

CHAPTER 3



Ant

Ant is becoming increasingly widely used as a build and deploy tool for Java applications. This chapter covers how I am using Ant in this book; the material covered has nothing in particular to do with Jini. Feel free to skip this chapter until you start building and deploying the examples from this book.

Applications consisting of multiple source files benefit from having a build tool to automate compilation, deployment, testing, and so on. Many of these tools are operating system-specific, such as `make` (although Windows versions now exist). Ant has become an increasingly popular tool for Java applications since it offers cross-platform support and Ant build files can be written in an operating system-independent way.

This book is adapted to use `ant` instead of `make` and Unix shell scripts. This chapter covers the use of `ant` for this book; it is not about Jini at all, so unless you want to see how the applications are currently built, I recommend that you skip this chapter. Individual build files for each project will be given in the relevant chapters.

Top-Level Build File

Two general parameters need to be set for your own environment:

- `jini.home`: The pathname to the location where Jini has been unpacked. This parameter is used to define the `jini.jars` variable that contains the standard Jini class files.
- `localhost`: The IP address or hostname of the current machine. In my testing, I run basic tests from this machine.

These parameters are defined in `build.xml` in the book's root directory on <http://jan.netcomp.monash.edu.au/java/jini/tutorial/>.

Similar to many projects, you adopt the following directory structure:

- `src`: The directory for all source files.
- `build`: The location where all class files are built.
- `dist`: The location where distribution files such as `.jar` files are created.
- `resources`: The location where things like policy files and configuration files are kept.
- `httpd.classes`: This is nonstandard, but it is the location where we need to copy files so that an HTTP server can find them.

These directories are all defined in the `build.xml` file in the tutorial's root directory. The following targets are defined:

- `compile`: Compile all source files.
- `dist`: Build the distribution, typically `.jar` files.
- `build`: Compile and distribute (redundant).
- `deploy`: Copy files to their destination, typically some `.jar` files to an HTTP server.
- `clean`: Remove all class files, `.jar` files and source backups.
- `run -DrunFile=...`: Run a project.
- `usage`: Print a list of options.

The top-level file `build.xml` defines these targets. The main function of each target is to run the target again in each of the projects. So `compile` runs `compile` in each project and `deploy` runs `deploy` in each project, whereas `run` calls `run` only in the selected project. The projects are each defined in an Ant file in the `antBuildFiles` directory. The `build.xml` file is as follows:

```
<project name="Jini book" default="usage" basedir=".">
  <!-- CONFIGURABLE STUFF HERE -->
  <property name="jini.home" value="/usr/local/jini2_1"/>
  <property name="localhost" value="dhcp-62-145.EECS.Berkeley.edu"/>
  <!-- END CONFIGURABLE STUFF -->
  <!-- Libraries -->
  <property name="jini.jars"
    value="${jini.home}/lib/jsk-platform.jar;${jini.home}/lib/jsk-lib.jar"/>
  <path id="compile.classpath">
    <pathelement path="${jini.jars}" />
    <pathelement path="build" />
  </path>
  <!-- Directories -->
  <property name="src" value="${basedir}\src"/>
  <property name="dist" value="${basedir}\dist"/>
  <property name="build" value="${basedir}\build"/>
  <property name="res" value="${basedir}/resources"/>
  <property name="httpd.classes" value="/home/httpd/html/classes"/>
  <!-- Show the usage options to the user -->
  <target name="usage" >
    <echo message=" compile"/>
    <echo message=" dist"/>
    <echo message=" build"/>
    <echo message=" deploy"/>
    <echo message=" clean"/>
    <echo message=" run -DrunFile='...' [-Dconfig='...']"/>
    <echo message=" usage"/>
  </target>
  <target name="all" depends="init,compile"/>

```

```
<!-- CLEAN -->
<target name="clean">
<!-- Delete our the ${build}, and ${dist} directory trees -->
<delete dir="${build}"/>
<delete dir="${dist}"/>
    <!-- delete all ~ backup files -->
    <delete>
        <fileset dir="." defaultexcludes="false" includes="**/*~"/>
    </delete>
    <!-- delete all .bak backup files -->
    <delete>
        <fileset dir="." defaultexcludes="false" includes="**/*.bak"/>
    </delete>
</target>
<target name="init">
<!-- Create the build directory structure used by compile N deploy -->
<mkdir dir="build"/>
<mkdir dir="dist"/>
</target>
<!-- call "compile" target in all build files in "antBuildFiles" dir -->
<target name="compile" depends="init">
    <subant target="compile" inheritall="true">
        <fileset dir="antBuildFiles"
            includes="*.xml"/>
    </subant>
</target>
<!-- call "dist" target in all build files in "antBuildFiles" dir -->
<target name="dist" depends="compile">
    <subant target="dist" inheritall="true">
        <fileset dir="antBuildFiles"
            includes="*.xml"/>
    </subant>
</target>
<!-- call "deploy" target in all build files in "antBuildFiles" dir -->
<target name="deploy" depends="dist">
    <subant target="deploy" inheritall="true">
        <fileset dir="antBuildFiles"
            includes="*.xml"
        />
    </subant>
</target>
<target name="build" depends="dist,compile"/>
<!-- call "run" on antfile determined by "runFile" property -->
<target name="run">
    <ant
        antfile="antBuildFiles/${runFile}.xml"
        target="run"/>
```

```
    </target>
</project>
```

