現場に活かすJakarta Project（2）：AntでJavaのビルドを簡単にする - @IT

ノートブ… 議事録 for iPad
作成日: 2014/10/18 11:32
URL: http://www.atmarkit.co.jp/ait/articles/0301/16/news002.html

現場に活かすJakarta Project（2）：

AntでJavaのビルドを簡単にする

[岡本隆史，NTTデータ]

東京高等裁判所IT専門委員が解説――情報漏えい事件のその後

今回は、Java Solutionフォーラムで行った「第7回読者調査：となりの会社はJakarta Projectを活用している？」で、Tomcatに次いで利用者が多かったビルドツールのAntを取り上げます。Antについては、基本編と応用編の2回に分けてご紹介します。基本編となる今回は、Antの概要と基本的な使い方について解説しましょう。

ビルドツールというと、makeを思い浮かべる方も多いと思いますが、AntのビルドファイルはXMLで記述するという特徴があり、Javaとの親和性も高くなっています。前述の読者調査によると、現在利用中の開発ツールとして「エディタ＋JDK」が最も多かったのですが、まさにこの開発パターンでこそAntを使うのが王道ということができるでしょう。

Borland JBuilderなどのIDEを使っている方は、「IDEを使ってビルドすればAntなんて必要ないのではないか？」と思われるかもしれません。しかし、例えば、Windows環境で開発を行い、動作確認をUNIX上で行うといったクロス開発を行っている場合、ソースコードの不具合を見つけるたびにWindows上でビルドして、UNIX環境にコピーして動作確認する……といった手間がかかる作業を繰り返していることはありませんか？動作確認を行うためのマシンがネットワークに接続されている場合はまだよいのですが、そうでない場合はフロッピーディスクやCD-Rに書き込むなど、手間が倍増します。

このようなときにも、Antを利用することにより、JDKがインストールされた環境であれば、簡単にソースからビルドすることができ、ファイルコピーの手間が省け開発効率が上がります。また、最近のIDEはAntをサポートしているものも多いので、Antで作成しておけば、ほかのIDE上で簡単にビルドすることもできます。

Antを使ってみよう

「百聞は一見にしかず」ということで、まずはAntを使ってみましょう。AntをJakartaのサイトhttp://www.apache.org/dist/ant/binaries/からダウンロードして、適当なディレクトリに解凍してください。筆者の環境では、執筆時の最新バージョン1.5.1を/usr/local/jakarta-ant-1.5.1（Windowsの場合は、c:¥usr¥local¥jakarta-ant-1.5.1）に展開しました。

では、サンプルをダウンロードして実際に試してみましょう。ここからダウンロードしてください。

Antの環境設定を行います。UNIX環境の場合、環境変数JAVA_HOMEにJDKのインストールディレクトリを、ANT_HOMEにAntのインストールディレクトリを設定し、PATHを$ANT_HOME/binに通せば準備は完了です。準備ができたら、下記のようにサンプルを展開し、antコマンドを実行してください。

●UNIX（bash）の場合

```
$ export ANT_HOME=/usr/local/jakarta-ant-1.5.1
$ export JAVA_HOME=/usr/local/j2sdk1.4.1_01
$ export PATH=/usr/local/jakarta-ant-1.5.1/bin:${PATH}
$ jar xvf hello-ant.zip
$ cd hello-ant
$ ant
Buildfile: build.xml

compile: compileターゲットの実行
[mkdir] Created dir:
/tmp/hello-ant/classes
[javac] Compiling 2 source files to
/tmp/hello-ant/classes
```

```
jar: jarターゲットの実行
[jar] Building jar:
/tmp/hello-ant/foo-1.0.jar
```

最初の環境変数の設定は、.bashrcなどに書いておくとよいでしょう。

Windowsの環境の場合は、ユーザー環境変数のJAVA_HOME、ANT_HOME、PATHをそれぞれ、下記のように設定します。

JAVA_HOME	c:¥j2sdk1.4.1_01（J2SDKのインストールディレクトリ）
ANT_HOME	c:¥usr¥local¥jakarta-ant-1.5.1
PATH	c:¥usr¥local¥jakarta-ant-1.5.1¥bin

