Introduction à `ant`

1 Qu'est ce que `ant` ?

Ant est un projet du groupe Apache-Jakarta. Il permet l'automatisation des différentes tâches d'élaboration d'un projet (compilation, exécution de tâches de pré et post-compilation, création d'archives `jar`, ...). Il est au java ce que `make` est au C ou au C++. L'objectif principal de ant est d'accélérer et de faciliter les tâches de compilation, distribution et déploiement des applications java.
La configuration de ant repose sur un fichier XML qui décrit les différentes tâches qui devront être exécutées par l'outil. Il porte par convention le nom de *build.xml*. Le fichier de configuration contient un ensemble de cibles (appellées *target*). Chaque cible contient une ou plusieurs tâches à réaliser et peut avoir une dépendance avec une ou plusieurs autres cibles.

Etudions un exemple pour clarifier les choses :

On veut automatiser la création d'une archive jar contenant les classes compilées d'un projet et sa javadoc. Il faut donc créer une tâche dédiée à la création et que l'on peut par exemple appeler `distribution`. Pour générer l'archive, il faut avoir compilé le projet et avoir généré sa javadoc. Cela correspond à deux autres tâches, `compile` et `doc`, qui seront totalement indépendantes de la première. La tâche `distribution` sera dépendante des tâches `compile` et `javadoc`, ce qui veut dire que `compile` et `javadoc` seront executées avant `compile`.

Cet exemple illustre un des concepts essentiels de la gestion de projet avec ant : il faut, comme lors de la conception orientée objet d'un projet, extraire les étapes atomiques de production du projet et tisser les liens logiques qui existent entre elles.

2 Premier fichier `build.xml`

Téléchargez l'archive `tp_ant.tar.gz` sur le portail.
Décompressez-la et entrez dans le répertoire `exemple_ant/`.

Ce projet est composé d'une seule classe, `ant.example.HelloWorld`.
Ouvrez le fichier `build.xml` à la racine du répertoire `exemple/` avec un éditeur "simple" (qui n'interprète pas le `XML` tels que `emacs`, `vi`, `kate`, ...).

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

<project name="example" default="compile" basedir=".">

  <property name="sources" value="src"/>
  <property name="classes" value="classes"/>

  <target name="initialization">
    <mkdir dir="${classes}"/>
  </target>

  <target name="compile" depends="initialization">
    <javac srcdir="${sources}" destdir="${classes}"/>
  </target>

