Développons en Java v 2.00

85. Ant

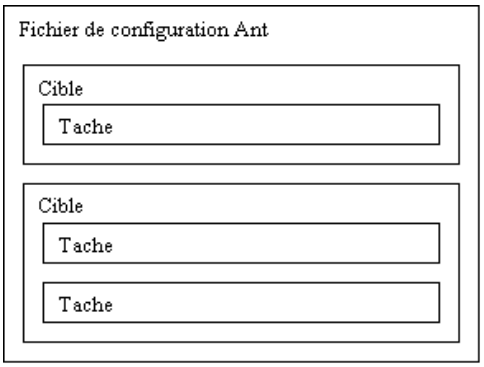
Niveau : Supérieur

Ant est un projet du groupe Apache-Jakarta. Son but est de fournir un outil écrit en Java pour permettre la construction d'applications (compilation, exécution de tâches post et pré compilation, ...). Ces processus de construction d'applications sont très importants car ils permettent d'automatiser des opérations répétitives tout au long du cycle de développement de l'application (développement, tests, recettes, mises en production, ...). Le site officiel de l'outil Ant est http://ant.apache.org/.

Ant pourrait être comparé au célèbre outil make sous Unix. Il a été développé pour fournir un outil de construction indépendant de toute plate-forme. Ceci est particulièrement utile pour des projets développés sur et pour plusieurs systèmes ou, pour migrer des projets d'un système sur un autre. Il est aussi très efficace pour de petits développements.

Ant repose sur un fichier de configuration XML qui décrit les différentes tâches qui devront être exécutées par l'outil. Ant fournit un certain nombre de tâches courantes qui sont codées sous forme d'objets développés en Java. Ces tâches sont donc indépendantes du système sur lequel elles seront exécutées. De plus, il est possible d'ajouter ses propres tâches en écrivant de nouveaux objets Java respectant certaines spécifications.

Le fichier de configuration contient un ensemble de cibles (target). Chaque cible contient une ou plusieurs tâches. Chaque cible peut avoir une dépendance envers une ou plusieurs autres cibles pour pouvoir être exécutée.



Les environnements de développement intégrés proposent souvent un outil de construction propriétaire qui est généralement moins souple et moins puissant que Ant. Ainsi des plugins ont été développés pour la majorité d'entre-eux (JBuilder, Forte, Visual Age, ...) pour leur permettre d'utiliser Ant, devenu un standard de fait.

Ant possède donc plusieurs atouts : multiplate-forme, configurable grâce à un fichier XML, open source et extensible.

Pour obtenir plus de détails sur l'utilisation de l'outil Ant, il est possible de consulter la documentation de la version courante à l'url suivante : http://ant.apache.org/manual/index.html

Une version 2 de l'outil Ant est en cours de développement.

Ce chapitre contient plusieurs sections :

- L'installation de l'outil Ant
- L'exécution de l'outil Ant
- Le fichier build.xml
- Les tâches (task)

85.1. L'installation de l'outil Ant

Pour pouvoir utiliser Ant, il faut avoir un JDK 1.1 ou supérieur et installer Ant sur la machine.

85.1.1. L'installation sous Windows

Le plus simple est de télécharger la distribution binaire de l'outil Ant pour Windows : jakarta-ant-version-bin.zip sur le site de Ant.

Il suffit ensuite de :

- décompresser le fichier (un répertoire jakarta-ant-version contenant l'outil et sa documentation est créé)
- ajouter le chemin complet du répertoire bin de l'outil Ant à la variable système PATH (pour pouvoir facilement appeler Ant n'importe où dans l'arborescence du système)
- s'assurer que la variable JAVA_HOME pointe sur le répertoire contenant le JDK
- créer une variable d'environnement ANT_HOME qui pointe sur le répertoire jakarta-ant-version créé lors de la décompression du fichier
- il peut être nécessaire d'ajouter les fichiers .jar contenus dans le répertoire lib de l'outil Ant à la variable d'environnement CLASSPATH

Exemple de lignes contenues dans le fichier autoexec.bat :

```
1. ...
2. set JAVA_HOME=c:\jdk1.3 set
3. ANT_HOME=c:\java\ant
4. set PATH=%PATH%;%ANT_HOME%\bin
5. ...
```

85.2. L'exécution de l'outil Ant

Ant s'utilise en ligne de commandes avec la syntaxe suivante :

ant [options] [cible]

Par défaut, Ant recherche un fichier nommé build.xml dans le répertoire courant. Ant va alors exécuter la cible par défaut définie dans le projet de ce fichier build.xml.

