Programación I
Grado en Ingeniería Informática
Curso 2015/2016

INTRODUCCIÓN A LA HERRAMIENTA ANT
Uso de Ant para la automatización de tareas

Tabla de contenidos

1. ¿Qué es Ant?

Ant es una herramienta del proyecto Jakarta de Apache que viene a ser un equivalente del make (mucho más moderno) para los desarrolladores Java (aunque en teoría nada impide usarlo con otros lenguajes). Ant permite automatizar tareas como actualizar el classpath, compilar el código, ejecutar código, pasar tests, copiar algunas clases compiladas a otro directorio, generar la documentación con *javadoc,…* Algunos IDEs hacen gran parte de estas cosas, pero no de una forma tan automatizada como se puede conseguir usando Ant.

La licencia de Ant permite obtener el código abierto. Entre otras ventajas que presenta están que usa XML como formato de sus ficheros. Ant es una herramienta escrita en Java, por tanto se trata de un software multiplataforma y además su uso está muy extendido, incluso los entornos de programación más conocidos como JBuilder o Eclipse tienen integrado el uso de Ant.

Para trabajar con Ant se necesitan tres cosas:

- Ant.
- JDK. (Ant es una aplicación Java)
- Un parser XML. Con la versión binaria de Ant ya viene incluido un parser XML.

2. Ventajas y "desventajas" de Ant respecto a otras herramientas

Se puede decir que la utilización de Ant aporta una serie de ventajas que facilitan, en cierta medida, la realización de actividades con respecto a otras herramientas de sus características y con finalidades similares.

La primera de ellas se mencionaba en el anterior punto, y es su fabricación en Java. Esto es algo que nos traslada automáticamente a asumir la portabilidad del trabajo realizado. Es decir, supóngase que se procede a la construcción del fichero `build.xml` (descrito en el apartado 4. del presente manual). Pues bien, podremos hacer un uso correcto del mismo sin necesidad de tener en cuenta sobre qué sistema operativo estamos trabajando (ya sea Windows, Linux o Mac). Esto es algo que nos facilita bastante la tarea con respecto a los ficheros de script, o a los comandos `.bat`, que nos obligan a tener una versión diferente del fichero en función del sistema operativo. Además, facilita bastante una de las labores que se pueden realizar en nuestro trabajo, como es (mencionado con posterioridad en este tutorial) la distribución.

No menos importante es otra de las ventajas proporcionadas por Ant, como es la determinación de dependencias, o lo que es lo mismo, el programador puede determinar un cierto orden de realización de las diversas tareas a ejecutar, imposibilitando la realización de una labor que haya de ser desarrollada después de una previa. Un ejemplo obvio de este hecho descrito se tendría en la necesidad de realizar la compilación anterior al empaquetado. Además, es de significación igualmente el hecho de no permitir la realización de una misma tarea en más de una ocasión, cuando esto es innecesario.

Si volvemos sobre la comparativa con los scripts, con Ant podemos ejecutar con simplemente escribir **ant**, en lugar del nombre del script correspondiente.

Y no se puede pasar por alto la disponibilidad de la mayoría de los entornos para proceder a la integración de Ant (véase, por ejemplo, *eclipse*). De esta forma, aquellas tareas que hayan sido definidas fundamentalmente para la línea de comando de ms-dos/shell de Unix, se encontrarán accesibles sin mayor dificultad desde el entorno de desarrollo en cuestión.

Sin embargo, no nos olvidamos del título de este apartado del tutorial, y también existen una serie de aspectos que, si bien son de un calado menor, hay que considerar apropiadamente para evitar hipotéticos problemas en un futuro.

Así, la elaboración de un fichero `build.xml` no es tan inmediata como pueda ser, en la mayoría de los casos, la creación de scripts a medida de las necesidades.

Pero, posiblemente, el mayor inconveniente de Ant hace referencia al consumo de recursos, precisamente por su elaboración en Java. Este consumo de recursos se aprecia fundamentalmente en el desarrollo de proyectos que pueden ser considerados de mediana/gran envergadura. Si el proyecto en cuestión se diseña de tal modo que permita la existencia de múltiples partes, en mayor o menor medida independientes, se necesitará un `build.xml` para cada una de estos cuerpos de código. Y si bien es posible hacer llamadas desde un `build.xml` a otro, también es cierto que la frecuencia con la que se obtienen en este punto mensajes de error del tipo **"**`Out Of Memory`**"** es bastante elevada.

