ANT kasutamine

Sisukord

ANT hankimine

ANT on võimalik saada ni binaarse distrona (*distribution*) kui ka lähtekoodina. Selles õpetuses vaatame bibaardistro kasutamist. Laadi alla viimane versioon binaardistrost aadressilt http://ant.apache.org.

ANT paigaldamine

ANT kasutamiseks peab oolema paigaldatud ja seadistatud kasutamiseks JDK (Java Development Kit).

Paki allalaaditud binaardistro meelepärasesse kohta. Soovitav on see lahti pakkida juurkausta. ANT kodukaust sisaldab järgmist struktuuri:

```
      ant
         +--- README, LICENCE, fetch.xml, teised tekstifailid
         +--- bin // sisaldab käivitusskripte
         +--- lib // sisaldab Ant jar faile ja vajalikke
        //sõltuvusi
         +--- docs // dokumentatsioon
              +--- images // erinevad logod
              +--- manual // Ant dokumentatisoon (NB! Tarvilik
                //lugemisvara)
         +--- etc // sisaldab xml faile erinevate taskide kohta
        //aruannete loomiseks, ehitusfailide
// migreerimiseks jne
```

Kausta kuhu Ant sai lahti pakitud tuntakse kui ANT_HOME.

Seadistamine

Enne kui Ant'i kasutama saab hakata tuleb seda seadistada:

1) Lisa keskonnamuutuja ANT_HOME, mis viitab kaustale kuhu Ant lahti pakiti. Windowsi konsoolilt saab seda teha näiteks nii:

```
>set ANT_HOME=c:\apache-ant-1.7.1
```

2) Lisa bin kaust keskonnamuutujale PATH. Windowsi konsoolilt saab seda teha nii:

```
>set PATH=%PATH%;%ANT_HOME%\bin
```

3) Kui seda pole veel tehtud, siis lisada JAVA_HOME keskonnamuutuja, mis viitab JDK paigalduskaustale. Windowsi konsoolilt saab seda teha nii:

```
>set JAVA_HOME=C:\Program Files\Java\jdk1.6.0_12
```

Paigalduse kontrollimine

Paigalduse kontrollimiseks tuleb konsoolile sisestada käs *ant*. Kui selle tulemusena ilmub ekraanile selline teade:

```
Buildfile: build.xml does not exist!
Build failed
```

siis Ant toimib. Järgmisena võib veel proovida käsku *ant -version* mis väljastab midagi sellist:

```
Apache Ant version 1.7.1 compiled on June 27 2008
```

Ant oskab kasutada ka CLASSPATH keskkonnamuutujat, kuid seda pole soovitav kasutada, kuna see teeb jar failid nähtavaks kõikide Java rakenduste jaoks.

CLASSPATH keskkonnamuutujast

CLASSPATH on paljude abipäringute allikas. Seepärast on hea kui võetakse arvesse järgmisi soovitusi:
Mitte kunagi ei ole vaja luua keskkonnamuutujat CLASSPATH. Ant ei vaja seda. See tekitab vaid segadust
Kui eirasid esimest soovitust, siis mitte kunagi ära kasuta jutumärke CLASSPATH muutujas.
Kui eirasid esimest soovitust, siis ära kunagi lisa längkriipsu (\) CLASSPATH muutujale.
Ant'i saab kasutada ilma CLASSPATH muutujat arvestamata kasutades võtit *-noclasspath.*

ANT õpetus

Ant õppimiseks teeme ühe ülimalt lihtsa Java projekti, mis koosneb ainult ühest failist ja mis väljastab kirja „Hello World“. Selleks loome projekti kausta. Näiteks JDevel\AntTutorial. Sinna alla loome omakorda kausta *src*, kuhu paneme kõik lähtekoodifailid (.java failid). Teeme lähtefaili HelloWorld.java:

```
package oata;
public class HelloWorld {
      public static void main(String[] args) {
      System.out.println("Hello World");
      }
}
```

