Java build system

by Zoltán Jakab

zoltan.jakab@ericsson.com

Contents

Hello world

```java
package demo;
public class HelloWorld {
   public static void main(String[] args) {
      if (args.length != 0) {
         throw new RuntimeException(
            "Unexpected number of arguments.");
      }
      System.out.println("Hello World!");
   }
}
```

Structure

```
<project_root>
|--build
|   `--build.sh
`--dev
    `--src
        `--demo
            `--HelloWorld.java
```

Build Java

```
# prepare environment
mkdir -p out/classes


# compile source
javac -d out/classes ../dev/src/**/*.java


# create archive with entrypoint
jar cfe demo.jar demo.HelloWorld -C out/classes .
```

```
> ./build.sh
> java -jar demo.jar
Hello World!
```

Use build system

- Generate source code
- Compile source code
- Create software package
  - Packaging with additional binaries (libraries, images, text files, …)
  - Edit metadata (build info, entry point)
- Validate the program
  - Execute test
  - Run code analyzer (FindBugs, PMD, …)

Contents

Apache Ant

- Java based build tool
- “In theory, it is kind of like make, without make's wrinkles”

- XML based
- Cross-platform
- Extendable

- Since 2000
- ant.apache.org

Structure

```
<project_root>
|--build
|  |--build.properties
|  `--build.xml
`--dev
   `--src
      `--demo
         `--HelloWorld.java
```

Build Java with Ant

```
<project name="demo" default="build">

   <target name="build">

      <mkdir dir="out/classes" />

      <javac srcdir="../dev/src"
             destdir="out/classes"
             includeantruntime="false" />

      <jar destfile="demo.jar" basedir="out/classes">

         <manifest>

            <attribute name="Main-Class"
                       value="demo.HelloWorld" />

         </manifest>

      </jar>

   </target>

</project>
```

Build Java with Ant

```
> ant
Buildfile: /ant-demo/build/build.xml


build:
    [mkdir] Created dir: /ant-demo/build/out/classes
    [javac] Compiling 1 source file to
                                /ant-demo/build/out/classes
    [jar] Building jar: /ant-demo/build/demo.jar


BUILD SUCCESSFUL
Total time: 0 seconds
```

Conventions

- File names
  - build.xml
  - build.properties
- Property names
  - lowercase.string.with.points

Run Ant script

- from the build folder

```
ant
ant <target_name(s)>
ant -D<property_name>=<value>
```

- from anywhere

```
ant -f <path>/build.xml
ant -f <path>/build.xml <target_name(s)> -D<property_name>=<value>
```

Execution tree

```xml
<project>
    <!-- common preconditions -->
    <target name="A" />
    <target name="B" />
    <target name="C" depends="B" />
    <target name="D" depends="A, C" />
    <target name="E" depends="B, D" />
</project>
```

```
> ant C
B -> C


> ant D
A -> B -> C -> D


> ant E
B -> A -> C -> D -> E
```

Execution tree

```xml
<project>
   <target name="A" />
   <target name="B" />
   <target name="C" depends="B">
      <antcall target="A" />
   </target>
</project>
```

```
> ant C
B -> C -> A
```

Contents

Properties

```
<property name="some.thing" value="thingthing" />
<property name="one.dir" location="this/is/a/path" />
<property file="build.properties" />
<property environment="env" />
```

- Write once
- Global visibility
- Organized in namespaces
- Everything is string
- Automatic type conversion

Properties

- Built-in – defined by Ant
  - basedir
  - ant.file
- System – accessible from Java
  - os.name
  - user.name

Properties - usage

```
<property name="first" value="build" />
<property name="other" value="${first} this" />
<echo>$${other}: ${other}</echo>
```

```
[echo] ${other}: build this
```

```
<property environment="env" />
<property name="module.root"
          value="${env.REPO_ROOT}/demo_module" />
```

Properties – visibility

```
<project>

    <property name="first" value="1" />

    <target name="foo">

        <property name="second" value="2" />

        <echo message=
"foo: ${first} ${second} ${third}" />

    </target>

    <target name="bar">

        <local name="third" />

        <property name="third" value="3" />

        <echo message=
"bar: ${first} ${second} ${third}" />

    </target>

</project>
```

```
> ant foo bar

foo:

    [echo] foo: 1 2 ${third}

bar:

    [echo] bar: 1 2 3




> ant bar foo

bar:

    [echo] bar: 1 ${second} 3

foo:

    [echo] foo: 1 2 ${third}
```