En cualquier caso, parece claro que atendiendo a los puntos expuestos, la utilización de Ant aporta, en general, una serie de ventajas para el programador dignas de consideración.

3. Instalación de Ant

1. Descargar los binarios de http://jakarta.apache.org/ant/index.html, y descomprimirlos en un directorio.
2. Añadir a la variable de entorno PATH la ruta hasta el directorio `bin` de la instalación de Ant.
3. Añadir las librerías `.jar` del directorio lib de la instalación de Ant al CLASSPATH de nuestro entorno.
4. En algunos casos es necesario fijar las variables JAVA_HOME y ANT_HOME

De todas formas en la página oficial de descarga de Ant tenemos las instrucciones detalladas de instalación.

4. El fichero build.xml

En los ficheros XML se guarda la información de nuestro proyecto, normalmente el fichero se llama `build.xml`,
Veamos ahora con un pequeño ejemplo la estructura básica que tiene el fichero

```
<?xml version="1.0"?>

<project name="ProbandoAnt" default="compilar" basedir=".">
<!-- propiedades globales del proyecto -->
<property name="fuente" value="./src" />
<property name="destino" value="./classes" />
<target name="compilar">
      <javac srcdir="${fuente}" destdir="${destino}" />
</target>
</project>
```

Se declara el proyecto con el nombre "ProbandoAnt" y se indica la acción a realizar por defecto (`default="compilar"`). Se fija como directorio base el actual (*`basedir="."`*) .

Después se indica en sendas etiquetas *property* los directorios de origen y de destino (*`property name="fuente" value="./src"`*, y *`property name="destino" value="./classes"`*), y por último se declara un objetivo (etiqueta `target`) llamado *compilar*, que es el que se ha declarado por defecto. En este objetivo hay una única tarea, la de compilación `javac`, a la que por medio de los atributos `srcdir` y `destdir` se le indica los directorios fuente y destino, que se recogen de las propiedades anteriormente declaradas con *`${fuente}`* y *`${destino}`*. Se pueden tener varios objetivos en el mismo proyecto y cada uno de ellos realizar a su vez múltiples tareas.

Para probar este primer ejemplo se necesita, además del fichero `build.xml`, el fichero java `HolaAlumnos.java` que se encuentra en `fuentesHolaAlumnos.tgz`, dentro del directorio src:

```
public class HolaAlumos {
   public static void main(String[] args) {
      System.out.println("¡Hola alumnos!");
   }
}
```

Lo único que queda es ejecutar Ant con el fichero que se acaba de construir, así que simplemente se ejecuta el comando Ant con el build construido (desde una consola DOS o terminal GNU/Linux):

```
>ant -v
```

Por defecto, si no se especifica fichero xml intentará leer el fichero `build.xml`. Si no se especifica objetivo a ejecutar cogerá el que se haya declarado por defecto. Con la opción –v se activan las trazas de Ant. Se puede forzar la ejecución de un objetivo en caso de que existan varios:

```
>ant nombre_objetivo
```

El ejemplo de arriba hace que **javac** sea ejecutado, compilando todos los ficheros .java que hay en el directorio especificado por la propiedad *src.* El resultado son ficheros `.class` generados en el directorio especificado por la propiedad *destdir*, que será creado en caso de que no existiera.

Para probar que la clase ha sido generada con éxito basta con probar que la clase se ejecuta sin problema:

```
> java –cp classes/ HolaAlumnos
```

El comando

```
>ant --help
```

muestra el uso de la herramienta con todos sus posibles opciones.

5. Proyecto Estándar con Ant

En el apartado anterior se ha visto un ejemplo muy simple del uso de Ant, ahora se verá un proyecto más complejo y cómo el uso de Ant automatiza numerosas tareas que de lo contrario serían muy tediosas.

El ejemplo cubrirá la compilación, distribución, documentación y ejecución de una aplicación. Se mantendrá una estructura jerárquica de directorios, separados por ficheros fuente, clases generadas, ficheros distribuibles y documentación. En el directorio donde empieza esta jerarquía situaremos el fichero build.xml con el proyecto y algún otro fichero que se necesite.