環境設定ができたら、以下のようにantコマンドを実行してください。次のような実行結果が得られるはずです。

●Windowsの場合

```
C:\tmp> jar xvf hello-ant.zip
C:\tmp> cd hello-ant
C:\tmp\hello-ant> ant
Buildfile: build.xml

compile: compileターゲットの実行
[mkdir] Created dir: C:\tmp\hello-ant\classes
[javac] Compiling 2 source files to C:\tmp\hello-ant\classes

jar: jarターゲットの実行
[jar] Building jar: C:\tmp\hello-ant\foo-1.0.jar

BUILD SUCCESSFUL
Total Time: 7 seconds
```

これで、Javaソースのコンパイルとjarファイルfoo-1.0.jarの作成が行われました。Antは実行したカレントディレクトリにあるビルドファイルbuild.xmlを読み込み、ビルドを実行します。この例では、後で紹介するビルドの実行単位「ターゲット」のデフォルト値がjarとなっているので、jarターゲットの実行と、jarターゲットに依存したcompileターゲットを実行しています。

では、classesディレクトリの下のFoo1.classを実行しみてみましょう。以下の実行結果が得られるはずです。

●Windowsの場合

```
C:\tmp\hello-ant> cd classes
C:\tmp\hello-ant>classes>java Foo1
こんにちはAnt
```

デフォルトでは実行されませんが、サンプルにはjavadocの生成プロセスも含めているため、次のようにjavadocターゲットを指定して実行すると、apiディレクトリの下にJavaDocが生成されます。

```
ant javadoc
```

また、build.xml以外のビルドファイルを利用したい場合は、-fオプションで、

```
ant -f [ビルドファイル名]
```

で、ビルドファイルを指定することもできます。

Antの特徴

Antの具体的な解説に入るまえに、Antの特徴について整理しましょう。Antを使うと以下のようなメリットがあります。

（1）複雑なコンパイルプロセスを簡略化可能

Antを利用すると、コンパイル、jarファイル作成などのビルド手順を形式化できるので、複雑なビルド手順を簡略化できます。逆に、簡単なビルドコマンドでコンパイルできる場合（例えば、ソースファイルが1つしかなく、javacコマンドを1回実行するだけでコンパイル可能な場合など）は、ビルドファイルの作成に手間がかかり、かえって作業負荷が増えます。

（2）豊富なJava開発ツールへの対応

javac、jar、rmic、war、native2asciiなどのコマンドの実行に対応しています。Antでは、それぞれのコマンドを実行するための「タスク」と呼ば

れるクラスが用意されています。また、最近では、JavaBeansからEJBを生成するxdoclet／vdocletなど、Jakarta以外からもタスクが提供されています。

（3）処理プロセスのグループ化

ソースのコンパイルやjarファイルの生成、javadoc生成など、あるまとまった処理をターゲットにまとめることができます。例えば、javacタスクとnative2asciiタスクをまとめてコンパイルターゲットとすることができます。ターゲットにより、その時々で必要とされる処理のみを簡単に選択して実行できるようになっています。

（4）依存関係の解決

例えばjarファイルの作成の前には、ソースのコンパイルが必要です。Antはこのような処理の依存関係を解決できます。そのためjarファイルを作成する前に、ソースがコンパイルされているか否かを意識する必要がなくなります。

（5）インクリメンタルなビルド

ビルド処理を行うタスクは、基本的にタイムスタンプを調べて処理が必要なファイルだけ処理します。例えば、一度ソースをコンパイルした後でソースを変更した場合、変更があったソースのみを再コンパイルすることができます。

Antの使い方をmakeと比較して理解する

では、makeで用いるMakefileとAntのビルドファイルを対比しながら、Antの使い方を見ていきましょう。まずは、makeで使用するMakefile**（リスト1）**を見てください（なお、リスト1とリスト2は機能毎に色分けしています）。

