Ant

Ant je nástroj na sestavování projektů (aplikací) podobný programu make, který se používá u programů v C či C++. Program Ant je volně k dispozici (tzv. The Apache Software License) na http://ant.apache.org/. Program je napsaný v jazyce Java a používá se v mnoha projektech realizovaných v Javě (např. tomcat, JDOM, jBOSS,). Instalace na vámi používaný operační systém je popsána v manuálu, který je součástí distribuce.

V porovnání s programem make má Ant následující výhody:

- je nezávislý na platformě (pokud se nevyvolávají příkazy operačního systému, což na rozdíl od programu make není obvykle potřeba),
- má přehlednější syntaxi zapsanou v souboru s XML formátem,
- při sestavování aplikací v Javě je rychlejší než make,
- obsahuje specifické konstrukce používané pro sestavování aplikací v Javě (podpora jar souborů, podpora EJB, generování dokumentace javadoc, výstupy z testování pomocí JUnit...)

Ant se obvykle spouští z příkazové řádky, je integrován nebo je možné ho i integrovat do mnoha vývojových nástrojů, např.:

- JBuilder
- Visual Age
- NetBeans, Forte for Java
- JEdit

Jak Ant pracuje?

Každý projekt má vlastní sestavovací (build) soubor, pro který se obvykle používá jméno **build.xml**, neboť se tento název považuje za implicitní a tudíž se nemusí specifikovat na příkazové řádce (parametr -buildfile *soubor*).

V sestavovacím souboru je projekt reprezentován kořenovým elementem ***project***. Tento element může být popsán následujícími atributy:

atribut	**popis**	**povinnost**
name	jméno projektu	ne
default	název defaultního cíle, který se bude provádět, pokud na příkazovém řádku při spouštění neuvedete jiný.	ano
basedir	kořenový adresář projektu	ne

tabulka 8 Atributy elementu project

Volitelně můžete přidat ještě element ***description***, do kterého lze zapsat podrobnější informace o účelu projektu. Tyto informace se vypíší do výstupu v případě použití přepínače -projecthelp při spouštění Antu.

V každém projektu je definován jeden nebo více **cílů (targets)**. Každý cíl představuje úlohu, která má být spuštěna (např. překlad nebo vygenerování dokumentace). Při spouštění Antu můžete říci, kterého cíle chcete dosáhnout. Pokud žádný neuvedete použije se implicitní nastavení cíle z popisu projektu.

U každého cíle může být specifikováno, na kterých cílech je závislý (např. vytvoření distribučního archivu bude závislé na překladu zdrojového kódu).

Každý cíl se provádí maximálně jednou za jedno vyvolání příkazu Ant. V případě, že vznikne chyba, přeruší Ant svoji činnost a nepokračuje s dalšími úlohami a cíli.

Element ***target*** má následující atributy.

atribut	**popis**	**povinnost**
name	jméno (název) cíle	ano
depends	seznam cílů, na kterých je tento cíl závislý	ne
if	jméno parametru (property), který musí být nastaven, aby byl cíl prováděn	ne
unless	jméno parametru (property), který nesmí být nastaven, aby se cíl prováděl	ne
description	krátký popis činnosti cíle	ne

tabulka 9 Atributy elementu target

Každý cíl se skládá z jednotlivých **úloh (task)**, které se provádějí sekvenčně za sebou. Některé úlohy se provádějí pouze v případě, že je to potřeba – např. zdrojový soubor se překládá pouze v případě, že je novější než odpovídající soubor .class.

Každá úloha může mít několik atributů či vnořených elementů v závislosti na její povaze. Ant obsahuje mnoho připravených úloh a je možné přidávat další nebo vytvářet svoje vlastní.

V projektu můžeme používat množinu **parametrů (properties)**. Parametry mohou být nastavovány v sestavovacím souboru nebo zvenku. Každý parametr má jméno a hodnotu. Parametry mohou být používány v jednotlivých atributech úloh, jméno parametru uvádíme mezi znaky **${** a**}**. V průběhu zpracování úlohy bude jméno parametru nahrazeno jeho hodnotou.