Project Files

Each project is defined in an Ant file in the `antBuildFiles` directory. The purpose is to implement the top-level build targets for each project. Each of these project files inherits values from the top-level file, namely the following:

- `jini.home`
- `jini.jars`
- `src`
- `dist`
- `build`
- `httpd.classes`

Each project uses only a small number of the files from the `src` directory; these are defined in the `src.files` variable. For example, for the `complete.FileClassifierServer` project discussed in Chapter 9, the source files are defined as follows:

```
<property name="src.files"
    value="
        common/MIMEType.java,
        common/FileClassifier.java,
        complete/FileClassifierImpl.java,
        complete/FileClassifierServer.java
    "
/>
```

Since Jini is a distributed system, not all class files are required by all components. Typically, a server will require some files, whereas a client will require others. These are defined by two further variables:

```
<!-- Class files to run the server -->
<property name="class.files"
    value="
        common/MIMEType.class,
        common/FileClassifier.class,
        complete/FileClassifierImpl.class,
        complete/FileClassifierServer.class
    "
/>
<!-- Class files for the client to download --->
<property name="class.files.dl"
```

```
        value="
            complete/FileClassifierImpl.class
        "
    />
```

The rest of each project file is fairly straightforward. The `compile` target compiles all files in the `src.files` list; the `dist` target builds `.jar` files (usually two of them: one for the server and one for the client); the `deploy` target copies the `.jar` files for the client to an HTTP server; and the `run` target starts a JVM with appropriate parameters. Note that the JVM must be started as a separate VM, as it sets a security policy (discussed later), which cannot be done within an already running Ant JVM.

The complete project file for `complete.FileClassifierServer` is in the file `complete.FileClassifierServer.xml`:

```
<!--
    Project name must be the same as the filename which must
    be the same as the main.class. Builds jar files with the
    same name
-->

<project name="complete.FileClassifierServer">
    <!-- Inherits properties from ../build.xml:
        jini.home
        jini.jars
        src
        dist
        build
        httpd.classes
        localhost
    -->
    <!-- files for this project -->
    <!-- Source files for the server -->
    <property name="src.files"
        value="
            common/MIMType.java,
            common/FileClassifier.java,
            complete/FileClassifierImpl.java,
            complete/FileClassifierServer.java
        "/>
    <!-- Class files to run the server -->
    <property name="class.files"
        value="
            common/MIMType.class,
            common/FileClassifier.class,
            complete/FileClassifierImpl.class,
            complete/FileClassifierServer.class
        "/>
    <!-- Class files for the client to download -->
```

```

<property name="class.files.dl"
    value="
        common/MIMETypes.class,
        common/FileClassifier.class,
        complete/FileClassifierImpl.class
    "/>
<!-- Uncomment if no class files downloaded to the client -->
<!-- <property name="no-dl" value="true"/> -->
<!-- derived names - may be changed -->
<property name="jar.file"
    value="${ant.project.name}.jar"/>
<property name="jar.file.dl"
    value="${ant.project.name}-dl.jar"/>
<property name="main.class"
    value="${ant.project.name}"/>
<property name="codebase"
    value="http://${localhost}/classes/${jar.file.dl}"/>
<!-- targets -->
<target name="all" depends="compile"/>
<target name="compile">
<javac destdir="${build}" srcdir="${src}"
    classpath="${jini.jars}"
    includes="${src.files}">
</javac>
</target>
<target name="dist" depends="compile"
    description="generate the distribution">
<jar jarfile="${dist}/${jar.file}"
    basedir="${build}"
    includes="${class.files}"/>
<antcall target="dist-jar-dl"/>
</target>
<target name="dist-jar-dl" unless="no-dl">
<jar jarfile="${dist}/${jar.file.dl}"
    basedir="${build}"
    includes="${class.files.dl}"/>
</target>
<target name="build" depends="dist,compile"/>
<target name="run" depends="build,deploy">
<java classname="${main.class}"
    fork="true"
    classpath="${jini.jars}:${dist}/${jar.file}">
    <jvmarg value="-Djava.security.policy=${res}/policy.all"/>
    <jvmarg value="-Djava.rmi.server.codebase=${codebase}"/>
</java>
</target>
<target name="deploy" depends="dist" unless="no-dl">

```



```
        <copy file="${dist}/${jar.file.dll}"  
            todir="${httpd.classes}"/>  
    </target>  
</project>
```

Summary

Ant is now used in many Java projects to control the build and distribution process. This book also uses Ant, and this chapter has described how this is done. Please note that the material in this chapter is not essential to understanding how Jini works, though.