赤	変数設定
青	jarファイルの生成
緑	コンパイル実行
紫	javadoc生成

灰色	ファイルのクリーンアップ

リスト1　Makefileの例

```
######## 変数の設定 ########

# ソースファイルの一覧
BUILD_SRC=src/foo1.java src/pkg/foo2.java

# ソースファイルから生成されるクラスファイル
BUILD_DEST=classes
BUILD_FILES=$(BUILD_DEST)/foo1.class $(BUILD_DEST)/pkg/foo2.class

# JavaDocファイル
JAVADOC_DEST=api
JAVADOC_FILE=$(JAVADOC_DEST)/foo1.html $(JAVADOC_DEST)/pkg/foo2.html

# jarファイル
JAR=foo.jar

## jarファイル生成
jar:$(JAR)
$(JAR):$(BUILD_FILES)
    jar cvf   $(JAR) -C $(BUILD_DEST)/ .

## コンパイル実行
compile:$(BUILD_FILES)
$(BUILD_FILES):$(BUILD_SRC)
    mkdir -p $(BUILD_DEST)
    javac -d $(BUILD_DEST) $(BUILD_SRC)

## javadoc生成
javadoc:$(JAVADOC_FILE)
$(JAVADOC_FILE): $(BUILD_SRC)
    javadoc -d api $(BUILD_SRC)

## ファイルのクリーンアップ
clean:
    rm -fr $(BUILD_DEST) $(JAVADOC_DEST)
```

　このMakefileをAntのビルドファイル（build.xml）に書き換えると次のようになります（先ほど実行したBuild.xmlを少し簡略化しています）。

リスト2　Antのビルドファイル（build.xml）

```
<?xml version="1.0" encoding="Windows-31J"?>

<project name="foo" default="jar" basedir=".">


 <!-- ソースファイルのディレクトリ -->
```

```
  <property name="build.src" value="src"/>

  <!-- Classファイルを出力するディレクトリ -->
  <property name="build.dest" value="classes"/>

  <!-- Javadoc を出力するディレクトリ -->
  <property name="javadoc.dest" value="api"/>

  <!-- プロジェクト名 -->
  <property name="project" value="foo"/>

  <!-- バージョン番号 -->
  <property name="version" value="1.0"/>

  <!-- 出力jarファイル -->
  <property name="buuld.jar" value="${project}-${version}.jar"/>


 <!-- jarファイル作成 -->
  <target name="jar" depends="compile">
    <jar
          jarfile="${build.jar}"
          basedir="${build.dest}">
    </jar>
  </target>


 <!-- コンパイル実行 -->
  <target name="compile">
    <mkdir dir="${build.dest}" />
    <javac srcdir="${build.src}"
           debug="${debug}"
   destdir="${build.dest}"
    />
  </target>


 <!-- Javadoc作成 -->
  <target name="javadoc">
    <javadoc
      packagenames="*"
      sourcepath="${build.src}"
      destdir="${javadoc.dest}">
    </javadoc>
  </target>


 <!-- ファイルのクリーンアップ -->
  <target name="clean">
    <delete dir="${build.dest}" />
    <delete file="${build.jar}" />
  </target>

</project>
```

最も大きな違いとして、Makefileはコマンドの羅列で記述されているのですが、AntのビルドファイルはXMLで記述されています。具体的に機能ごとに違いを見てみましょう。

■デフォルトの処理の指定

ここでのデフォルトの処理とは、オプションの指定を省略したとき、AntとMakeが何の処理を行うか、ということを指します。ビルドコマンドを実行したときのデフォルトの処理は、Makeの場合、一番初めに記述された処理になりますが、Antではprojectタグで指定します。上記の例では、jarファイル作成をデフォルトの処理にしています。Makefileファイルでは jar:$(JAR)の部分が、Antのビルドファイルでは<project .. default="jar"…>の部分が該当します。

■変数の設定

AntもMakeもビルドファイルの中で自由に参照可能な変数を宣言することができます。makeは、Makefileの先頭のブロックで以下のように変数設定を行います。

変数名=値

そして、以下のように変数の参照を行います。

$(変数名)

これに対して、Antは、以下のようにpropertyタグを使って、変数（Antでは変数のことを通常プロパティと呼びます）の設定を行います。

<property name="変数名" value="値"/>

そして、以下のように変数の参照を行います。

${変数名}

■コマンドの実行

makeでは、例えば下記の記述を見れば分かるように、mkdir、javacなどのシェルコマンドを直接呼び出して処理を進めます。

```
$(BUILD_FILES):$(BUILD_SRC)
  mkdir -p $(BUILD_DEST)
    javac -d $(BUILD_DEST) $(BUILD_SRC)
```

