9.4 Workshop: Using Apache Ant and Ivy

Workshop: Ant and Ivy (258)

This workshop explains the use of two Apache projects for managing the delivery and deployment of software applications:

- Ant[9], a build and deployment management system; and
- Ivy[10] an extension to the ant system that provides dependency management system.

Both the Ant and Ivy systems are well documented and their are numerous tutorials available on the web if you need further help. The example project for this workshop is available in an archive called `demos-ant.zip` that accompanies these notes. In addition, most of the other workshops provided with these notes contain Ant and Ivy scripts which you can experiment with as well.

9.4.1 Using Ant Scripts

Ant has a command line user interface. The main command is `ant` which is issued with a number of space separated options followed by a list of targets to be invoked. The targets for a project are specified in an XML formatted file, called an *ant project file*, which by convention is called `build.xml` and stored in the project's root directory.

Start by opening a terminal in the `ant-demo` directory and listing the directory contents. You can see the default state of the project and the `build.xml` directory. There is also a file called `ivy.xml` that we will look at in the next section.

Invoking Ant (259)

```
%> ls
build.xml  config  ivy.xml  src
```

When the ant command is invoked, the project file is searched for matching targets which are then invoked in sequence listed. There is also a default target that just lists available targets. This is invoked by specifying the `-p` option as shown below.

```
%> ant -p
Buildfile: C:\Users\tws\Documents\teaching\2013\se-notes\
   demos-ant\build.xml

Main targets:

 build      Invokes the compile target.
 build-jar  Creates an executable jar of the project class
    files, excluding test cases.
 clean      Deletes all intermediate files.
```

[9] ant.apache.org
[10] ant.apache.org/ivy/

```
 cleanall   Deletes all files not included in the
    distribution archive.
 compile    Compiles all java class files.
 init       Initalises the distribution in preparation for
    compilation of code and documentation.
 install    Copies the project jar and dependencies to ${
    install.dir} and an executable script 'helloworld' to $
    {exec.dir} (both set on command line)
 resolve    Retrieves necessary dependencies for this
    project.
 run        Invokes the application from the default entry
    point.
 test       Invokes the project's suite of JUnit test cases
 uninstall  Removes project artifacts from the environment
Default target: build
```

We can see that many of these targets for the example project follow the conventions described above.

Let's start by resolving the dependencies for the project. We do this by invoking the `resolve` target, as shown below:

```
%> ant resolve
Buildfile: C:\Users\tws\Documents\teaching\2013\se-notes\demos-ant\build.xml

resolve:
[ivy:retrieve] :: Apache Ivy 2.3.0 - 20130110142753 :: http://ant.apache.org/ivy/ ::
[ivy:retrieve] :: loading settings :: url = jar:file:/C:/Program%20Files%20(x86)/apache-ant-1.8.4/lib/ivy-2.3.0.jar!/org/
    apache/ivy/core/settings/ivysettings.xml
[ivy:retrieve] :: resolving dependencies :: uk.ac.glasgow#senotes.demos.ant;working@misima
[ivy:retrieve]  confs: [default]
[ivy:retrieve]  found junit#junit;4.8.2 in public
[ivy:retrieve] downloading http://repo1.maven.org/maven2/junit/junit/4.8.2/junit-4.8.2.jar ...
[ivy:retrieve] ..............  (231kB)
[ivy:retrieve] .. (0kB)
[ivy:retrieve]  [SUCCESSFUL ] junit#junit;4.8.2!junit.jar (378ms)
[ivy:retrieve] :: resolution report :: resolve 1509ms :: artifacts dl 385ms
        ---------------------------------------------------------------------
        |                  |            modules            ||   artifacts   |
        |       conf       | number| search|dwnlded|evicted|| number|dwnlded|
        ---------------------------------------------------------------------
        |      default     |   1   |   1   |   1   |   0   ||   1   |   1   |
        ---------------------------------------------------------------------
[ivy:retrieve] :: retrieving :: uk.ac.glasgow#senotes.demos.ant
[ivy:retrieve]  confs: [default]
[ivy:retrieve]  1 artifacts copied, 0 already retrieved (231kB/18ms)

BUILD SUCCESSFUL
Total time: 5 seconds
```

The resolve target shows output from an ivy:retrieve command that downloads project dependencies (in this case JUnit) from a remote repository. By default, the dependencies are placed in the project's lib sub-directory.