Se tomarán como ficheros fuentes los de src2.zip y se descomprirán sobre el directorio src. La definición del fichero build.xml que se verá en adelante no coincide exactamente con el build necesario para tratar los fuentes de src2.zip, será labor del alumno modificar el build para generar todas las tareas necesarias.

5.1 Definición del proyecto

El directorio actual, donde se encuentra el fichero build.xml servirá como directorio base del proyecto. Ya se puede ir incluyendo algunas propiedades, que actuarán a modo de constantes, con los directorios que vamos a ir utilizando. Concretamente, se usará un directorio 'src' para guardar los ficheros fuente, un directorio 'lib' para guardar clases y ficheros .jar necesarios para compilar y ejecutar, un directorio 'classes' para guardar las clases Java compiladas, un directorio 'dist' donde se guardarán los ficheros a distribuir como aplicación final lista para ser utilizada, un directorio 'doc' para guardar la documentación generada con el javadoc y un directorio ‘etc’ para guardar otros ficheros.

Para indicar todo esto, utilizaremos un elemento `'project'` como el siguiente:

```
<project name="Ejemplo" basedir=".">
    <property name="src" value="src"/>
    <property name="lib" value="lib"/>
    <property name="classes" value="classes"/>
    <property name="dist" value="dist"/>
    <property name="doc" value="doc"/>
    <property name="conf" value="etc"/>
</project>
```

5.2 Compilación de los ficheros fuente

Para compilar los ficheros fuente se usará el objetivo 'compilar'. La tarea para compilar permite indicar el directorio de los fuentes, el directorio destino donde dejar las clases generadas y el classpath que se use, donde buscar clases y librerías necesarias. También puede ser interesante indicar los modos verbose y debug. Esta tarea será algo como:

```
<javac srcdir="${src}" destdir="${classes}" classpath="${lib}"
verbose="true" debug="true">
```

Es posible que interese guardar en algún fichero una indicación acerca de la compilación, como por ejemplo qué paquetes se están compilando. Para guardar en un fichero un sencillo mensaje se puede usar la siguiente tarea, a la que se le pasa el mensaje y el fichero a escribir:

```
<echo message="Compilacion terminada"  file="${conf}/log.txt"/>
```

El objetivo completo quedaría como:

```
<target name="compilar">
    <javac srcdir="${src}" destdir="${classes}" classpath="${lib}"
         verbose="true" debug="true"/>
    <echo message="Compilacion terminada"
         file="${conf}/log.txt"/>
</target>
```

Es necesario que los directorios que se vayan a utilizar de destino estén creados como por ejemplo el directorio destino de clases, el de distribución de los jar, el de la documentación javadoc,...

5.3 Generación de la documentación

Java incluye una utilidad llamada javadoc para generar documentación automáticamente a partir de los ficheros fuente. Para ello es necesario que el código se encuentre comentado bajo los estándares javadoc.

La tarea para documentar admite muchos parámetros aunque la mayoría son la extensión literal de los parámetros pasados a la herramienta javadoc. Básicamente, hay que indicar el directorio

donde buscar los ficheros fuente, el classpath donde buscar clases y librerías existentes, el directorio de destino y los paquetes a documentar. Además, hay que indicar otras opciones: establecer el modo verbose, indicar el nivel de acceso hasta el cual documentar (`access`), indicar que incluya las etiquetas author y version, indicar los títulos de la ventana (`windowtitle`) y de la lista de paquetes
(`doctitle`) y pedir que separe el índice por letras (`splitindex`). El objetivo queda más o menos como sigue:

```
<target name="documentar">
    <javadoc sourcepath="${src}" classpath="${lib}"
         destdir="${doc}"
         packagenames="${paquetes}"
         verbose="true" author="true"
         access="private" splitindex="true" version="true"
         windowtitle="${titulo}" doctitle="${titulo}"/>
</target>
```

Se utilizan dos propiedades, `'paquetes'` y `'titulo'` que podemos declarar como elementos dentro del elemento 'project', de la siguiente manera:

```
<property name="paquetes" value="es.unileon.prg.*"/>
<property name="titulo" value="Un ejemplo completo"/>
```

5.4 Distribución de la aplicación

Distribuir la aplicación básicamente consiste en generar un fichero .jar con las clases generadas, un fichero .zip con los ficheros fuente y un fichero 'LEEME.txt' adicional.