Il est possible de préciser le nom du fichier de configuration en utilisant l'option -buildfile et en la faisant suivre du nom du fichier de configuration.

Exemple :

```
1. ant -buildfile monbuild.xml
```

Il est possible de préciser une cible à exécuter. Dans ce cas, Ant exécute les cibles dont dépend la cible précisée et exécute cette dernière.

Exemple : exécuter la cible clean et toutes les cibles dont elle dépend

```
1. ant clean
```

Ant possède plusieurs options dont voici les principales :

Option	Rôle
-quiet	fournit un minimum d'informations lors de l'exécution
-verbose	fournit un maximum d'informations lors de l'exécution

-version	affiche la version de l'outil ant
-projecthelp	affiche les cibles définies avec leurs descriptions
-buildfile	permet de préciser le nom du fichier de configuration
-Dnom=valeur	permet de définir une propriété dont le nom et la valeur sont séparés par un caractère =

85.3. Le fichier build.xml

Le fichier build est un fichier XML qui contient la description du processus de construction de l'application.

Comme tout document XML, le fichier débute par un prologue :

<?xml version="1.0">

L'élément principal de l'arborescence du document est le projet représenté par le tag <project> qui est donc le tag racine du document.

A l'intérieur du projet, il faut définir les éléments qui le composent :

- les cibles (targets) : des étapes du projet de construction
- les propriétés (properties) : des variables qui contiennent des valeurs utilisables par d'autres éléments (cibles ou tâches)
- les tâches (tasks) : des traitements unitaires à réaliser dans une cible donnée

Pour permettre l'exécution sur plusieurs plate-formes, les chemins de fichiers indiqués dans le fichier build.xml doivent utiliser le caractère slash '/' comme séparateur, et ce, même sous Windows qui utilise le caractère anti-slash '\'.

85.3.1. Le projet

Il est défini par le tag racine <project> dans le fichier build.

Ce tag possède plusieurs attributs :

- name : précise le nom du projet
- default : précise la cible par défaut à exécuter si aucune cible n'est précisée lors de l'exécution
- basedir : précise le répertoire qui servira de référence pour l'utilisation d'une localisation relative des autres répertoires.

Exemple :

```
1. <project name="mon projet" default="compile" basedir=".">
```

85.3.2. Les commentaires

Les commentaires sont inclus dans un tag <!-- -->.

Exemple :

```
1. <!-- Exemple de commentaires -->
```

85.3.3. Les propriétés

Le tag <property> permet de définir une propriété qui pourra être utilisée dans le projet : c'est souvent la définition d'un répertoire ou d'une variable qui sera utilisée par certaines tâches. Sa définition, en tant que propriété, permet de facilement changer sa valeur une seule fois même si la valeur de la propriété est utilisée plusieurs fois dans le projet.

Exemple :

```
1. <property name= "nom_appli" value= "monAppli"/>
```

Les propriétés sont immuables et peuvent être définies de deux manières :

- avec le tag <property>
- avec l'option -D sur la ligne de commandes lors de l'appel de la commande ant

Pour utiliser une propriété sur la ligne de commandes, il faut utiliser l'option -D immédiatement suivie du nom de la propriété puis du caractère =, suivi lui-même de la valeur, le tout sans espace.

Le tag <property> possède plusieurs attributs :

- name : définit le nom de la propriété
- value : définit la valeur de la propriété
- location : permet de définir un fichier avec son chemin absolu. Peut être utilisé à la place de l'attribut value
- file : permet de préciser le nom d'un fichier qui contient la définition d'un ensemble de propriétés. Ce fichier sera lu et les propriétés qu'il contient seront définies.

L'utilisation de l'attribut file est particulièrement utile car il permet de séparer la définition des propriétés du fichier build. Le changement d'un paramètre ne nécessite alors pas de modification dans le fichier xml build.

Exemple :

```
1. <property file="mesproprietes.properties" />
2. <property name="repSources" value="src" />
3. <property name="projet.nom" value="mon_projet" />
4. <property name="projet.version" value="0.0.10" />
```

L'ordre de définition des propriétés est très important : Ant gère cet ordre. La règle est la suivante : la première définition d'une propriété est prise en compte, les suivantes sont ignorées.