Properties file

build.properties

```
some.thing=thingthing
thread.count=4
# comments are allowed
source.dir=dev/src
base.package=${source.dir}/demo
lib.dir=${basedir}/lib
```

- Loaded in order
- Expansion based on namespace
- Only name-value pairs

Absolute vs relative path

```
<property name="classes.dir" value="out/classes" />


<property name="lib.dir" location="lib" />
<property name="other.dir" value="${basedir}/other" />
```

```
[echo] classes.dir: out/classes


[echo] lib.dir: /ant-demo/build/lib
[echo] other.dir: /ant-demo/build/other
```

Attributes

- Common attributes
  - id
  - taskname
  - description

- Boolean values

  true = yes = on

  false = no = off

Conditional execution

```
<project>
    <target name"check-file">
        <available property="file.exist" file="${target.file}" />
    </target>
    <target name="check-content" if="file.exist">
        <echo message="process target file" />
    </target>
    <target name="create-file" unless="file.exist">
        <echo message="create target file" />
    </target>
    <target name="build"
            depends="check-file, check-content, create-file" />
</project>
```

Conditions

```
<condition property="build.possible">
   <and>
      <not>
         <os family="mac" />
      </not>
      <available file="${my.lib}" />
   </and>
</condition>
```

- not, and, or, xor
- equals, isset, istrue, isfalse, contains, …
- available, uptodate, filesmatch, os, …

Resources

- "file-like" entities
- Access attributes
- Read/write content

resource, file, javaresource, javaconstant, url, zipentry, …

```
<copy todir="${config.dir}">

   <javaconstant name="com.mycomp.MyApi.DEFAULT_CONFIG_FILE"
                 classpath="${my.api.lib}" />

</copy>
```

Resource collections

- Base collections

  fileset, dirset, path, zipfileset, propertyset, …

- Set operations

  union, intersect, difference

- Selectors

  first, last, restrict, …

Path like structure

```
<path id="test.path">
   <pathelement location="${my.lib}" />
   <fileset dir="${external.lib.dir}">
      <include name="**/*.jar" />
      <exclude name="${junit.lib}" />
   </fileset>
   <path refid="test.lib.path" />
   <pathelement location="${classes.dir}" />
</path>
```

Test

```
<junit printsummary="on"
       fork="yes"
       forkmode="once"
       errorproperty="test.error">
   <classpath refid="${test.path}" />
   <formatter type="xml" />
   <batchtest todir="${report.dir}">
      <fileset dir="${classes.test.dir}">
         <exclude name="**/*$*.class" />
      </fileset>
   </batchtest>
</junit>
```

Test

```
> ant test

Buildfile: /ant-demo/build/build.xml

prepare:

     [mkdir] Created dir: /ant-demo/build/out/classes

     [mkdir] Created dir: /ant-demo/build/out/classes-test

     [mkdir] Created dir: /ant-demo/build/out/reports

build:

     [javac] Compiling 2 source files to /ant-demo/build/out/classes

test:

     [javac] Compiling 2 source files to /ant-demo/build/out/classes-test

     [junit] Running demo.TestMyClass

     [junit] Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.069 sec

     [junit] Running demo.TestMyUtility

     [junit] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.005 sec

BUILD SUCCESSFUL

Total time: 2 seconds
```

Contents

Reuse and extension

- Macro
- External build file
- Task library
- Custom task

Macro

```
<macrodef name="get-build-version">

    <attribute name="modulename" />

    <attribute name="property" />

    <sequential>

        <echo message=
"Fetching build version for @{modulename} to @{property}" />

        <exec executable="${db.script.dir}/db_access.py"
               outputproperty="@{property}"
               failonerror="true">

            <arg value="get-build-version" />

            <arg value="--module-name" />

            <arg value="@{modulename}" />

        </exec>

    </sequential>

</macrodef>
```