Salvestame selle faili src\oata\HelloWorld.java.
Kasutades Java vahendeid kompileerime selle faili:

```
md build\classes
javac -sourcepath src -d build\classes src\oata\HelloWorld.java
java -cp build\classes oata.HelloWorld
```

Selle tulemusena kuvatakse

```
Hello world
```

Java rakenduste levitamiseks ja juurutamiseks (*deploy*) on mõistlik kompileeritud klassifailid koondada .jar arhiivifailidesse. Jar faili loomine pole kuigi keerukas Java vahenditega kuid käivitatava jar failiga on pisut rohkem tegemist:

```
echo Main-Class: oata.HelloWorld>myManifest
md build\jar
jar cfm build\jar\Helloworld.jar myManifest -C build\classes .
Java -jar buils\jar\HelloWorld.jar
```

NB! Eelmises näites real, mis algab käsuga jar on lõpus punkt. See on käsu osa ja seda ei tohi ära jätta.

Sedasi saime siis Java vahenditega faili kompileeritud ja loodud käivitatava jar faili,mis sisaldab kompileeritud faili.

Vaadates seda läbiviidud protsessi kus meil tuleb kompileerida failid, määrata käivitusfail (millises failis on Main meetod, mis kogu rakenduse käivitab) ja lõpuks veel failid kokku pakkida. Meie näite korral ei anna pakkimine erilist eelist kuid kui meil on käsil suure, tuhandeid klasse, mis on veel erinevates pakettides, rakenduse ehitamine, siis käiks see Java vahendeid kasutades üle jõu, oleks aeganõudev ja veaaldis. Lisaks on ka hea praktigka see, et genereeritud failid kustutatakse enne uut ehitust e tehakse nn ehituse puhastus („clean build“). Vaatame kuidas sama asja teha kasutades Ant'i. Ant kasutab vaikimisi ehitusfaili build.xml. Meie väga lihtne ehitusfail on järgmine:

```
<project>
      <target name="clean">
            <delete dir="build" />
      </target>

      <target name="compile">
            <mkdir dir="build/classes" />
            <javac srcdir="src" destdir="build/classes"/>
      </target>

      <target name="jar">
                  <mkdir dir="build/jar" />
                  <jar destfile="build/jar/HelloWorld.jar"
                        basedir="build/classes">
                        <manifest>
                              <attribute name="Main-Class"
                              value="oata.HelloWorld"/>
                        </manifest>
                  </jar>
      </target>

      <target name="run">
                  <java jar="build/jar/HelloWorld.jar"
                                    fork="true" />
      </target>

</project>
```

Nüüd on võimalik kompileerida, pakkida ja käivitada rakendus järgmiste käskudega:

```
ant compile
ant jar
ant run
```

või lühidalt

```
ant compile jar rn
```

Nüüd kus meil on töötav ehitusfail saame teha sinna täiendusi: palju kordi viidatakse samadele kaustadele, peaklass (klass *main*) ja jar nimi on koodi sissekirjutatud ja käivituse ajal tuleb jälgida õiget ehitusetappide järjekorda. Esimene ja teine puudus on parandatav omaduste (*properties*) kasutusele võtuga, kolmas srilise märgendi <project> abil ja neljas sõltuvuste abil.