Ainsi, les propriétés définies par la ligne de commandes sont prioritaires par rapport à celles définies dans le fichier build. Il est aussi préférable de mettre le tag <property> contenant un attribut file avant les tags <property> définissant des variables.

Pour utiliser une propriété définie dans le fichier, il faut utiliser la syntaxe suivante :
${nom_propriete}

Exemple :

```
1. ${repSources}
```

Il existe aussi des propriétés prédéfinies par Ant et utilisables dans chaque fichier build :

Propriété	Rôle
basedir	chemin absolu du répertoire de travail (cette valeur est précisée dans l'attribut basedir du tag project)
ant.file	chemin absolu du fichier build en cours de traitement
ant.java.version	version de la JVM qui exécute ant
ant.project.name	nom du projet en cours d'utilisation

85.3.4. Les ensembles de fichiers

Le tag <fileset> permet de définir un ensemble de fichiers. Cet ensemble de fichiers sera utilisé dans une autre tâche. La définition d'un tel ensemble est réalisée grâce à des attributs du tag <fileset> :

Attribut	Rôle
dir	Définit le répertoire de départ de l'ensemble de fichiers
includes	Liste des fichiers à inclure
excludes	Liste des fichiers à exclure

L'expression **/ permet de désigner tous les sous-répertoires du répertoire défini dans l'attribut dir.

Exemple :

```
1. <fileset dir="src" includes="**/*.java">
```

85.3.5. Les ensembles de motifs

Le tag <patternset> permet de définir un ensemble de motifs pour sélectionner des fichiers.

La définition d'un tel ensemble est réalisée grâce à des attributs du tag <patternset> :

Attribut	Rôle
id	Définit un identifiant pour l'ensemble qui pourra ainsi être réutilisé
includes	Liste des fichiers à inclure
excludes	Liste des fichiers à exclure
refid	Demande la réutilisation d'un ensemble dont l'identifiant est fourni comme valeur

L'expression **/ permet de désigner tous les sous-répertoires du répertoire définit dans l'attribut dir. Le caractère ? représente un unique caractère quelconque et le caractère * représente zéro ou n caractères quelconques.

Exemple :

```
1. <fileset dir="src">
2.   <patternset id="source_code">
3.     <includes="**/*.java"/>
4.   </patternset>
5. </fileset>
```

85.3.6. Les listes de fichiers

Le tag <filelist> permet de définir une liste finie de fichiers. Chaque fichier est nommément ajouté dans la liste, séparé du suivant par une virgule. La définition d'un tel élément est réalisée grâce à des attributs du tag <filelist> :

Attribut	Rôle
id	Définit un identifiant pour la liste qui pourra ainsi être réutilisé
dir	Définit le répertoire de départ de la liste de fichiers
files	liste des fichiers séparés par des virgules
refid	Demande la réutilisation d'une liste dont l'identifiant est fourni comme valeur

Exemple :

```
1. <filelist dir="texte" files="fichier1.txt,fichier2.txt" />
```

85.3.7. Les éléments de chemins

Le tag <pathelement> permet de définir un élément qui sera ajouté à la variable classpath. La définition d'un tel élément est réalisée grâce à des attributs du tag <pathelement> :

Attribut	Rôle
location	Définit un chemin d'une ressource qui sera ajoutée
path	

Exemple :

```
1. <classpath>
2.   <pathelement location="bin/mabib.jar">
3.   <pathelement location="lib/">
4. </classpath>
```

Il est préférable, pour assurer une meilleure compatibilité entre plusieurs systèmes, d'utiliser des chemins relatifs au répertoire de base du projet.

85.3.8. Les cibles

Le tag <target> définit une cible. Une cible est un ensemble de tâches à réaliser dans un ordre précis. Cet ordre correspond à celui des tâches décrites dans la cible.