Notice that the resolve target doesn't have any dependencies. Let's look at the compile target, which depends on the retrieve and init targets, before compiling the project's source files to byte code class files.

Compiling a project (260)

```
%> ant compile
Buildfile: C:\Users\tws\Documents\teaching\2013\se-notes\
    demos-ant\build.xml

init:
    [mkdir] Created dir: C:\Users\tws\Documents\teaching
        \2013\se-notes\demos-ant\bin
resolve:
compile:
    [javac] C:\Users\tws\Documents\teaching\2013\se-notes\
        demos-ant\build.xml:80: warning: 'includeantruntime'
         was not set, defaulting to build.sysclasspath=last;
         set to false for repeatable builds
    [javac] Compiling 2 source files to C:\Users\tws\
        Documents\teaching\2013\se-notes\demos-ant\bin

BUILD SUCCESSFUL
Total time: 5 seconds
```

The output from ant lists the sequence of targets invoked and the output from the tasks undertaken for each target. The full sequence is:

$$\texttt{init} \rightarrow \texttt{resolve} \rightarrow \texttt{compile}$$

The init command just creates a directory for storing compiled class files. The compile target compiles the java source code into class files stored in the bin directory. Notice that the resolve task does not re-download all the project dependencies. This is because Ivy can determine which of the locally cached dependencies needs to be updated (in this case none) because the remote copy has changed.

Now lets try running the test target as this is probably sensible before we try and run it or install it.

Testing the project (261)

```
%> ant test
Buildfile: C:\Users\tws\Documents\teaching\2013\se-notes\
    demos-ant\build.xml

init:

resolve:
compile:
test:
```

```
    [junit] Running uk.ac.glasgow.senotes.ant.test.
       HelloWorldTest
    [junit] Tests run: 1, Failures: 0, Errors: 0, Time
       elapsed: 0.052 sec

BUILD SUCCESSFUL
Total time: 3 seconds
```

The last lines of the output summarise the results of running the test case for the project (there is only one in the demo!) that thankfully passes. Notice that:

- The dependency sequence for `test` is the same as the full sequence for `compile`.This means we could do the compilation and test as a single step, if we'd wanted to.
- The tasks associated with the dependencies were not executed in full as they were the first time. This is because Ant has detected that the source files for these tasks have not changed since the target files were compiled or copied.

Running the project (262)

We're now ready to try running the project from its home directory.

```
%> ant run
Buildfile: C:\Users\tws\Documents\teaching\2013\se-notes\
   demos-ant\build.xml

init:

resolve:
compile:
run:
     [java] if it says [java] to the left, I was invoked
        from ant.

BUILD SUCCESSFUL
Total time: 3 seconds
```

The application is very simple: it just prints out a hard coded statement and exits. Notice again that the dependencies are invoked but that the tasks are not actually executed.

Finally, let's try running the installation target, notice that to do this, we need to specify where the application is to be installed. We can pass arguments to ant on the command line using the `-D` flag. For example, if we want to install the application on a Windows platform running cygwin, we could type:

Installing the project (263)

```
%> ant install -Dinstall.dir=/cygwin/usr/local/
   helloworldproject -Dexec.dir=/cygwin/bin
Buildfile: C:\Users\tws\Documents\teaching\2013\se-notes\
   demos-ant\build.xml
```

```
init:

resolve:
compile:
build-jar:
      [jar] Building jar: C:\Users\tws\Documents\teaching
          \2013\se-notes\demos-ant\helloworldproject.jar

install:
     [copy] Copying 2 files to C:\cygwin\usr\local\
         helloworldproject
     [copy] Copying 1 file to C:\cygwin\bin

BUILD SUCCESSFUL
Total time: 2 seconds
```

The application can now be executed directly from the environment, as shown.