Muudetud ehitusfail on nüüd järgmine:

```
<!-- default viitab targetile mis käivitatakse kui ant käivitatakse ilma
     parameetriteta
-->
<project name="HelloWorld" basedir="." default="main">
     <property name="src.dir" value="src" />
     <property name="build.dir" value="build" />
     <property name="classes.dir"
          value="${build.dir}/classes" />
     <property name="jar.dir" value="${build.dir}/jar" />
     <property name="main-class" value="oata.HelloWorld" />

     <target name="clean">
          <delete dir="${build.dir}" />
     </target>

     <target name="compile">
          <mkdir dir="${classes.dir}" />
          <javac srcdir="${src.dir}" destdir="${classes.dir}"/>
     </target>

     <!-- atribuut depends viitab kompileerimisetapile, mis           tuleb
enne teostada -->
     <target name="jar" depends="compile">
               <mkdir dir="${jar.dir}" />
               <jar destfile="${jar.dir}/${ant.project.name}"
     basedir="${classes.dir}">
                    <manifest>
                         <attribute name="Main-Class"
                              value="${main-class}"/>
                    </manifest>
               </jar>
     </target>

     <target name="run" depends="jar">
               <java jar="${jar.dir}/${ant.project.name}.jar"
          fork="true" />
     </target>
     <target name="clean-build" depends="clean,jar" />
     <target name="main" depends="clean,run" />
</project>
```

Nüüd pole rakenduse ehitamiseks muud tarvis kui konsoolilt sisestada käsk *ant*. Selle tulemusena kuvatake:

```
C:\JDevel\AntTutorial>ant
Buildfile: build.xml

clean:
   [delete] Deleting directory C:\JDevel\AntTutorial\build

compile:
    [mkdir] Created dir: C:\JDevel\AntTutorial\build\classes
    [javac] Compiling 1 source file to C:\JDevel\AntTutorial\build\classes

jar:
    [mkdir] Created dir: C:\JDevel\AntTutorial\build\jar
      [jar] Building jar: C:\JDevel\AntTutorial\build\jar\HelloWorld.jar

run:
     [java] Hello World

main:

BUILD SUCCESSFUL
```

Väliste teekide (*library*) kasutamine

Millegipärast öeldakse, et ei ole hea kasutada syso (*system out*) käske.Logi käskudena peaksime kasutama logimise API't.Selleks kasutame log4J'd. Selle saab allalaadida Logging'u kodulehelt http://logging.apache.org/log4j/1.2/download.html.

Selles õpetuses kasutame Log4I versiooni 1.2.15. Järgnavalt tuleb fail log4j-1.2.15.jar kopeerida meie rakenduse kausta lib. Nüüd muudame oma HelloWorld.java faili:

```
package oata;

import org.apache.log4j.Logger;
import org.apache.log4j.BasicConfigurator;

public class HelloWorld {
      static Logger logger = Logger.getLogger(HelloWorld.class);
      public static void main(String[] args) {
      BasicConfigurator.configure();
            logger.info("Hello World"); // the old SysO-statement
      }
}
```

Peale neid muudatusi ei ole mõtet kohe Ant'i käivitada sest see annaks meile terve hulga kompileerimisvigu. Log4J ei ole classpath'il, seega tuleb meil pisut muudatusi teha. Kuid pole vaja muuta CLASSPATH keskkonnamuutujat. Selle asemel me tutvustame Lo4J,mis paikneb kuskil .\lib kaustas meie ehitusfailile.

```
<project name="HelloWorld" basedir="." default="main">
      ...
      <property name="lib.dir" value="lib"/>
      <path id="classpath">
            <fileset dir="${lib.dir}" includes="**/*.jar"/>
      </path>
      ...
      <target name="compile">
            <mkdir dir="${classes.dir}"/>
            <javac srcdir="${src.dir}" destdir="${classes.dir}"
            classpathref="classpath"/>
      </target>

      <target name="run" depends="jar">
            <java fork="true" classname="${main-class}">
                  <classpath>
                        <path refid="classpath"/>
                        <path location="${jar.dir}/${ant.project.name}.jar"/>
                  </classpath>
            </java>
      </target>
      ...
</project>
```