Al crear el fichero .jar deberemos indicar el nombre del fichero y el directorio base a partir del cual coger los ficheros a empaquetar. También es posible indicar el modo compress para comprimir la información empaquetada:

```
<jar jarfile="${dist}/ejemplo.jar" basedir="${classes}"
compress="true"/>
```

La creación del fichero .zip es muy similar. Sólo cambian los directorios a usar. Previamente, habremos copiado el fichero 'LEEME.txt' al directorio de los fuentes. La tarea resultante sería algo como:

```
<target name="distribuir">
    <jar destfile="${dist}/bar.jar" basedir="${classes}"
         compress="true"/>
    <copy file="LEEME.txt" todir="${src}"/>
    <zip zipfile="${dist}/bar_src.jar" basedir="${src}"
         compress="true"/>
</target>
```

5.5 Ejecución

Es útil tener una tarea para realizar alguna prueba con la aplicación ya compilada. Para ello se puede usar la tarea java, a la que se le indica la clase a ejecutar "`es.unileon.prg1.bar.PruebaMenuPrincipal`", cuál es el classpath (compuesto) donde buscar clases y librerías existentes y el parámetro que se le pasa (Pepe). Todo ello se puede hacer con un objetivo como:

```
<target name="probar" depends="compilar">
    <java classname="es.unileon.prg1.bar.PruebaMenuPrincipal">
         <classpath>
              <pathelement path="${classes}"/>
              <pathelement path="${lib}"/>
         </classpath>
         <arg value="Pepe"/>
</java>
</target>
```

Por primera vez se introduce una dependencia. El atributo `depends` obliga a Ant a ejecutar primero la tarea 'compilar'. Una vez terminada, se ejecutará la tarea 'probar'.

5.6 Objetivo global

Es útil tener un objetivo para ejecutar varias tareas, una detrás de otra. Concretamente, puede ser útil para compilar, distribuir y documentar la aplicación. Para ello se puede usar un objetivo como:

```
<target name="todo" depends="compilar, distribuir, documentar, probar"/>
```

Además, es posible indicar que por defecto, se ejecute esta tarea cuando se utilice este fichero build.xml sin indicar explícitamente un objetivo. Esto se consigue con el atributo `'default'` en el elemento `'project'`:

```
<project name="Ejemplo" basedir="." default="todo">
```

5.7 Limpieza

Es interesante mantener el entorno limpio de clases Java obsoletas, documentación inadecuada, etc. Al fin y al cabo, cuando se compila, ya no interesan las clases antiguas sino las nuevas. Ocurre lo mismo con la documentación generada o los ficheros .jar y .zip.

Se puede crear una tarea para 'limpiar' los directorios destino, y evitar que se junten demasiados archivos en desuso. Con el siguiente objetivo borramos los directorios cuyo contenido se genera dinámicamente:

```
<target name="limpiar">
     <delete dir="${classes}"/>
     <delete dir="${doc}"/>
     <delete dir="${dist}"/>
     <delete file="${conf}/log.txt"/>
</target>
```

Será necesario crearlos antes de guardar contenido en su interior, mediante las siguientes tareas, repartidas en los objetivos compilar, documentar y distribuir, respectivamente:

```
<mkdir dir="${classes}"/>
<mkdir dir="${conf}"/>
<mkdir dir="${doc}"/>
<mkdir dir="${dist}"/>
```

