8 Guía primeros pasos con Ant

8.1 ¿Qué es ant?

Una vez que tenemos hecho nuestro programa java, o mientras lo estamos haciendo, hay una serie de tareas repetetivas que hacemos con cierta frecuencia: borrar todos los .class para recompilar desde cero, compilar, generar la documentación de nuestro programa con javadoc, generar el jar con nuestro proyecto, etc, etc.

La forma normal de hacer esto al principio es hacerse unos ficheros de comandos en linux o ficheros .bat en windows. Ahí ponemos todos los comandos necesarios y llamamos a estos scripts cuando nos hace falta.

Ant es una herramienta gratuita (se puede bajar de http://ant.apache.org) que funciona similar a los scripts o ficheros .bat. En un fichero de texto en formato xml ponemos qué tareas queremos que se ejecuten. Este fichero habitualmente se llama **build.xml**. Luego, desde línea de comandos de una ventana de MS-DOS o Shell de Linux, nos basta ejecutar

ant

o bien

ant compila

ant documenta

y la tarea en cuestión se hace. "compila" y "documenta" son nombres que hemos definido nosotros para compilar y generar el javadoc respectivamente.

8.2 ¿Por qué usar ant?

Bueno, si ant hace lo mismo que los ficheros de script, ¿para qué usamos **ant**?. Realmente no hay ningún motivo, pero **ant** tiene unas pequeñas ventajas que nos pueden facilitar la vida.

La primera es que está hecho en java, así que **es portable**. Si hacemos nuestro fichero **build.xml** y tenemos instalado ant en Windows, en Linux o en iMac, funciona todo bien sin necesidad de tocar nada. Con los ficheros de script o de comandos .bat, tenemos que tener una versión de dicho fichero para cada sistema operativo. Es más, es posible incluso que en distintas distribuciones de Linux tengamos que hacer pequeñas modificaciones en el script. Esto lo hace un método ideal cuando distribuimos las

fuentes para que las gente los compile en sus casas. Nos ahorra hacer varios tipos de scripts para varios Sistemas Operativos. Podemos así, incluso distribuir el fichero **build.xml** con nuestras fuentes, de forma que si la persona que los va a usar tiene instalado **ant**, puede compilar y generar la documentación fácilmente.

La segunda es que nos permite definir fácilmente varias tareas a hacer y **fijar las dependencias** entre ellas (cuales deben hacerse antes que cuales). Cuando mandamos a **ant** que ejecute una tarea (por ejemplo, compilar), **ant** analizará según las dependencias que hayamos puesto, qué tareas debe realizar, las ordenará de más simple a más compleja y las ejecutará una a una en ese orden. No ejecutará ninguna tarea dos veces ni ejecutará una tarea antes de ejecutar las que le preceden según las dependencias que hemos indicado. Por ejemplo, está claro que para generar el jar, necesitamos previamente haber compilado. Nos bastará decir que la tarea "empaqueta" depende de "compila".

Otra ventaja, aunque parezca tonta, es que en todos los proyectos que hagamos, nos bastará con ejecutar "**ant**". No tendremos que andar pensando el nombre del script o si es un proyecto entre varias personas, ¿cómo demonios habrá llamado Felipe al script que genera el javadoc?

Y otra ventaja más es que casi todos **los entornos de desarrollo integran o tienen posibilidad de integrar ant** (eclipse por ejemplo). De esta forma, las tareas que hayamos definido para línea de comandos de MS-DOS/Shell de Linux, las tenemos accesibles directamente desde el entorno de desarrollo.

No todo son ventajas. Me he tropezado con dos pegas principalmente. La primera es que **hacer un script para compilado es bastante cómodo** (sobre todo en Unix), mientras que un fichero **build.xml** de estos me resulta más difícil y siempre tengo que andar copiando uno de otro proyecto. Suelo hacer scripts con cierta frecuencia para cosas que no son compilar java y por ello conozco esa forma de programar y me siento cómodo con ella. Los **build.xml** sólo los hago para compilar, una vez al principio del proyecto y no los vuelvo a usar. Creo que nunca pillaré soltura con ellos.

La segunda pega más gorda, es que **ant** está en java y **consume bastantes recursos**. Participo en proyectos grandes, de más de 5000 clases. Estos proyectos suelen estar subdivididos en subproyectos, cada uno con su propio **build.xml**. Teóricamente, es posible llamar desde un **build.xml** a otros, pero en proyectos de esta envergadura no lo he conseguido sin obtener errores de "*OutOfMemory*". Hay forma de cambiar la cantidad de memoria que usa **ant** y lo he hecho, gracias a ello si he conseguido compilar algunos de los subproyectos que de otra forma no lo hacían. Lo que no he podido es hacer un **build.xml** principal que vaya llamando a los demás sin que me de el dichoso

error. Mi solución final fue hacer los **build.xml** parciales y un script tradicional que se encargue de ir ejecutando los **build.xml** de los subproyectos y juntándolo todo luego.

8.3 Instalamos Ant

Para instalar **ant** sólo tenemos que ejecutar la siguiente instrucción desde el terminal:

sudo apt-get install ant

8.4 Un build.xml

Hemos comentado que hace tenemos que hacer un fichero **build.xml** en el que indicamos las tareas. Vamos a hacer uno sencillo con un par de cosas, por ejemplo "**compilar**", generar el jar con "**empaqueta**" y generar el javadoc con "**documenta**".

Empezamos escribiendo algo como esto con el notepad, el vi o el editor de texto que más nos guste.