Executing the installed application (264)

```
%> helloworld.sh
if it says [java] to the left, I was invoked from ant.
```

Sometimes a task won't execute properly. This can be because the ant script contains a defect, because the environment is not compatible with the project, or because a target hasn't been implemented, for example. Try running the `uninstall` target.

Failed targets (265)

```
%> ant uninstall
Buildfile: C:\Users\tws\Documents\teaching\2013\se-notes\
    demos-ant\build.xml

uninstall:

BUILD FAILED
C:\Users\tws\Documents\teaching\2013\se-notes\demos-ant\
    build.xml:169: Not implemented.

Total time: 0 seconds
```

The target fails because it contains an explicit `fail` task that automatically causes the task to fail and print out a message reporting that the target hasn't been implemented yet.

9.4.2 Creating Ant Scripts

Now lets take a look at creating new ant scripts. Ant scripts are XML files that follow a prescribed schema. The overall project is specified by a top level `project` tag as shown below.

Ant XML schema (266)

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
```

```
<project
 name="se-notes.workshop.ant"
 basedir="."
 xmlns:ivy="antlib:org.apache.ivy.ant"
 default="build"
>
 </target>
```

The project tag has a number of attributes that specify:

- the name of the project
- the default working directory for task
- where to find Ivy tasks that should be available to the project
- the default target to invoke if no target is specified on the command line

There are several different tags that can be specified as children of the top level `project` tag.

- `target` tags specify the details of individual targets, including the target dependencies and tasks to invoke.
- `property` tags specify variables that can be referenced elsewhere in the script.
- `path` tags specify collections of files and directories that can be referenced elsewhere in the system.

`property` and `path` tags (267)

Lets take a look at the `path` and `property` tags at the top of the example script.

```
<path id="project.classpath">

 <pathelement location="bin" />

 <fileset dir="lib" />

</path>


<property name="install.dir" value=""/>


<property name="exec.dir" value=""/>
```

The `path` has a single attribute `id` that can be used to refer to the path elsewhere in the project. The path tag contains two sub-tags. The `pathelement`

ant	echo	javadoc	replaceregexp
antcall	exec	javah	script
antlr	fail	jdepend	serverdeploy
apply	ftp	junit	sql
checksum	hostinfo	length	sshsession
chmod	image	mail	sync
chown	include	mkdir	tar
concat	input	move	touch
condition	jar	nice	war
copy	java	parallel	zip
delete	javac	replace	

Table 9.2: some built in Ant tasks

specifies a single member of the path (the `bin` sub-directory). The `fileset` tag specifies that every file in the `lib` directory should also be added to the specified path.

In addition, the project specifies two `property` tags that act as variables in the script. Currently, neither have been assigned a value.

All Ant targets are specified as children of the `project` tag using the `target` tag. In turn, the tasks to be executed by a target are specified as children of the target task. The name of the task is used for the name of the tag. For example, the task for creating a directory in the file system is called `mkdir`. The tasks are executed in the order in which they are specified in the target.

There are a large number of built in Ant tasks with extensive documentation[11]. Table 9.2 summarises the more frequently used tasks.

(Some) built in Ant tasks (268)

A complete list of tasks with full documentation can be found in the Ant user manual.[12]

Lets take a look at some of the targets and their accompanying tasks in the project Ant script, beginning with the `init` target, shown below.

The `init` target (269)

```
<target
 name="init"
 description="Initalises the distribution in preparation
    for compilation of code and documentation."
>