Le tag <target> possède plusieurs attributs :

- name : contient le nom de la cible. Cet attribut est obligatoire
- description : contient une brève description de la cible. Cet attribut est optionnel mais il est recommandé de l'utiliser car la plupart des IDE l'affichent lors de l'utilisation de l'outil ant
- if : permet de conditionner l'exécution à l'existence d'une propriété. Cet attribut est optionnel
- unless : permet de conditionner l'exécution à l'inexistence de la définition d'une propriété. Cet attribut est optionnel
- depends : permet de définir la liste des cibles dont dépend la cible. Cet attribut est optionnel

Il est possible de faire dépendre une cible d'une ou plusieurs autres cibles du projet. Lorsqu'une cible doit être exécutée, Ant s'assure que les cibles dont elle dépend ont été complètement exécutées préalablement. Une dépendance est définie grâce à l'attribut depends. Plusieurs cibles dépendantes peuvent être listées dans l'attribut depends. Dans ce cas, chaque cible doit être séparée de la suivante avec une virgule.

85.4. Les tâches (task)

Une tâche est une unité de traitements contenue dans une classe Java qui implémente l'interface org.apache.ant.Task. Dans le fichier de configuration, une tâche est un tag qui peut avoir des paramètres pour configurer le traitement à réaliser. Une tâche est obligatoirement incluse dans une cible.

Ant fournit en standard un certain nombre de tâches pour des traitements courants lors du développement en Java :

Catégorie	Nom de la tâche	Rôle
Tâches internes	echo	Afficher un message
	dependset	Définir des dépendances entre fichiers
	taskdef	Définir une tâche externe
	typedef	Définir un nouveau type de données
Gestion des propriétés	available	Définir une propriété si une ressource existe
	condition	Définir une propriété si une condition est vérifiée
	pathconvert	Définir une propriété avec la conversion d'un chemin de fichier spécifique à un OS
	property	Définir une propriété
	tstamp	Initialiser les propriétés DSTAMP, TSTAMP et TODAY avec la date et heure courante
	uptodate	Définir une propriété en comparant la date de modification de fichiers
tâches Java	java	Exécuter une application dans la JVM
	javac	Compiler des sources Java
	javadoc	Générer la documentation du code source
	rmic	Générer les classes stub et skeleton nécessaires à la technologie rmi
	signjar	Signer un fichier jar
Gestion des archives	ear	Créer une archive contenant une application J2EE
	gunzip	Décompresser une archive
	gzip	Compresser dans une archive
	jar	Créer une archive de type jar
	tar	Créer une archive de type tar
	unjar	Décompresser une archive de type jar
	untar	Décompresser une archive de type tar
	unwar	Décompresser une archive de type war
	unzip	Décompresser une archive de type zip
	war	Créer une archive de type war
	zip	Créer une archive de type zip
	apply	Exécuter une commande externe appliquée à un ensemble de fichiers

tâches diverses	cvs	Gérer les sources dans CVS
	cvspass	
	exec	Exécuter une commande externe
	genkey	Générer une clé dans un trousseau de clés
	get	Obtenir une ressource à partir d'une URL
	mail	Envoyer un courrier électronique
	replace	Remplacer une chaîne de caractères par une autre
	sql	Exécuter une requête SQL
	style	Appliquer une feuille de style XSLT à un fichier XML
Gestion des fichiers	chmod	Modifier les droits d'un fichier
	copy	Copier un fichier
	delete	Supprimer un fichier
	mkdir	Créer un répertoire
	move	Déplacer ou renommer un fichier
	touch	Modifier la date de modification du fichier avec la date courante
Gestion de l'exécution de l'outil Ant	ant	Exécuter un autre fichier de build
	antcall	Exécuter une cible
	fail	Stopper l'exécution de l'outil Ant
	parallel	Exécuter une tâche en parallèle
	record	Enregistrer les traitements de l'exécution dans un fichier journal
	sequential	Exécuter une tâche en mode séquentiel
	sleep	Faire une pause dans les traitements

Certaines de ces tâches seront détaillées dans les sections suivantes : pour une référence complète de ces tâches, il est nécessaire de consulter la documentation de l'outil Ant.

85.4.1. echo

La tâche <echo> permet d'écrire dans un fichier ou d'afficher un message ou des informations durant l'exécution des traitements.

Les données à utiliser peuvent être fournies dans un attribut dédié ou dans le corps du tag <echo>.