</project>
```

Analysons ce fichier un peu plus en détails :

- On peut remarquer qu'il existe deux types de champs dans un fichier ant : des propriétés (`property`), qui correspondent à des variables locales au fichier, et des cibles (`target`) qui sont des tâches exécutables par ant. Il est possible d'accéder à une propriété que l'on a définie grâce à la syntaxe `${nomDeLaPropriete}`. Chaque cible est exécutable en ligne de commande grâce à :

  `ant nomDeLaCible`.

- `<project name="example" default="compile" basedir=".">`
  Cette ligne définit notre projet en lui donnant un nom, une tâche cible par défaut et un répertoire de base relatif à la position du fichier `build.xml`.

- `<property name="sources" value="src"/>`
  définit une *variable* `sources` qui pointe vers le répertoire contenant les sources à compiler. Cette variable est accessible dans le reste du fichier avec `${sources}`.

- `<target name="initialization">`
  définit une cible qui a pour nom "`initialization`" et qui crée un répertoire nommé `classes`.

- `<target name="compile" depends="initialization">`
  définit une cible qui a pour nom `compile` et qui est dépendante de la cible `initialization`, c'est à dire que l'exécution de la cible `compile` provoque l'exécution préalable de la cible `initialization`. Après l'initialisation, les sources contenues dans le répertoire `src` sont compilées et les fichiers `.class` sont stockés dans le répertoire `classes`.

Exercice 1 :

La syntaxe des principales commandes ant est disponible en annexe.

1.1 : Dans une console, tapez `ant initialization` dans le répertoire où vous avez décompressé les sources de l'exemple. Observez maintenant le contenu du répertoire.

1.2 : Supprimez le répertoire `classes` puis exécutez la commande `ant` sans argument. Observez le contenu du répertoire courant puis du répertoire `classes`.

1.3 : Supprimez le répertoire `classes` et son contenu puis changez la valeur de la variable `classes` dans le fichier `build.xml`. Observez le résultat.

1.4 : En utilisant la commande ant `delete`, écrivez une cible `clean` qui supprime le répertoire `classes`.

Testez la en vous inspirant de la question **1.1**.

1.5 : Modifiez maintenant le fichier `build.xml` de façon à ce que la cible `clean` soit exécutée à chaque appel de la cible `compile`. Re-testez le tout.

1.6 : En utilisant la commande ant `java`, écrivez une cible `test` qui compile le projet et qui exécute le programme de test `HelloWorld`. Modifiez également l'entête du fichier `build.xml` de façon à ce que cette cible soit exécutée par défaut.

3 Intégration de ant à Eclipse

Nous allons maintenant voir comment utiliser en même temps `ant` et `eclipse`.

Lancez `eclipse`. Choisissez `File -> New -> Java -> Java Project`. Donnez un nom au nouveau projet, puis cliquez sur `create project from existing file`. Choisissez le répertoire où vous avez décompressé l'exemple de la section précédente. Cliquez sur `Next` puis sur `Finish`. L'exemple de la section précédente est maintenant importé dans eclipse.
Vous pouvez maintenant voir le nouveau projet dans l'onglet `package explorer` sur la gauche. Remarquez que le fichier `build.xml` apparaît dans l'arborescence du projet. Double cliquez pour l'ouvrir. Le fichier est reconnu automatiquement comme un fichier `ant`.
Placez vous par exemple sur une nouvelle ligne en dessous de `<target name="initialization">` puis exécutez la combinaison `ctrl + espace` au clavier. Vous voyez apparaître la liste possible des balises `ant` autorisées à cet endroit.

Cliquez maintenant sur `Window -> Show view -> Ant`. Une nouvelle fenêtre nommée `Ant` apparaît dans votre espace de travail. Faites un clic droit dedans et sélectionnez `Add buildfiles...`. Ajoutez alors le fichier `build.xml` de l'exemple 1. Vous voyez apparaître la liste des tâches ant que vous avez définies auparavant dans l'exercice 1. Vous pouvez faire un clic droit sur chacune des tâches et les exécuter (`Run as -> Ant build`). Des raccourcis pour ces fonctionnalités sont disponibles sous forme d'icônes dans la fenêtre `Ant`.

4 Gestion de projet avec `ant`

Reprenez les sources modifiées de l'exemple 1 pour cet exercice.

Exercice 2 :

2.1 : Ajoutez une cible `build` à votre projet qui crée un jar du projet dans un répertoire `build` en utilisant la commande `jar`. Cette nouvelle cible doit évidemment être dépendante de la cible `compile` écrite à l'exercice **1**. Le nom du fichier jar et le nom du répertoire cible doivent être facilement paramétrables. N'oubliez pas de mettre à jour la cible `clean` pour qu'elle supprime le fichier `jar` en plus des classes.

2.2 : Pour que le fichier jar du projet soit exécutable, il faut lui ajouter un fichier `Manifest`. Modifiez la tâche `build` écrite en **2.1** pour que l'archive jar créée soit exécutable. Utilisez pour cela la commande `manifest` **dans** la commande `jar`. Vérifiez que votre `jar` est exécutable.

2.3 : Ajoutez une cible `doc` au projet. Cette cible doit générer automatiquement la javadoc du projet dans un répertoire `docs` dont le nom doit être facilement paramétrable. N'oubliez pas de mettre à jour la cible `clean` pour qu'elle supprime ce répertoire.

2.4 : En vous inspirant de l'exemple ci-dessous, créez une nouvelle cible `distribution` qui crée une archive `tar` contenant le répertoire `src`, le répertoire `rapport` et le fichier `build.xml`.

```
<tar destfile="test.tar">
  <fileset dir=".">
    <include name="unFichier.txt"/>
    <include name="unRepertoire/**"/>
  </fileset>
</tar>
```

5 Annexes

5.1 Syntaxe des tâches ant principales

<javac srcdir=*sourcesDirectory* destdir=*destinationDirectory* classpath=*classPath*/>
compile les sources contenues dans `srcdir` et range les classes compilées dans le répertoire `destdir`

<java classname=*value* classpath=*value*/>
exécute la classe `classname`

<jar jarfile=*jarFilename* basedir=*classesDirectory*></jar>
génère un fichier jar de nom `jarfile` à partir du répertoire `basedir`

<unjar src=*jarFile* dest=*destinationDirectory*/>
décompresse le fichier jar `src` vers le répertoire `dest`, ici on précise la `Main-Class`

<manifest><attribute name="Main-Class" value=*mainClassName*/></manifest>

permet de définir un fichier manifeste à l'interieur d'un environnement `jar`

<javadoc sourcepath=*sourceDirectory* destdir=*javaDocDirectory*></javadoc>

génère la javadoc des sources contenues dans le répertoire `sourcepath` et la range dans le répertoire `destdir`

<tar destfile=*tarFilename* basedir=*baseDirectory* excludes=*excludeList*/>

crée un fichier tar nommé `destfile` et contenant le répertoire `basedir`. `excludes` contient la liste des fichiers omis séparés par des virgules.

<delete dir=*value*/>
supprime le répertoire `dir` ainsi que l'**intégralité** de son contenu

<delete file=*value*/>
supprime le fichier `file`

<mkdir dir=*value*/>
crée le répertoire `dir`

5.2 Liens

http://jakarta.apache.org/ant
le site officiel de `ant`

http://ant.apache.org/manual/
le manuel de `ant`

http://ant.apache.org/manual/tasksoverview.html
la liste complète des tâches ant disponibles

http://help.eclipse.org/help31/topic/org.eclipse.platform.doc.user/gettingStarted/qs-80_ant.htm le tutoriel officiel sur l'intégration de `ant` à `eclipse`