Jednoduchý příklad

V tomto příkladu předpokládáme následující adresářovou strukturu:

adresář	**obsah adresáře**
kořenový adresář projektu	soubor build.xml a obvykle soubor README
- adresář **src**	**v tomto adresáři a podadresářích (dle balíků, packages) jsou zdrojové texty jednotlivých tříd,**
- adresář **lib**	v tomto adresáři jsou knihovny .jar používané v projektu,
- adresář **classes**	přeložené třídy (soubory .class),
- adresář **dist**	do tohoto adresáře se vytváří distribuce (jar soubor),
- adresář **docs**	adresář s dokumentací, obvykle obsahuje podadresář api s dokumentací vygenerované pomoci javadoc,

tabulka 10 Přehled adresářů ukázkového příkladu

Ukázka souboru **build.xml,** ve kterém jsou definovány cíle init, compile, javadoc, dist, run a clear, následuje:

```
<project name="CestovniKancelar" default="dist" basedir=".">

<!-- set global properties for this build -->
<property name="src" value="src"/>
<property name="build" value="classes"/>
<property name="dist" value="dist"/>
<property name="docDir" value="docs"/>
```

```
<target name="init"
    description=”Create the build directory structure”>
    <mkdir dir="${build}"/>
</target>

<target name="compile" depends="init"
    description=”Compile the java code from ${src} into ${build}”>
    <javac srcdir="${src}"
        destdir="${build}"/>
</target>

<target name="javadoc" depends="compile"
    description="Generate javadoc from all .java files">
    <delete dir="${docDir}/api"/>
    <mkdir dir="${docDir}/api"/>
    <javadoc sourcepath="${src}"
        destdir="${docDir}/api"
        classpath="${build}"
        sourcefiles="${src}/*.java"/>
</target>

<target name="dist" depends="compile"
    description=”Create the distribution jar file”>
    <mkdir dir="${dist}/lib"/>
    <jar jarfile="${dist}/lib/CestovniKancelar.jar" >
        <fileset dir="${build}"/>
    </jar>
</target>

<target name="run" depends="dist"
    description=”Run CestovniKancelar application”>
    <java classpath="${dist}/lib/CestovniKancelar.jar"
        classname="CestovniKancelar"
        fork="yes"/>
</target>

<target name="clean"
    description = "Delete the ${build} and ${dist} directory trees">
    <delete dir="${build}"/>
    <delete dir="${dist}"/>
    <delete dir="${docDir}/api"/>
</target>

</project>
```

Vytvořili jsme tedy popis projektu, který se jmenuje CestovniKancelar a defaultním cílem je cíl dist. Jako výchozí adresář je nastaven aktuální adresář. V další části popisu projektu jsou nastaveny parametry pro projekt (jedná se o jména podadresářů). Dále jsou definovány jednotlivé cíle.

První cíl se jmenuje init a spustí úlohu mkdir, která vytvoří adresář classes.

Druhý cíl se jmenuje compile, je závislý na cíli init a spouští úlohu javac, která přeloží kód z adresáře src a uloží class soubory do adresáře classes.

Třetí cíl se jmenuje javadoc a je závislý na cíli compile. V rámci tohoto cíle se spouští několik úloh. Jako první úloha delete, která smaže starší verzi dokumentace, pokud existuje. Další úloha mkdir vytvoří adresář docs/api. Poslední úloha tohoto cíle je samotné vytvoření dokumentace prostřednictvím úlohy javadoc. Pro javadoc je třeba nastavit několik parametrů, zadat které zdrojové texty mají být zpracovány do dokumentace, kam má být dokumentace uložena a kde jsou class soubory těchto tříd.

Čtvrtý cíl se jmenuje dist, je závislý na cíli compile, a spouští dvě úlohy. První je mkdir, která vytvoří adresář dist/lib a druhá úloha jar do tohoto adresáře uloží vytvořený jar archiv.

Pátý cíl se jmenuje run, je závislý na cíli dist, a spouští úlohu java, která spustí aplikaci z vytvořeného jar archivu.

Posledním cílem je cíl clean, který obsahuje tři úlohy delete, které smažou vytvářené adresáře classes, dist a docs/api. Tento úkol slouží k návratu do výchozího stavu.

Používání programu Ant

Ant se spouští z příkazového řádku příkazem **ant.** Obvykle se spouští v adresáři, ve kterém je sestavovací soubor a jako parametr se zadává název cíle, který se má provést (vždy se provedou i všechny cíle, na kterých je tento cíl závislý). Lze zadat i různé parametry, úplný seznam parametrů lze získat po zadání příkazu **ant –help** .

