dest) {
        for (File f in dirs) {
            fileset(dir: f)
        }
    }
}

target(dist: 'Create distribution package') {
    depends(build)
    jarDirectory([Path.GEN_CLASS_DIR, Path.CLASS_DIR], Path.JAR_FILE)
...
```

ループが使える。

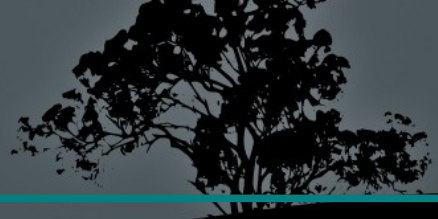
マクロ定義

- クロージャ内から呼びたい時は delegate を渡す。

```
target(dist: 'Create distribution package') {
...
    Ant.zip(destfile: Path.ZIP_FILE) {
        fileset(dir: '.', includes: 'Path.groovy')
        fileset(dir: '.', includes: Path.SCHEMA)
        fileset(dir: '.', includes: Path.JAR_FILE)
        fileset(dir: '.', includes: 'build.gant')
        zipfileset(dir: Path.DOCS_DIR,
                   prefix: Path.DOCS_DIR)
    }
}
```

fileset の記述が冗長なので、まとめたい。

```
def rootFiles(GantBuilder builder, List file) {
    for (f in file) {
        builder.fileset(dir: '.', includes: f)
    }
}

target(dist: 'Create distribution package') {
...
    Ant.zip(destfile: Path.ZIP_FILE) {
        rootFiles(delegate, [
            'Path.groovy', Path.SCHEMA,
            Path.JAR_FILE, 'build.gant'
        ])
        zipfileset(dir: Path.DOCS_DIR,
                   prefix: Path.DOCS_DIR)
    }
}
```



大規模開発へ向けて

- Ivy による依存性解決

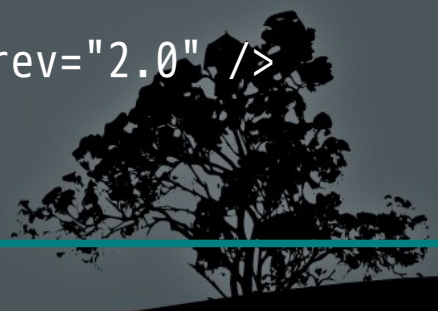
```
includeTool << gant.tools.Ivy

target(resolve: 'Resolve dependencies.') {
  ivy.retrieve()
}

setDefaultTarget(resolve)
```

要素名にハイフンが入る場合は、このようにシングルクオートで囲む。

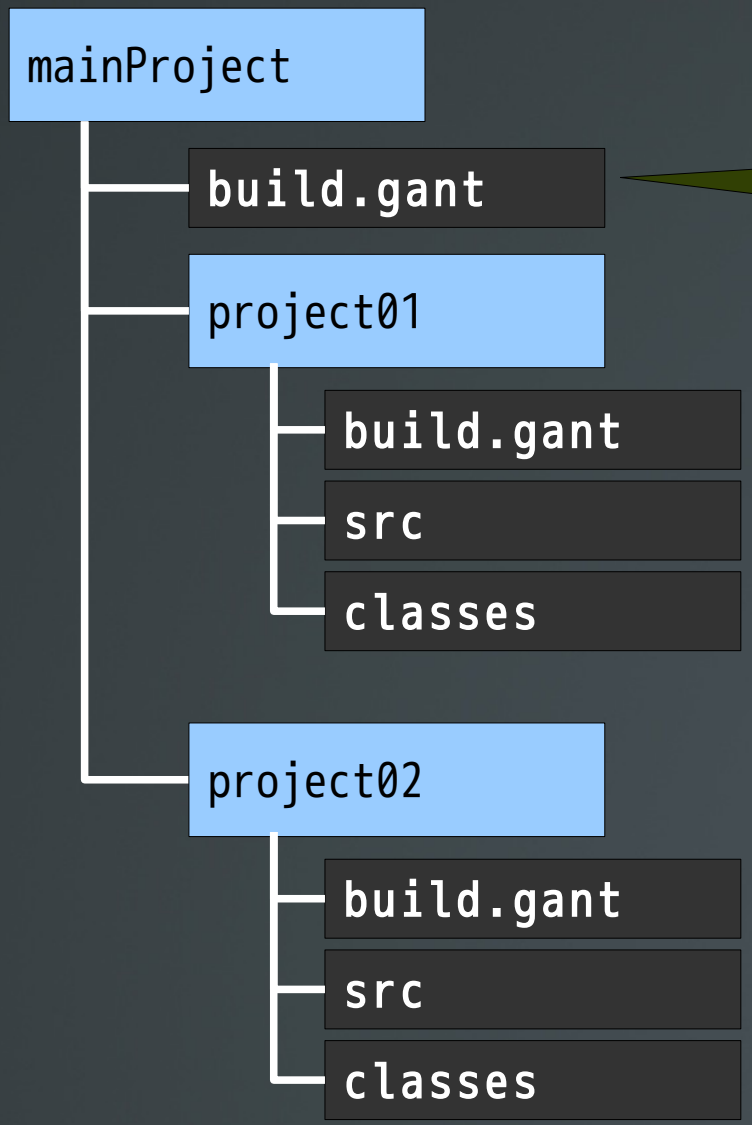
将来は以下のように書けるようになる予定