External build file

include	import
To rewrite targets	To override targets
Targets accessibility - name with prefix	Target accessibility - name - name with prefix
Rewritten dependencies	Preserved dependencies
Multiple times with different prefix	Only once

External build file

common.xml

```
<project name="util">
   <target name="setup">
      <property name="usage" value="common" />
   </target>
   <target name="print" depends="setup">
      <echo message="used from ${usage}" />
   </target>
</project>
```

External build file

include

```
<include file="common.xml"/>

<target name="a"
     depends="util.print" />




<include file="common.xml"
           as="common" />

<target name="a"
     depends="common.print" />
```

import

```
<import file="common.xml"/>

<target name="a"
     depends="util.print" />

<target name="b"
     depends="print" />


<import file="common.xml"
           as="common" />

<target name="a"
     depends="common.print" />

<target name="b"
     depends="print" />
```

External build file

include

```
<include file="common.xml"
           as="common" />
<target name="setup">
   <property name="usage"
              value="main"/>
</target>
<target name="main"
     depends="common.print" />
```

```
common.setup:
common.print:
   [echo] used from common
main:
```

import

```
<import file="common.xml"
           as="common" />
<target name="setup">
   <property name="usage"
              value="main"/>
</target>
<target name="main"
     depends="common.print" />
```

```
setup:
common.print:
   [echo] used from main
main:
```

Task library

- Ant-Contrib (ant-contrib.sourceforge.net)

if, for, assert, math, propertyregex, …

```
<taskdef resource="net/sf/antcontrib/antlib.xml"
         classpath="${ant.contrib.jar}" />
<if>
   <equals arg1="${result}" arg2="pass" />
   <then> <echo message="Finished successful!" /> </then>
   <else> <fail message="Something was wrong." /> </else>
</if>
```

Custom Task

```
package demo;
import org.apache.tools.ant.BuildException;
import org.apache.tools.ant.Task;

public class MyTask extends Task {
    @Override
    public void execute() throws BuildException {
        log("Hello World");
    }
}
```

Custom Task

```
public class MyTask extends Task {
    private int threadCount;

    public void setThreadCount(int threadCount) {
        this.threadCount = threadCount;
    }

    @Override public void execute() throws BuildException {
        if (threadCount <= 0) {
            throw new BuildException("Improper thread count.");
        }
...
```

Custom Task

```
...


        String osName = getProject().getProperty("os.name");


        String message = String.format(
            "Running %d thread on %s", threadCount, osName);
        log(message, LogLevel.INFO.getLevel());
    }
}
```

Custom Task – usage

```
<target name="do-my-task">

   <javac srcdir="${src.mytask.dir}"
          destdir="${classes.mytask.dir}"
          includeantruntime="true" />


   <taskdef name="myTask"
            classname="demo.MyTask"
            classpath="${classes.mytask.dir}" />


   <myTask threadCount="2" />

</target>
```

Contents

Other Java build tools

- Maven
  maven.apache.org
- Gradle
  gradle.org

Apache Maven

- Depends heavily on convention
  - Default directory structure
  - Default task list
- Repository concept
- Declarative

- Transitive dependency resolution
  - Auto download dependencies during build
- Central repo with local cache

Project Object Model

pom.xml

```xml
<project ...>
    <groupId>com.mycompany.app</groupId>
    <artifactId>demo</artifactId>
    <version>1.0</version>
    <packaging>jar</packaging>
    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.8.2</version>
            <scope>test</scope>
        </dependency>
    </dependencies>
</project>
```

Structure

```
demo
|-- pom.xml
`-- src
    |-- main
    |   `-- java
    |       `-- com
    |           `-- mycompany
    `-- test            `-- app
        `-- java            `-- App.java
            `-- com
                `-- mycompany
                    `-- app
                        `-- AppTest.java
```

Gradle

- Multi language support
  - Java, C++, Python
- Still relies on conventions
- Build files are Groovy DSL instead of XML
- Can read Ant
- Looks heavyweight

Gradle build file

build.gradle

```
apply plugin: 'java'
apply plugin: 'application'
dependencies {
   testCompile 'junit:junit:4.12'
}
mainClassName = 'App'
```

Thanks for listening & Questions

Zoltán Jakab – zoltan.jakab@ericsson.com