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

<project name="Mi_Proyecto" default="todo">

</project>
```

La primera línea es más o menos obligatoria en cualquier fichero .xml para cualquier aplicación, no solo de **ant**.

En la siguiente línea simplemente indicamos que a nuestro proyecto (*project*) lo llamamos (*name*) "Mi_Proyecto", aunque por supuesto podemos darle el nombre que nos dé la gana y que la tarea que se ejecutará por defecto (*default*) se llama "todo" y que también podemos darle el nombre que queramos. De momento esa tarea "todo" no hemos definido qué hace.

La última línea indica que se cierra el proyecto y se acaba el fichero.

Vamos ahora a definir lo que hace la tarea "todo"

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

```
<project name="Mi_Proyecto" default="todo">
        <target name="todo" depends="empaqueta, documenta">
        </target>
</project>
```

Listo. Hemos metido un par de líneas para indicar la tarea (*target*).

Hemos llamado a la tarea (*name*) "todo" (igual que en el default de la línea anterior). Hemos indicado que para hacer esta tarea "todo" hay que hacer previamente (*depends*) las tareas "empaqueta" y "documenta" que todavía no hemos definido.

Luego, indicamos el fin de lo que hay que hacer en la tarea "todo" (*/target*)

Vamos con la tarea de "empaqueta"

```
<?xml version="1.0"?>
<project name="Mi_Proyecto" default="todo">
        <target name="todo" depends="empaqueta, documenta">
        </target>
        <target name="empaqueta" depends="compila">
                <jar destfile="../jar/fichero.jar" includes="../class/**/*.class"/>
        </target>
</project>
```

Nuevamente definimos una tarea de nombre "empaqueta", que debe ejecutar previamente una tarea "compila" todavía no definida.

Entre la apertura (*target*) y cierre (*/target*) de la tarea, ahora sí decimos como generar el jar. El jar que aparece en rojo es una tarea predefinida de ant, que genera ficheros jar. Le hemos indicado que el fichero a generar (*destfile*) se llamará **fichero.jar**, lo dejará en **../jar** y meterá dentro de él (*includes*) todos los ***.class** que encuentre desde el directorio **../class** hacia abajo.

En *includes* se admiten comodines para poner los ficheros. El doble asterisco indica todos los subdirectorios recursivamente a patir de ahí.

Vamos ahora a definir la tarea "compila"

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

<project name="Mi_Proyecto" default="todo">

        <target name="todo" depends="empaqueta, documenta">

        </target>

        <target name="empaqueta" depends="compila">

                <jar destfile="../jar/fichero.jar" includes="../class/**/*.class"/>

        </target>

        <target name="compila">

                <javac srcdir="." destdir="../class"/>

        </target>

</project>
```

Esta vez la tarea "compila" no depende de nadie. El javac que aparece dentro es una tarea predefinida de **ant** que compila ficheros java. Le hemos indicado que compile todas las fuentes que encuentre del directorio actual (srcdir) hacia abajo y que deje los ***.class** generados en el directorio **../class** (*destdir*).

Ahora sólo nos falta la tarea "documenta", vamos a ella

build.xml (final)

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

<project name="Mi_Proyecto" default="todo">

	<target name="todo" depends="empaqueta, documenta">

	</target>

	<target name="empaqueta" depends="compila">

		<jar destfile="../jar/fichero.jar" includes="../class/**/*.class"/>

	</target>

	<target name="compila">

		<javac srcdir="." destdir="../class"/>

	</target>
	<target name="documenta">

		<javadoc sourcepath="." destfile="../javadoc" packagenames="*"/>

	</target>

</project>
```

Esta tarea no depende de nadie. javadoc es una tarea predefinida de **ant** a la que le hemos indicado dónde encontrar las fuentes (*sourcepath*) y donde debe dejar la documentación de javadoc (*destfile*). Hay que indicarle además de qué paquetes (*packagenames*) queremos que haga el javadoc.

Metiendo este fichero **build.xml** en el directorio de las fuentes **src**, podemos desde MS-DOS o Shell de Linux, situados en ese directorio, ejecutar los siguientes comandos:

$ ant	Hace por defecto la tarea "todo", que a su vez hace "empaqueta" y "documenta". "empaqueta" hace previamente compila. Es decir, se hace todo.
$ ant empaqueta	Realiza la tarea "empaqueta", que debe "compila" previamente
$ ant compila	Sólo "compila"
$ ant documenta	Sólo "documenta"

Bueno, esto es lo básico para ir empezando con **ant**. El **build.xml** que hemos hecho es bastante genérico. Basta que nuestro proyecto tenga una estructura de directorios como la indicada y colocar el **build.xml** en **src**.

ant tiene muchísimas tareas por defecto, que además de las cosas propias de java incluyen copia y borrados de ficheros y directorios, empaquetado y desempaquetado de ficheros zip, tar, jar, comandos de CVS e incluso el envío de e-mail. En http://ant.apache.org/manual/coretasklist.html tienes una lista de las tareas soportadas por defecto por **ant**, así como todos los posibles parámetros para cada una de ellas.

Además de las tareas por defecto, existen otras que podemos bajarnos opcionalmente. Estas incluyen la ejecución de test de JUnit, cosas de EJBs, métricas, etc, etc. En http://ant.apache.org/manual/optionaltasklist.html tienes una lista de las tareas opcionales que nos ofrece apache. Si buscas en internet encontrarás otros sitios que también ofrecen sus propias tareas.