Cette tâche possède plusieurs attributs dont les principaux sont :

Attribut	Rôle
message	Message à afficher
file	Fichier dans lequel le message sera inséré
append	Booléen qui précise si le message est ajouté à la fin du fichier (true) ou si le fichier doit être écrasé avec le message fourni (false)

Exemple :

```
01. <?xml version="1.0" encoding="ISO-8859-1"?>
02. <project name="Test avec Ant" default="init" basedir=".">
03.   <!-- ================================================================= -->
04.   <!-- Initialisation -->
05.   <!-- ================================================================= -->
06.   <target name="init">
07.     <echo message="Debut des traitements" />
08.     <echo>
09.       Fin des traitements du projet ${ant.project.name}
10.     </echo>
11.     <echo file="${basedir}/log.txt" append="false" message="Debut des traitements" />
12.     <echo file="${basedir}/log.txt" append="true" >
13. Fin des traitements
14.     </echo>
15.   </target>
16. </project>
```

Résultat :

```
C:\java\test\testant>ant
Buildfile: build.xml

init:
     [echo] Debut des traitements
     [echo]
     [echo]       Fin des traitements du projet Test avec Ant
     [echo]

BUILD SUCCESSFUL
Total time: 2 seconds
C:\java\test\testant>type log.txt
Debut des traitements
Fin des traitements

C:\java\test\testant>
```

85.4.2. mkdir

La tâche <mkdir> permet de créer un répertoire avec éventuellement ses répertoires pères si ceux-ci n'existent pas.

Cette tâche possède un seul attribut:

Attribut	Rôle
dir	Précise le chemin et le nom du répertoire à créer

Exemple :

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<project name="Test avec Ant" default="init" basedir=".">
  <!-- ================================================================== -->
  <!-- Initialisation -->
  <!-- ================================================================== -->
  <target name="init">
    <mkdir dir="${basedir}/gen" />
  </target>
</project>
```

Résultat :

```
C:\java\test\testant>ant
Buildfile: build.xml

init:
    [mkdir] Created dir: C:\java\test\testant\gen

BUILD SUCCESSFUL
Total time: 2 seconds
C:\java\test\testant>
```

85.4.3. delete

La tâche <delete> permet de supprimer des fichiers ou des répertoires.

Cette tâche possède plusieurs attributs dont les principaux sont :

Attribut	Rôle
file	Permet de préciser le fichier à supprimer
dir	Permet de préciser le répertoire à supprimer
verbose	Booléen qui permet d'afficher la liste des éléments supprimés
quiet	Booléen qui permet de ne pas afficher les messages d'erreurs
includeEmptyDirs	Booléen qui permet de supprimer les répertoires vides

Exemple :

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
```

```xml
<project name="Test avec Ant" default="init" basedir=".">
  <!-- ================================================================== -->
  <!-- Initialisation -->
  <!-- ================================================================== -->
  <target name="init">
    <delete dir="${basedir}/gen" />
    <delete file="${basedir}/log.txt" />
    <delete>
      <fileset dir="${basedir}/bin" includes="**/*.class" />
    </delete>
  </target>
</project>
```

Résultat :

```
C:\java\test\testant>ant
Buildfile: build.xml

init:
   [delete] Deleting directory C:\java\test\testant\gen
   [delete] Deleting: C:\java\test\testant\log.txt

BUILD SUCCESSFUL
Total time: 2 seconds
C:\java\test\testant>
```

85.4.4. copy

La tâche <copy> permet de copier un ou plusieurs fichiers dans le cas où ils n'existent pas dans la cible ou s'ils sont plus récents dans la cible.

Cette tâche possède plusieurs attributs dont les principaux sont :

Attribut	Rôle
file	Désigne le fichier à copier
todir	Permet de préciser le répertoire cible dans lequel les fichiers seront copiés
overwrite	Booléen qui permet d'écraser les fichiers cibles s'ils sont plus récents (false par défaut)

L'ensemble des fichiers concernés par la copie doit être précisé avec un tag fils <fileset>.

Exemple :

```xml
<project name="utilisation de hbm2java" default="init" basedir=".">

  <!-- Definition des proprietes du projet -->
  <property name="projet.sources.dir"    value="src"/>
  <property name="projet.bin.dir"        value="bin"/>

  <!-- Initialisation des traitements -->
  <target name="init" description="Initialisation">
    <!-- Copie des fichiers de mapping et parametrage -->
    <copy todir="${projet.bin.dir}" >
      <fileset dir="${projet.sources.dir}" >
        <include name="**/*.properties"/>
        <include name="**/*.hbm.xml"/>
        <include name="**/*.cfg.xml"/>
      </fileset>
    </copy>
  </target>
</project>
```

Résultat :

```
C:\java\test\testhibernate>ant
Buildfile: build.xml

init:
     [copy] Copying 3 files to C:\java\test\testhibernate\bin

BUILD SUCCESSFUL
Total time: 3 seconds
```

85.4.5. tstamp

La tâche <tstamp> permet de définir trois propriétés :

- DSTAMP : initialisée avec la date du jour au format AAAMMJJ
- TSTAMP : initialisée avec l'heure actuelle sous la forme HHMM
- TODAY : initialisée avec la date du jour au format long

Cette tâche ne possède pas d'attribut.