これに対してAntでは、前述したようにjavac、jar、javadocなど、実際に実行するコマンドに対応づけられたタスクを実行することで処理を進めていきます。タスクは、次のように表記します。

```
<タスク名 引数(属性)="値" ...>
    <引数(要素)>値</引数(要素)>
</タスク名>
```

引数は、各タスク固有の値を取りますが、例えば、mkdir、delete、copyなどのファイル操作を行うタスクは、dir（ディレクトリを指定する）などの共通引数を指定できます。また、引数は、属性と要素の2種類が指定できますが、基本的には属性に引数を設定します。ファイルの集合、クラスパスなどは引数で指定することもできます。タスクの具体的な使い方については、詳しく後述します。

■依存関係の記述

ビルドを行うときに、jarファイルの作成の前には、ソースのコンパイルが必要などの依存関係が発生しますが、makeもAntも依存関係を記述することができます。Makefileでは、

```
生成ファイル:中間ファイル
  コマンド2  # 中間ファイルから生成ファイルを生成するコマンド
  (コマンド2')  # 複数のコマンドを実行したい場合、改行して記述

中間ファイル:ソースファイル
  コマンド1  # ソースファイルから中間ファイルを生成するコマンド
  (コマンド2')  # 複数のコマンドを実行したい場合、改行して記述
```

という形式で、依存関係を記述します。makeは“:”より左側のファイルのタイムスタンプと右側のファイルのタイムスタンプを調べて、左より右のタイムスタンプが新しい場合、もしくは、左側のファイルが存在しない場合、コ

マンドを実行します。例えば、

```
$(JAR):$(BUILD_FILES)
  jar cvf  $(JAR) -C $(BUILD_DEST)/ .

$(BUILD_FILES):$(BUILD_SRC)
  mkdir -p $(BUILD_DEST)
  javac -d $(BUILD_DEST) $(BUILD_SRC)
```

に注目すると、src/foo1.java,src/pkg/foo2.java ($(BUILD_SRC))に対して、mkdir/javacが実行されます。生成されたクラスファイル classes/foo1.class classes/pkg/foo2.class ($(BUILD_FILE))に対してjarが実行されてfoo.jar ($(JAR))が生成されます。

　各ステップは、処理の必要がなければスキップされます。

注：“jar:$(JAR)”という行は、make jarコマンドでjarファイルを生成するための何も実行しないダミー行です。

　これに対して、Antは、ターゲットに対して依存関係を記述して定義します。

```
<target name="ターゲット1">
  タスク2
</target>

<target name="ターゲット2" depends="ターゲット1">
  タスク1
</target>
```

　この例は、Antがターゲット2を実行しようとすると、ターゲット1が実行済みかどうか調べて、

未実行の場合
　ターゲット1を実行した後、ターゲット2を実行

実行済みの場合
　ターゲット1を実行せずに、ターゲット2を実行

という動作をします。ここで注意が必要なのは、makeは、生成されたファイルとソースのタイムスタンプを調べて依存関係を処理しますが、ターゲットは「現在実行されているAntですでに処理されたかどうか」を調べ、ファ

イルのタイムスタンプを調べていません。つまり、Antを実行するごとに各ターゲットは実行されます。実際にファイルのタイムスタンプを調べてコマンドを実行するかどうかは、タスクが判別します。

Antの利点

さて、ここまでの話を読むと、一見して、「できることはMakefileと同じだが、XMLで記述しなければならない分、antの方が面倒なのでは？」と思われる方もいるかもしれません。ここまでの内容を整理すると、Antを使うと次のようなメリットがあります。

（1）ファイルの依存関係を記述する必要がなくなる

タスクが与えられた各ソースと生成されるファイルのタイムスタンプを調べて、処理が必要なソースだけを処理してくれます。Makefileのように1つ1つファイルを指定して依存関係を記述する必要はありません。