5.8 Proyecto completo

El fichero build.xml completo queda como sigue:

```
<project name="Ejemplo" basedir="." default="todo">

  <property name="src" value="src"/>
  <property name="lib" value="lib"/>
  <property name="classes" value="classes"/>
  <property name="dist" value="dist"/>
  <property name="doc" value="doc"/>
  <property name="conf" value="etc"/>
  <property name="paquetes" value="es.unileon.prg1.bar.*"/>
  <property name="titulo" value="Un ejemplo completo"/>

  <target name="limpiar">
    <delete dir="${classes}"/>
    <delete dir="${doc}"/>
    <delete dir="${dist}"/>
    <delete file="${conf}/log.txt"/>
  </target>

  <target name="compilar" depends="limpiar">
    <mkdir dir="${conf}"/>
    <mkdir dir="${classes}"/>
    <javac srcdir="${src}" destdir="${classes}" classpath="${lib}"
      verbose="true" debug="true">
    <echo message="Compilacion terminada"
     file="${conf}/log.txt"/>
  </target>

  <target name="documentar">
    <mkdir dir="${doc}"/>
    <javadoc sourcepath="${src}" classpath="${lib}"
         destdir="${doc}"
         packagenames="${paquetes}"
         verbose="true" author="true"
         access="private" splitindex="true" version="true"
         windowtitle="${titulo}" doctitle="${titulo}"/>
  </target>

  <target name="distribuir" depends="compilar">
    <mkdir dir="${dist}"/>
    <jar jarfile="${dist}/bar.jar" basedir="${classes}"
         compress="true"/>
    <copy file="LEEME.txt" todir="${src}"/>
    <zip zipfile="${dist}/bar_src.jar" basedir="${src}"
         compress="true"/>
  </target>
```

```
  <target name="probar" depends="compilar">
    <java classname="es.unileon.prg1.bar.PruebaMenuPrincipal">
         <classpath>
              <pathelement path="${classes}"/>
              <pathelement path="${lib}"/>
         </classpath>
         <arg value="Pepe"/>
</java>
  </target>

  <target name="todo" depends="compilar, distribuir, documentar, probar"/>

</project>
```

6. Conjuntos de ficheros

Un elemento `fileset` permite indicar un conjunto de ficheros. Los elementos `fileset` se utilizan en tareas donde sea necesario referirse a varios ficheros (ej.: los ficheros fuente en javac). Para determinar qué ficheros se incluyen en un conjunto de ficheros se pueden utilizar uno o más elementos `patternset`. También es posible definir el `patternset` a usar mediante los atributos `includes`, `includesfile`, `excludes` y `excludesfile`, que tienen el mismo significado que en `patternset`.

Se pueden utilizar los siguientes caracteres comodín:

- ? es usado para representar cualquier carácter
- * es usado para representar cero o más caracteres
- ** es usado para representar cero o más directorios

Un `FileSet` debe especificar un directorio base a partir del cual hace sus cálculos.

```
<fileset dir="BASEDIR">
</fileset>
```

Hay una serie de ficheros que por defecto no se incluyen en los `fileset`, éstos son los que cumplen las siguientes condiciones:

```
**/*~
**/#*#
**/.#*
**/%*%
**/._*
**/CVS
**/CVS/**
**/.cvsignore
**/SCCS
**/SCCS/**
**/vssver.scc
**/.svn
**/.svn/**
```

En su mayoría estos ficheros no se incluyen pues son ficheros de utilidades, backups, temporales que usan distintas herramientas como el CVS (Sistema de Control de Versiones). Si

no queremos excluir por defecto estos ficheros deberemos marcar la propiedad `defaultexcludes` a no.

Los patrones pueden ser encapsulados por el elemento `patternset`. Así otros elementos que hagan uso de patternset podrán referenciar al ya definido:

```
<fileset dir=".">
  <patternset id="miConjunto">
    <include name="*.txt"/>
    <exclude name="**/prueba*"/>
  </patternset>
</fileset>
```

7. Apéndice de tareas básicas

En la documentación oficial de Ant aparecen las tareas básicas con todos sus parámetros. Como tareas más representativas se pueden citar las siguientes:

- **javac:** para compilar ficheros fuente de Java.
- **java:** para ejecutar una aplicación Java.
- **javadoc:** para la generación de documentación a partir de los ficheros fuente.
- **copy:** para copiar ficheros y directorios.
- **delete:** para borrar ficheros y directorios.
- **mkdir:** para crear directorios.
- **move:** para mover ficheros o directorios.
- **chmod:** para cambiar los permisos de acceso en ficheros. Esta tarea sólo tiene efecto en sistemas de ficheros UNIX.
- **touch:** para cambiar la fecha y hora de modificación de un fichero.
- **echo:** para mostrar mensajes por la salida estándar o un fichero.
- **exec:** para ejecutar comandos de sistema.

8. Referencias

Sitio web del proyecto http://ant.apache.org/

Manual de usuario http://ant.apache.org/manual/index.html