Exemple :

```
01. <?xml version="1.0" encoding="ISO-8859-1"?>
02. <project name="Test avec Ant" default="init" basedir=".">
03.   <!-- ================================================================== -->
04.   <!-- Initialisation -->
05.   <!-- ================================================================== -->
06.   <target name="init">
07.     <tstamp/>
08.     <echo message="Nous sommes le ${TODAY}" />
09.     <echo message="DSTAMP = ${DSTAMP}" />
10.     <echo message="TSTAMP = ${TSTAMP}" />
11.   </target>
12. </project>
```

Résultat :

```
01. C:\java\test\testant>ant
02. Buildfile: build.xml
03.
04. init:
05.      [echo] Nous sommes le August 25 2004
06.      [echo] DSTAMP = 20040825
07.      [echo] TSTAMP = 1413
08.
09. BUILD SUCCESSFUL
10. Total time: 2 seconds
```

85.4.6. java

La tâche <java> permet de lancer une machine virtuelle pour exécuter une application compilée.

Cette tâche possède plusieurs attributs dont les principaux sont :

Attribut	Rôle
classname	nom pleinement qualifié de la classe à exécuter
jar	nom du fichier de l'application à exécuter
classpath	classpath pour l'exécution. Il est aussi possible d'utiliser un tag fils <classpath> pour le spécifier
classpathref	utilisation d'un classpath précédemment défini dans le fichier de build
fork	lancer l'exécution dans une JVM dédiée au lieu de celle où s'exécute Ant
output	enregistrer les sorties de la console dans un fichier

Le tag fils <arg> permet de fournir des paramètres à l'exécution.

Le tag fils <classpath> permet de définir le classpath à utiliser lors de l'exécution

Exemple :

```
01. <project name="testhibernate1" default="TestHibernate1" basedir=".">
02.
03.   <!-- Definition des proprietes du projet -->
04.   <property name="projet.sources.dir"    value="src"/>
05.   <property name="projet.bin.dir"        value="bin"/>
06.   <property name="projet.lib.dir"        value="lib"/>
07.
08.   <!-- Definition du classpath du projet -->
09.   <path id="projet.classpath">
10.     <fileset dir="${projet.lib.dir}">
11.       <include name="*.jar"/>
12.     </fileset>
13.     <pathelement location="${projet.bin.dir}" />
```

```
  </path>

  <!-- Execution de TestHibernate1 -->
  <target name="TestHibernate1" description="Execution de TestHibernate1" >
    <java classname="TestHibernate1" fork="true">
      <classpath refid="projet.classpath"/>
    </java>
  </target>
</project>
```

85.4.7. javac

La tâche <javac> permet la compilation de fichiers sources contenus dans une arborescence de répertoires.

Cette tâche possède plusieurs attributs dont les principaux sont :

Attribut	Rôle
srcdir	précise le répertoire racine de l'arborescence du répertoire contenant les sources
destdir	précise le répertoire où les résultats des compilations seront stockés
classpath	classpath pour l'exécution. Il est aussi possible d'utiliser un tag fils <classpath> pour le spécifier
classpathref	utilisation d'un classpath précédemment défini dans le fichier de build
nowarn	précise si les avertissements du compilateur doivent être affichés. La valeur par défaut est off
debug	précise si le compilateur doit inclure les informations de débogage dans les fichiers compilés. La valeur par défaut est off
optimize	précise si le compilateur doit optimiser le code compilé qui sera généré. La valeur par défaut est off
deprecation	précise si les avertissements du compilateur concernant l'usage d'éléments deprecated doivent être affichés. La valeur par défaut est off
target	précise la version de la plate-forme Java cible (1.1, 1.2, 1.3, 1.4, ...)
fork	lance la compilation dans une JVM dédiée au lieu de celle où s'exécute Ant. La valeur par défaut est false
failonerror	précise si les erreurs de compilations interrompent l'exécution du fichier de build. La valeur par défaut est true
source	version des sources Java : particulièrement utile pour Java 1.4 et 1.5 qui apportent des modifications à la grammaire du langage Java