Následující výstup je získán po zadání příkazu ant v situaci, kdy se změnil jeden zdrojový soubor. Ve výstupu jsou vidět jak všechny realizované cíle, tak všechny realizované úlohy v rámci jednotlivých cílů (vypisují se pouze ty úlohy, které provedou nějakou akci – v následujícím výpisu není např. uvedena úloha mkdir v rámci cíle init, neboť příslušný adresář již existoval).

```
Buildfile: build.xml

init:

compile:
    [javac] Compiling 1 source file to C:\cestovka2\build

dist:
    [jar] Building jar: C:\cestovka2\dist\lib\CestovniKancelar.jar

BUILD SUCCESSFUL

Total time: 14 seconds
```

Spuštění programu ant s parametrem **–projecthelp** vypíše jednotlivé úlohy definované v buildfile. Následuje ukázka výstupu:

```
Buildfile: build.xml
Main targets:

 clean       Delete the ${build} and ${dist} directory trees
 compile     Compile the java code from ${src} into ${build}
 dist        Create the distribution jar file
 init        Create the build directory structure
 javadoc     Generate javadoc from all .java files
 run         Run CestovniKancelar application

Default target:  dist
```

Pokud chceme dosáhnout jiného cíle než dist, musíme jméno požadovaného cíle uvést jako parametr příkazu ant. Budou provedeny i všechny cíle, na kterých je daný cíl závislý.

Nastavování path a classpath v sestavovacím souboru.

Pro provedení cílů je často třeba nastavit příslušnou classpath. V rámci elementu úkolu je možné používat element ***classpath***. Možnosti použití různých atributů u tohoto elementu jsou stejné jako u elementu *path*, který je popsán níže. Pokud je stejná classpath používána pro více úloh, je vhodnější popsat ji globálně v elementu *path* s jednoznačným id, který se definuje na úrovni elementů *target*. Pak se classpath nastaví takto:

```
<classpath refid="idOdpovidajiciPath"/>
```

Obecně se pro nastavování cest používá element ***path***, který může obsahovat několik dalších elementů. Jsou to: *pathelement*, *fileset* , *didset*, *filelist* případně i další element *path*. Použití některých z těchto elementů si ukážeme na následujícím příkladě.

```
<path id="base.path">
    <fileset dir="lib">
        <include name="**/*.jar"/>
    </fileset>
    <pathelement location="classes"/>
</path>
```

V takto nastavené cestě budou všechny jar archívy z adresáře lib a adresář classes. Na takto vytvořenou cestu je možné se odkázat z jakéhokoli úkolu pomocí refid = "base.path". Podrobnější popis a více příkladů najdete v dokumentaci.

Přehled úloh programu Ant

Následující přehled obsahuje většinu úloh, které jsou k dispozici v základní instalaci programu Ant.

úloha	**popis**
Ant	provedení cíle v jiném build souboru, používá se v případě, že existují relativně samostatné subprojekty
AntStructure	vygeneruje DTD soubor, který obsahuje všechny známé úlohy
Apply	spustí příkaz z operačního systému
Available	nastaví specifikovanou proměnnou (property) na true, pokud je dostupný zadaný soubor, či zadaná třída či zadaný zdroj JVM
Chmod	mění přístupová práva k souboru (v Unixu)
Condition	nastaví proměnnou na základě podmínky, je to zobecnění úlohy Available
Copy	kopíruje soubory či adresáře
Cvs	spolupracuje se systémem pro správu verzí (CVS)
Delete	vymaže jednotlivé soubory či celé adresáře včetně podadresářů
Ear	vytváří tzv. Enterprise application archive (ear), obsahující .jar a .war soubory
Echo	vypíše text na standardní výstup či do souboru
Exec	spustí příkaz operačního systému
Fail	ukončí sestavování s chybovou hláškou
GenKey	generuje klíč pro digitální podpis
Get	zkopíruje soubor ze zadaného URL
Gzip, Gunzip	komprimování, dekomprimování souboru

úloha	popis
Jar	vytvoření archivu .jar
Java	spustí aplikaci v Javě
Javac	přeloží zdrojový soubor programu v Javě do .class
Javadoc	vygeneruje dokumentaci v HTML formátu ze zdrojových souborů
Mail	odešle zprávu elektronickou poštou
Mkdir	vytvoří adresář(e)
Move	přesune/přejmenuje soubor(y)
Parallel	umožňuje využít více vláken pro spouštění úloh, používá se např. pro testování serverů (server je v jednom vlákně, test se spustí ve druhém vlákně)
Patch	umožňuje aplikovat diff soubory na zdrojové texty
PathConvert	upravuje jména souborů do formátu vhodného pro aktuální platformu
Property	nastavuje vnitřní proměnné, které lze použít v dalších úlohách, proměnné lze též načíst ze souboru
Record	zapisuje výstup (log) z programu ant do souboru
Replace	v zadaném souboru zamění výskyty daných řetězců za jiné
Rmic	vyvolá rmic překladač (podpora tříd spouštěných na vzdálených počítačích)
SignJar	doplní digitální podpis k jar souboru
Sleep	čeká určitou dobu
SQL	vykonává SQL příkazy vůči databázi
Style	aplikuje XSLT předpisy na XML soubory a vytvoří příslušné výstupy
Tar	vytváření archivů tar
Taskdef	umožňuje začlenit vlastní úlohy (specifikuje se jméno třídy, která se má vykonat)
Touch	vytvoří prázdný soubor pokud neexistuje, nastaví čas modifikace souboru na aktuální (obdoba příkazu touch v unixu)
Tstamp	vytvoří vnitřní proměnné programu ant, které obsahují údaje o aktuálním čase
Unjar	rozbalení archivu jar
Untar	rozbalení archivu tar
Unwar	rozbalení archivu war
Unzip	rozbaluje ZIP archivy
Uptodate	nastavuje vnitřní proměnné programu ant na základě porovnání aktuálnosti dvou souborů
War	vytváří Web Application Archive z jednotlivých souborů
Zip	vytváří ZIP archivy

