Introduction to Software Engineering

Ant Tutorial

Rami Sayar @ramisayar

GitHub Talks If you find a mistake, don't hesitate to fork the repository, clone it to your local machine, edit the mistake, `git commit -a -m` and then press the pull request button to notify me.

Introduction

The purpose of these notes is to introduce you to building and compiling things with ant. Ant is a tool often used to build Java applications. Ant supplies built-in tasks to compile, assmble, test and run Java applications using a build file.

Project Preparation

Ideally, you want to generate class files to a different location than the same directory as your source files. Typically, you will have a `classes` or `objects` folder for compiled files, `jar` or `bin` for a jar or executable file and a `src` or `source` directory.

To do the above using the compiler directly can become complex and unweildy for large projects.

```
mkdir -p build/classes build/jar src
vim src/HelloWorld.java
javac -sourcepath src -d classes src/HelloWorld.java
java -cp classes HelloWorld
```

Ant to the Rescue!

If we have more files, more directories, dependencies on other jar files, a more complicated build process, etc… it becomes unnecessarily difficult to interface with the compiler directly, tedious, error prone and genuinely unpleasant.

Perhaps, you need a more complicated build process than the one described above. Perhaps, you need to download the source before compiled, prepare a build area by creating standard set of directories, configure the source code, validate it, compile it, test it, zip the executabe, build documentation, etc…

Build tools allow you to describe your build process, the dependencies between the steps, execute and manage the build, etc… without forcing every developer on a team to know the intimate details of the process. The build tool manages the build process for you and handles all of the details and steps. Ant provides you with a set of tools and techniques to define a build process and execute it.

Ant uses a build.xml file to describe your process and is typically placed at the root of your project directory. Ant uses xml to describe the build process, the following a simple example to do what we did above using the command line directly with the exception that we also added the ability to generate a build file.

NOTE: The example code is taken from [the official Apache Ant tutorial]

```
<project>

    <target name="clean">
        <delete dir="build"/>
    </target>

    <target name="compile">
        <mkdir dir="build/classes"/>
        <javac srcdir="src" destdir="build/classes"/>
    </target>

    <target name="jar">
        <mkdir dir="build/jar"/>
        <jar destfile="build/jar/HelloWorld.jar" basedir="build/classes">
            <manifest>
                <attribute name="Main-Class" value="HelloWorld"/>
            </manifest>
        </jar>
    </target>

    <target name="run">
        <java jar="build/jar/HelloWorld.jar" fork="true"/>
    </target>

</project>
```

Looking at the above build.xml file, we see a root node called project and several children called target. Each target has a name attribute which identifies what the target does. Inside each target you will see several nodes. The first node we can see is delete which simply deletes a directory or file. The second node is mkdir which simple creates a directory. The next node of interest is javac, this is very similar to the javac command line interface, we can set a src dir, a dest dir and more. The next node of interest is the jar node, which also has a child called manifest which allows us to set attributes to the jar file, such as a `Main-Class` . The last node of interest is the java node, which allows us to specify a jar to run with a fork (in another process).

To compile, build and run any of the targets above, you can execute the following in the root directory. `ant TARGET_NAME`

```
ant compile
ant jar
ant run
```

`ant compile jar run`

Let's enhance the above build process to include nomenclature and cleaning mechanism to remove previously build class files.

```
<project name="HelloWorld" basedir="." default="main">

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

    <property name="build.dir"   value="build"/>
    <property name="classes.dir" value="${build.dir}/classes"/>
    <property name="jar.dir"     value="${build.dir}/jar"/>

    <property name="main-class"  value="HelloWorld"/>



    <target name="clean">
        <delete dir="${build.dir}"/>
    </target>

    <target name="compile">
        <mkdir dir="${classes.dir}"/>
        <javac srcdir="${src.dir}" destdir="${classes.dir}"/>
    </target>

    <target name="jar" depends="compile">
        <mkdir dir="${jar.dir}"/>
        <jar destfile="${jar.dir}/${ant.project.name}.jar"
basedir="${classes.dir}">
            <manifest>
                <attribute name="Main-Class" value="${main-class}"/>
            </manifest>
        </jar>
    </target>

    <target name="run" depends="jar">
        <java jar="${jar.dir}/${ant.project.name}.jar" fork="true"/>
    </target>

    <target name="clean-build" depends="clean,jar"/>

    <target name="main" depends="clean,run"/>

</project>
```

The new nodes such as property allow us to set a key-value which we can then refer to in other places in the build process using the `${}` mechanism.

Managing Dependencies

Large java software projects will typically use many libraries and as a result have multiple dependencies. A common library used by java projects is Log4J. To integrate log4j into your project, you would download the latest jar and add it to a `lib` directory in you project root folder, you would then have to make that log4j is in your classpath for your project to compile. Handling dependencies is a perfect use case for build tools. Let's modify our above build.xml to add support for a `lib` directory and automatically adding jars to the class path.

```
<project name="HelloWorld" basedir="." default="main">
    ...
    <property name="lib.dir" value="lib"/>

    <path id="classpath">
        <fileset dir="${lib.dir}" includes="**/*.jar"/>
    </path>
    ...
    <target name="compile">
        <mkdir dir="${classes.dir}"/>
        <javac srcdir="${src.dir}" destdir="${classes.dir}"
classpathref="classpath"/>
    </target>

    <target name="run" depends="jar">
        <java fork="true" classname="${main-class}">
            <classpath>
                <path refid="classpath"/>
                <path location="${jar.dir}/${ant.project.name}.jar"/>
            </classpath>
        </java>
    </target>
    ...
</project>
```

Above you can see we specified a path node to easily referene paths in our build process, this is then used to build the classpath list in the run and compile mode.

Conclusion

The above notes should be enough to give you a test of Ant and to get you started with a simple build process for your project. Ant offers many built-in tools which can simplify your life. I highly recommend you take a look at the Ant documentation to learn more.

NOTE: You may have used other build tools for Java projects such as `Maven`. These are also wonderful tools which take a different approach than Ant. However, the project grader expects an Ant build file for the same of consistency.

References

Tutorial: Hello World with Apache Ant What is Ant?