 <mkdir dir="bin"/>

</target>
```

The target is relatively simple, but it shows the typical attributes of a target tag.

[11] ant.apache.org/manual/tasklist.html
[12] ant.apache.org/manual/tasklist.html

- a unique name (mandatory);
- a description, that is printed when a list of targets is requested on the command line.

The `init` tag doesn't have any dependencies, so these aren't listed explicitly. The target does include a single task, making a directory for storing compiled class files called `bin`.

Now lets take a look at the compile target. This target has two dependencies, `init` that we've already seen and resolve, which is used to manage project dependencies (we'll look at how `resolve` works next).

The `compile` target (270)

```
<target
 name="compile"
 depends="init,resolve"
 description="Compiles all java class files."
>

 <javac
  srcdir="src"
  destdir="bin"
  debuglevel="lines,vars,source"
  classpathref="project.classpath"
 />

</target>
```

Assuming these dependencies complete successfully, the target has a single task, `javac` to execute. This task invokes a Java compiler on the project's source code (specified by the `srcdir` attribute) in order to generate compiled class files and stored them in the `bin` directory, as specified by the `destdir` attribute. The `classpathref` attribute is used to specify the path of jar files and directories that contain any necessary dependencies for the project.

Lets take a look at how the project dependencies are satisfied. The `resolve` target is shown below.

The `resolve` target (271)

```
<target
 name="resolve"
 description="Retrieves necessary dependencies for this
     project."
>

 <ivy:retrieve />

</target>
```

Like the `init` target it has no dependencies and a single task, `ivy:retrieve` which tells the Ivy system to download project dependencies as specified in the `ivy.xml` configuration file.

The `ivy.xml` file (272)

```
<ivy-module version="2.0">


 <info
  organisation="uk.ac.glasgow"
  module="senotes.workshop.ant"
 />


 <dependencies>

  <dependency
   org="junit"
   name="junit"
   conf="default"
   rev="4.8.2"/>

 </dependencies>


</ivy-module>
```

The file specifies some details about the project under management (`organisation` and `module`), and also lists the libraries that should be downloaded for the project. This project is only dependent on the `junit` project binaries. By default, Ivy searches the Maven repository[13] for the necessary binaries and downloads them into the `lib` directory.

Let's take a look at one more target, install. This is the most complex of the targets in the project.

The `install` target (273)

```
 <target
  name="install"
  description="Copies the project jar and dependencies to ${install.dir}
      and an executable script 'helloworld' to ${exec.dir} (both set on
      command line)"
  depends="build-jar"
 >
  <fail
   message="$${install.dir} and $${exec.dir} properties not set.">
   <condition>
    <or>
     <equals arg1="${install.dir}" arg2=""/>
     <equals arg1="${exec.dir}" arg2=""/>
    </or>
   </condition>
  </fail>

  <copy todir="${install.dir}">
   <fileset dir="lib/" includes="*.jar"/>
   <fileset dir="./" includes="*.jar"/>
  </copy>

  <copy todir="${exec.dir}" file="config/ant/helloworld.sh"/>
```

[13] search.maven.org

```
    <replace
     file="${exec.dir}/helloworld.sh"
     token="@INSTALLDIR"
     value="${install.dir}"
    />

  <chmod
   perm="ugo+rx"
   file="${exec.dir}/helloworld.sh"
  />

</target>
```

The first task is a `fail`, which checks whether the two properties needed for installation have been set properly by the user. If either property has been omitted the target fails.

The next steps copy the project dependencies and compiled jar file to the environments installation directory. After that the `helloworld.sh` script is copied to a directory for executables and configured. The configuration involves substituting a token `@INSTALLDIR` in the copied file with the directory of where the project's jar file was just copied. In addition, the scripts permissions are altered so that it can be executed directly.

9.4.3 Next Steps

- Recall that the `uninstall` target has not been implemented. What tasks does it need to implement? Remember that uninstall should remove the installation from the current system's environment, but it shouldn't affect the compiled distribution.
- Adapt the Ant script and Ivy configuration files provided to fit your own project. You may need to add additional dependencies to the Ivy file, for example, or alter some of the tasks.