tabulka 11 Přehled základních úloh nástroje Ant

Součástí distribuce programu ant je též archiv tzv. volitelných úloh, z nichž některé jsou uvedeny v následujícím přehledu. Pro spuštění některých z nich je potřeba doinstalovat ještě dodatečné programy (např. junit.jar pro testy JUnit).

úloha	popis
Cab	vytváří archivy CAB používané firmou Microsoft
EJB úlohy	úlohy pro vytváření Enterprise Java Beans
NET úlohy	podpora C# a dalších .NET technologii
FTP	umožňuje pracovat se soubory na ftp serveru

úloha	popis
JDepend	spouští nástroj JDepend nad zdrojovými programy pro měření kvality návrhu a závislostí mezi balíky,
JLink	umožňuje vytvářet archivy ze souborů v různých archivech,
JUnit	spouští testy vytvořené pomocí JUnit
JUnitReport	generuje HTML výstup z provedených JUnit testů
MimeMail	umožňuje odesílat el. poštou soubory
PropertyFile	umožňuje upravovat parametrické soubory Javy (property files)
Rpm	vytváří .rpm soubory používané v distribucích v některých verzích Linuxu
Sound	podpora pro přehrávání zvukových souborů
Telnet	podpora protokolu telnet (přihlašování na vzdálených počítačích),
XmlValidate	kontroluje, zda jsou XML soubory v pořádku (odpovídají .xsl souborům) či aspoň správně naformátované

tabulka 12 Přehled doplňkových úloh nástroje Ant

Spuštění úlohy, která není součástí distribuce Antu.

Existuje však velké množství dalších užitečných aplikací, pro které nejsou v distribuci Antu připraveny úlohy. Tyto úlohy bývají součástí distribuce těchto aplikací. Jak zapsat do sestavovacího souboru spuštění úlohy, jejíž popis je součástí této aplikace, si ukážeme na příkladě spuštění programu PMD.

Program PMD slouží pro kontrolu konvencí a hledání některých problematických konstrukcí v kódu. V dokumentaci k PMD najdete informaci, že pro spouštění z Antu je určena třída **net.sourceforge.pmd.ant.PMDTask**. Pro použití PMD z Antu musíme udělat několik kroků. Nejprve je třeba nadefinovat classpath pro jeho spuštění. Pak je třeba Antu oznámit, jak má spustit úlohu pmd. To se dělá pomocí elementu ***taskdef***, který pomocí svých atributů propojí jméno úlohy se spouštěcí třídou a classpath, na které je možné tuto třídu najít.

Samotný element *pmd*, pak prostřednictvím atributů nastavuje, jaká pravidla bude kontrolovat, výstup bude ve formátu html uložen do souboru vystupPMD.html a kontrolovat se budou všechny zdrojové soubory z adresáře src a podřízených.

Jak bude vypadat zápis spuštění programu PMD ze sestavovacího souboru vidíte v následujícím kódu.

```
<path id="pmd.classpath">
    <pathelement location="${build}"/>
    <fileset dir="C:/java/pmd-1.9/lib/">
        <include name="*.jar"/>
    </fileset>
</path>

<taskdef name="pmd" classname="net.sourceforge.pmd.ant.PMDTask"
                    classpathref="pmd.classpath"/>
```

```
<target name="pmd">
    <pmd rulesetfiles="rulesets/imports.xml,
                       rulesets/unusedcode.xml">
        <formatter type="net.sourceforge.pmd.renderers.HTMLRenderer"
                           toFile="vystupPMD.html"/>
        <fileset dir="${src}">
            <include name="**/*.java"/>
        </fileset>
    </pmd>
</target>
```