（2）処理対象となるファイルを個別に記述する必要はない

Makefileでは、SRC=foo1.java foo2.javaというように個別にファイルを指定する必要がありました。これに対してAntでは、srcdir="..."というようにディレクトリを指定すれば、そのディレクトリ以下のすべてのファイルを処理対象にしてくれます。Makefileでもfindを利用すれば、あるディレクトリ以下の*.javaにマッチするファイルなどをグループ化することができますが、それに比べても手間がかかりません。

ただし、上記のメリットは、Javaのアプリケーションのビルド時に受けられる恩恵で、C言語の開発などでは、taskからコマンド呼び出しを行うだけになってしまい、Antの恩恵は受けられないので注意してください。

コマンドを実行する「タスク」の使い方

次に、Antの要であるタスクの中で、特に重要と思われるjavac、jar、javadoc、war、echoについて紹介します。引数については、よく使われるものを抜粋して掲載しています。より詳細が知りたい場合はJakartaプロジェクトのタスク一覧を参照してください。日本語による情報が欲しい場合

は、Ja-Jakartaによる日本語訳をご覧ください。

●javacタスク

javacはその名のとおり、javacコマンドをタスクにしたもので、**表1**に示す引数を持ちます。

表1　javacタスクの引数

引数	機能	必須か？（デフォルト値）
srcdir	javaファイルの場所	YES
destdir	クラスファイルを出力する場所	YES
classpathref	あらかじめpathで設定したパスを指定	NO
encoding	Javaファイルのエンコーディング	NO（システムのエンコーディング）
excludes	コンパイル対象から除きたいファイルのリスト	NO
debug	デバッグ用にコンパイルするかどうか判断するフラグ	NO（false） ※true／falseを指定する

クラスパスにjarファイルをいくつか含めたい場合は、pathタスクと組み合わせて利用すると便利です。例えば、libディレクトリの下のjarファイルにすべてクラスパスを通し、サーブレットクラスのためにTomcatもしくはWebLogicのjarファイルにクラスパスを通してコンパイルを行うには、次のように記述します。

```
<!-- javacで利用するクラスパスの設定 -->
<path id="myclasspath">
  <!-- libディレクトリ以下のincludeで指定したファイルをmyclasspath
          に含める。-->
  <fileset dir="lib">
    <!-- 全てのjarファイルをクラスパスに含める -->
    <include name="*.jar"/>
  </fileset>

  <!-- 各アプリケーションサーバ固有の
       jarをmyclasspathに含める
       (特定のjarファイルを指定する場合は、pathelementを利用)-->
  <pathelement path="${catalina.home}/commons/lib/servlet.jar"/>
```

```
  <pathelement path="${weblogic.home}/server/lib/weblogic.jar"/>
</path>
..
<!-- 設定したmyclasspathを利用しコンパイル -->
<javac  srcdir="src/java"
        excludes="**/bak/*"
        destdir="build/classes"
        encoding="Windows-31J"
        classpathref="myclasspath" />
```

pathの設定は、javacを実行するターゲット内に含めることもできますが、初期設定という意味で、プロパティを定義した後に入れるとよいでしょう。

●jarタスク

jarタスクは、jarファイルを作成するタスクで、**表2**の引数を持ちます。

表2　jarタスクの引数

引数	機能	必須か？
basedir	jarに含めるファイルのルートディレクトリ	YES
destfile	jarファイル名	YES
manifest	マニフェストファイル	NO

例えば、マニフェストファイルにmanifestを指定してbuild/classes以下のファイルをjarに圧縮したい場合、次のように記述します。

```
<jar destfile="foo.jar"
basedir="build/classes"
manifest="manifest"/>
```

●warタスク

warタスクは、WebアプリケーションのWARアーカイブを作成します。warファイル自体、もともとjar形式のファイルであることもあり、jarファイルと同じですが、war用にいくつか引数が追加されています。ここでは、jarに追加された引数を**表3**に紹介します（実は、warタスクを実行するクラス自身がjarタスクのクラスを継承しています）。

表3　warタスクの引数

引数	機能	必須か？
webxml	Webアプリケーションの配備記述子META-INF/web.xmlファイルを指定	YES※
lib	WEB-INF/libディレクトリに含めるjarファイル。warタスクにネストしたタグとして含める	NO
classes	WEB-INF/classesディレクトリに含めるクラスファイルを指定。warタスクにネストしたタグとして含める	NO