Exemple :

```
<project name="compiltation des classes" default="compile" basedir=".">
  <!-- Definition des proprietes du projet -->
  <property name="projet.sources.dir" value="src"/>
  <property name="projet.bin.dir" value="bin"/>
  <property name="projet.lib.dir" value="lib"/>

  <!-- Definition du classpath du projet -->
  <path id="projet.classpath">
    <fileset dir="${projet.lib.dir}">
      <include name="*.jar"/>
    </fileset>
    <pathelement location="${projet.bin.dir}" />
  </path>

  <!-- Compilation des classes du projet -->
  <target name="compile" description="Compilation des classes">
    <javac srcdir="${projet.sources.dir}"
              destdir="${projet.bin.dir}"
              debug="on"
              optimize="off"
              deprecation="on">
      <classpath refid="projet.classpath"/>
    </javac>
  </target>
</project>
```

Résultat :

```
C:\java\test\testhibernate>antBuildfile: build.xmlcompile:
    [javac] Compiling 1 source file to C:\java\test\testhibernate\bin
    [javac] C:\java\test\testhibernate\src\TestHibernate1.java:9: cannot resolve symbol
    [javac] symbol : class configuration
    [javac] location: class TestHibernate1
    [javac] Configuration config = new configuration();
    [javac] ^
    [javac] 1 error

BUILD FAILED
```

```
file:C:/java/test/testhibernate/build.xml:22: Compile failed; see the compiler e
rror output for details.

Total time: 9 seconds
```

85.4.8. javadoc

La tâche <javadoc> permet de demander la génération de la documentation au format javadoc des classes incluses dans une arborescence de répertoires.

Cette tâche possède plusieurs attributs dont les principaux sont :

Attribut	Rôle
sourcepath	précise le répertoire de base qui contient les sources dont la documentation est à générer
destdir	précise le répertoire qui va contenir les fichiers de documentation générés

Exemple :

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<project name="Test avec Ant" default="javadoc" basedir=".">
  <!-- ================================================-->
  <!-- Génération de la documentation Javadoc          -->
  <!-- ================================================-->
  <target name="javadoc">
    <javadoc sourcepath="src"
                destdir="doc" >
      <fileset dir="src" defaultexcludes="yes">
        <include name="**" />
      </fileset>
    </javadoc>
  </target>
</project>
```

Résultat :

```
C:\java\test\testant>ant
Buildfile: build.xml

javadoc:
  [javadoc] Generating Javadoc
  [javadoc] Javadoc execution
  [javadoc] Loading source file C:\java\test\testant\src\MaClasse.java...
  [javadoc] Constructing Javadoc information...
  [javadoc] Standard Doclet version 1.4.2_02
  [javadoc] Building tree for all the packages and classes...
  [javadoc] Building index for all the packages and classes...
  [javadoc] Building index for all classes...

BUILD SUCCESSFUL
Total time: 9 seconds
```

85.4.9. jar

La tâche <jar> permet la création d'une archive de type jar.

Cette tâche possède plusieurs attributs dont les principaux sont :

Attribut	Rôle
jarfile	nom du fichier .jar à créer
basedir	précise de répertoire qui contient les éléments à ajouter dans l'archive
compress	précise si le contenu de l'archive doit être compressé ou non. La valeur par défaut est true
manifest	précise le fichier manifest qui sera utilisé dans l'archive

Exemple :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
  <project name="Test avec Ant" default="packaging" basedir=".">

```

```
  <!-- ================================================================== -->
  <!-- Génération de l'archive jar                                        -->
  <!-- ================================================================== -->
  <target name="packaging">
    <jar jarfile="test.jar" basedir="src" />
  </target>
</project>
```

Résultat :

```
C:\java\test\testant>ant
Buildfile: build.xml

packaging:
      [jar] Building jar: C:\java\test\testant\test.jar

BUILD SUCCESSFUL
Total time: 2 seconds
```

La suite de ce chapitre sera développée dans une version future de ce document