```
ivy.'ivy-module'(version : '2.0'){
  info(organisation: 'foo', module: 'bar')
  dependencies {
    dependency(org: 'commons-lang',
               name: 'commons-lang',rev: '2.0')
  }
}
```

```
--- ivy.xml ---
<ivy-module version="2.0">
  <info organisation="ruimo" module="hello-ivy" />
  <dependencies>
    <dependency org="commons-lang" name="commons-lang" rev="2.0" />
  </dependencies>
</ivy-module>
```



大規模開発へ向けて



ここでの gant 実行では、project01 と project02 をビルドしたい。

ここでの gant 実行では、project01 をビルドしたい。

課題は、パスの解決。単に new File('src') とした場合、mainProject の下で実行した場合は、mainProject/src に、project01 の下で実行した場合は、project01/src になってしまう。

gantsample.zip

大規模開発へ向けて

- gant から gant を呼び出す。

```
SUB_PROJECTS = ['project01', 'project02']

def processSubProjects(target) {
  commonDir = new File('.')

  SUB_PROJECTS.each { prj ->
    b = new GantBinding()
    baseDir = new File(prj)
    b.setVariable('BASE_DIR', baseDir)
    b.setVariable('COMMON_DIR', commonDir)
    rc = new Gant(baseDir + 'build.gant', b).processTargets(target)
    if (rc != 0) System.exit(rc)
  }
}
target(all: 'Create all stuff') {
  processSubProjects('all')
}
```

呼び出したいビルドファイルを引数にして Gant を生成して、processTargets() を呼び出せば良い。
戻り値は、エラーコード。0 なら正常。

バインディングを使うと、変数定義を渡すことができる。呼び出された側で、普通に BASE_DIR/COMMON_DIR でアクセス可能。

大規模開発へ向けて

- 各サブプロジェクトでは、

変数が定義されていなければ、カレントを BASE_DIR に、親を COMMON_DIR とする。

```
if (! binding.variables.containsKey('BASE_DIR')) BASE_DIR = new File('.')
if (! binding.variables.containsKey('COMMON_DIR')) COMMON_DIR = new File('..')

includeTargets <<
  [COMMON_DIR + "path.groovy",
   COMMON_DIR + 'build.common']

target(resolve: 'Resolve dependencies.') {
  ivy.retrieve(organisation: 'commons-lang', inline: 'true',
               module: 'commons-lang', revision: '2.0',
               pattern: "${LIB_DIR}/[artifact]-[revision].[ext]")
}

target(all: 'Create all stuff') {
  depends(resolve)
  depends(compile)
}
```

共通のビルドルール、パス設定はまとめておく。
内容は次ページ

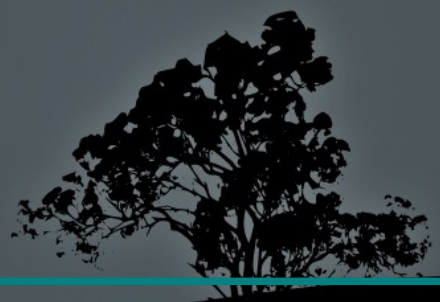
ivy.xml を用意せずに、このように直接書いても良い。

大規模開発へ向けて

- 共通のビルドルールとパス設定

```
--- path.groovy ---
LIB_DIR = BASE_DIR + 'lib'
CLASS_DIR = BASE_DIR + 'classes'
SRC_DIR = BASE_DIR + 'src'
```

```
--- build.common ---
includeTool << gant.tools.Ivy
includeTargets << gant.targets.Clean

cleanPattern <<  '**/*~'
cleanDirectory << CLASS_DIR << LIB_DIR

target(compile: 'Complile utilities') {
  Ant.mkdir(dir: CLASS_DIR)
  Ant.javac(srcdir: SRC_DIR, destdir: CLASS_DIR)
}
```



まとめ

- Java アプリケーションのビルドツールとしては理想的。手続き的に書けるし、Ant タスクも呼べる。
- Ant タスク呼び出しは、build.xml の記述と 1 対 1 に対応するので、乗り換えが簡単。
- Java の全機能が利用できる (GUI も使える) 。
- 本日のファイル、サンプルは、後日以下で公開します。
  http://www.ruimo.com/publication/
- もっとうまい技があったら、是非教えてください！