※：updateオプション（既存のファイルを上書きする）をtrueに設定していれば、設定する必要はありません

Webアプリケーションの配備記述子にmyweb.xml、サーブレットなどのクラスファイルをclassesディレクトリ以下のファイルに指定し、libディレクトリ以下のjarファイルを含めるサンプルは次のようになります。

```
<war dest="sample.war"
     webxml="myweb.xml">
    <classes dir="classes" />
    <lib dir="lib/*.jar" />
</war>
```

classesとlibは、ネストしたタグに含める必要がある点に注意してください。

●javadocタスク

javadocを生成するタスクで、表4の引数を取ります。

表4　javadocタスクの引数

引数	機能	必須か？（デフォルト値）
packagenames	javadocを生成したいパッケージ名	YES
lib	WEB-INF/libディレクトリに含めるjarファイル。warタスクにネストしたタグとして含める	NO
	WEB-INF/classesディレクトリに含めるクラスファイルを	

classes	指定。warタスクにネストしたタグとして含める	NO
sourcepath	javaファイルの場所	YES
destdir	出力するディレクトリ	YES
Windowtitle	ブラウザのバーに表示するタイトルを指定	NO
Doctitle	概要に表示するタイトルを指定	NO
encoding	Javaファイルのエンコーディング	NO（システムのエンコーディング）
docencoding	出力するJavadocのエンコーディング	NO（システムのエンコーディング）
excludes	コンパイル対象から除きたいファイルのリスト	NO
bottom	HTMLファイルの一番下に挿入するテキストを指定	NO（著作権情報などを入れるのに利用）
Author	コード中の@authorタグで指定した開発者名を含める	NO（false）（true／falseを指定）

javadocは、次のように実行します。

```
<javadoc sourcepath="src/java
    excludes="**/bak/*"
    encoding="Windows-31J"
    packagenames="*"
    Windowtitle="my application"
    doctitle="my application"
    docencoding="iso-2022-jp"
    bottom="<center>&copy Arege Building</center>"
    Author="true"
    destdir="api"/>
```

●echoタスク

ビルド中にメッセージを表示することができます。echoは、次のように使います。

```
<echo message="This is echo message" />
```

ビルドファイルをデバッグする際にプロパティの値や実行経過を表示した

り、ビルド方法の表示などに利用すると便利です。

日本語利用時の注意点

Antを日本語環境で利用する際にいくつか注意すべき点があるので、少し述べておきます。

●ビルドファイルのエンコーディングを指定すること

ビルドファイルのxml宣言部で、ビルドファイルのエンコーディングを、

```
<?xml version="1.0" encoding="Windows-31J"?>
```

のようにencoding属性で指定する必要があります。この例では、Windows-31J（MS932）を指定していますが、UNIXなどをお使いの場合は、euc-jpなどに変えてください。文字コードが指定されていない場合、ビルドファイル読み込み時にエラーが発生します。ちなみにデフォルトエンコーディングはUTF-8ですので、UTF-8でビルドファイルを記述すれば、encoding属性を指定する必要はありません。

●javac、javadocタスクのエンコーディングを指定すること

ソースコードに日本語が含まれる場合は、javacタスクの中で次のように必ずソースコードのエンコーディングを指定するようにしてください。

```
<javac encoding="Windows-31J"
    …
```

この記述を行わなくても、システムデフォルトのエンコーディングでソースコードを記述していれば問題なくコンパイルできますが、Windows上で作成したコードをLinux上でコンパイルするなど、デフォルトのキャラクタセットが異なる環境でAntを実行すると、正しくコンパイルできなくなります。encodingは常に指定するようにしておきましょう。javadocタスクの場合は、次のようにソースのエンコーディング、出力HTMLのエンコーディング、ロケールを指定するようにします（ロケールを指定しないと英語環境で

javadocを利用すると、javadocが生成するHTMLが英語になるので注意してください）。

```
<javadoc
    encoding="Windows-31J"
    docencoding="iso-2022-jp"
    locale="ja"
    …
```

　この例では、ソースがWindows-31J、javadocで出力するHTMLファイルはiso-2022-jpとなります。また、そのほかのタスクでも、適宜エンコーディングを指定する癖をつけておいた方がよいでしょう。

COLUMN　ファイルセットとパターン

Antを利用する際に覚えておくと便利な機能にファイルセットとパターンがあります。ファイルセッの名のとおり「ファイルの集まり」を記述したものです。filesetは、javaのソース、classファイルなどのようなファイルでも扱うことができますが、特にクラスパスを設定するときjarファイルの集合をするのに便利です。例えば、次のようにlibディレクトリの下にコンパイル時に必要なjarファイルがあします。

```
myproject/
lib/
commons-logging.jar
oro.jar
servlet.jar
ext/j2ee.jar
```

このとき、

```
<!-- クラスパスの設定 -->
<path id="clspath">
<!-- libディレクトリ以下のincludeで指定したファイルをclspath
に含める。-->
<fileset dir="lib">
<!-- すべてのjarファイルをクラスパスに含める -->
<include name="*.jar"/>
</fileset>
</path>

…

<javac src="...">
<!--クラスパスにclspathを追加 -->
<classpath refid="clspath" />
</javac>
```

とすれば、lib以下の赤色のjarファイル（commons-logging.jar、oro.jar、servlet.ja
に対して、簡単にクラスパスに通すことができます。Antでは、パターンと呼ばれる正
現を簡単にしたものをfilesetタグの中で利用することができ、*は、0個以上の文字を表
パターンなので、*.jarによりカレントディレクトリ（ここではlib）中の拡張子がjarて
ファイルすべてを指定できることになります。また、

```
<fileset dir="lib">
<include name="**/*.jar">
</fileset>
```

と、**を用いると、カレントディレクトリ以下の拡張子がjarであるファイルを再帰的
し出し、すべてのjarを指定することができます。上記の例では、j2ee.jarを含んだ赤色
青色すべてのjarファイルを含めることができます。

次回は、応用編ということで、Antを使って、1つのJavaコードから簡単にEJBを作成する方法をご紹介します。

筆者プロフィール

岡本隆史（おかもと たかし）

岡山大学工学研究科修了後、（株）NTTデータに入社。文字認識ソフトウェアの研究開発を経て、W
ービス関連の研究開発に携わる。個人では、Debian GNU／Linuxの優れたメンテナンス性とほかのデ
トリビューションを圧倒するパッケージ数に引かれDebianを使い始めたのをきっかけに、Debianプロ
クトの開発者となる。DebianプロジェクトのStefan Gybas、Ola Lundqvistらとともに、Javaサポー
強化を行う。Jakartaに関しては、Tomcat／JMeter／ORO／Luceneなどの国際化／日本語へのロー
イズ、AntのKaffe VM対応などを行っている。『Jakartaプロジェクト徹底攻略』（技術評論社）、
『WEB+DB PRESS』（技術評論社）、『Java World』（IDGジャパン）、『JAVA Developer』（
トバンクパブリッシング）などで執筆活動を行っている。

Ja-Jakarta Projectについて

Ja-Jakartaプロジェクトでは、Jakartaプロジェクトのドキュメントの和訳やプロダクトの国際化／日
化などを行っている。現在、プロジェクトのメンバーを募集中。Ja-Jakartaプロジェクトの活動に参
ようという方は、「Ja-Jakartaプロジェクトへの参加方法」
（http://www.ingrid.org/jajakarta/site/getinvolved.html）を参照。

「次回」へ

「現場に活かすJakarta Project」バックナンバー

- Struts開発をIDEで効率化
- TomcatでWebDAVを実現
- JSP開発を効率化するTaglibs
- 効率的なログ出力をCommonsで実現
- java.langの機能を拡張するLang
- コレクションフレームワークを拡張するCollections
- DBのコネクションプールを簡単に実現する

類似記事の掲載をメールで通知

TechTargetジャパン

この記事を読んだ人にオススメの記事

Android SDK標準の何でもテストツールuiautomatorの基本的な使い方
ビルドツールGradleのインストールと使い方、Jenkins/Git連携
ブラックなWeb開発現場の救世主、Gruntのインストールと使い方
Androidアプリのビルド／テストはCIでここまで変わる

@IT Special