Konfiguratsioonifailid

Miks me kasutasime Log4J'd. Sest see on hästi kofigureeritav? Ei – kõik on koodi sissekirjutatud. Kuid see pole Log4J viga vaid meie viga. Me kirjutasime java koodi `BasicConfigurator.configure();`

mis teostab lihtsat kuid sissekodeeritud konfiguratsiooni. Mugavam oleks kasutada konfiguratsioonifaili. Java koodis kustuta `BasicConfigurator.configure()` ja sellega seotud impordikäsk. Siis otsib Log4J konfiguratsiooni nagu kirjeldatud Log4J juhendis. Loome uue faili `src/log4j.properties. See on Log4J vaike konfiguratsioonifaili nimi.`

Faili sisu on järgmine:

```
log4j.rootLogger=DEBUG, stdout
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%m%n
```

See konfiguratsioon loob kanali („Appender“) konsoolile nimega stout mis prindib sõnumi (%m), millele järgneb reavahetus (%n). Tulemus on sama mis meie kõige esimeses lähtefailis System out.println(). Aga see pole veel kõik. Meil tuleb ka konfiiuratsioonifail edastada koos muude rakenduse osadega. Seega tuleb muuta ka ehitusfaili:

```
...
<target name="compile">
      <mkdir dir="${classes.dir}"/>
      <javac srcdir="${src.dir}" destdir="${classes.dir}"
      classpathref="classpath"/>
      <copy todir="${classes.dir}">
            <fileset dir="${src.dir}" excludes="**/*.java"/>
      </copy>
</target>
...
```

See kopeerib kõik ressursid (mis pole laiendiga .java) ehituskausta, nii et saame käivitada rakenduse sellest kaustast ja need failid lisatakse jar'i.

Klassi testimine

Tutvume Junit testimisraamistikuga kombineerituna Ant'iga. Kuna Ant omab sisseehitatud JUnit 3.8.2, saab seda kasutada. Kirjutame testimisklassi src\HelloWorldTest.java:

```
public class HelloWorldTest extends junit.framework.TestCase {
      public void testNothing() {
      }
      public void testWillAlwaysFail() {
      fail("An error message");
      }
}
```

Kuna meil ei ole mingit äriloogikat, mida testida, siis on se testimisklass väga lühike ja ainult näitab kust alustada. Täiendavaks info samiseks tuleb uurida Junit'i dokumentatsiooni ja Ant manuaali junit taski. Järgnevalt lisame junit instruktsiooni ehitusfaili. Kuna Ant.1.7.2 ei sisalda Junit.jar faili, siis tuleb see alla laadida lehelt http://www.junit.org/. Selles näites kasutame JUnit versiooni 3.8.2. Alla laaditud .zip failist tuleb extraktida fail junit.jar kausta %ANT_HOME%\lib.

```
...
<target name="run" depends="jar">
      <java fork="true" classname="${main-class}">
            <classpath>
            <path refid="classpath"/>
            <path id="application" location="${jar.dir}/
                        ${ant.project.name}.jar"/>
            </classpath>
      </java>
</target>
<target name="junit" depends="jar">
      <junit printsummary="yes">
      <classpath>
            <path refid="classpath"/>
            <path refid="application"/>
      </classpath>
      <batchtest fork="yes">
            <fileset dir="${src.dir}" includes="*Test.java"/>
      </batchtest>
      </junit>
</target>
...
```

Me taaskasutame rada (*path*) jar failini, mis on juba defineeritud run taskis andes sellele ID. Atribuut printsummary=“yes“ lubab näha rohkem infot kui „FAILED“ või „PASSED“ teated. Testide käivitamiseks kasutame märgendit batchtest, siis on lihtne täiendavaid teste lisada.Selleks tuleb lihtsalt vastavad klassid nimetada *Test.java. See on üldkasutatav nimekokkuleppe skeem. Kui nüüd konsoolile kirjutada ant junit, siis saame tulemuseks:

```
...
junit:
[junit] Running HelloWorldTest
[junit] Tests run: 2, Failures: 1, Errors: 0, Time elapsed: 0,01 sec
[junit] Test HelloWorldTest FAILED
BUILD
...
```

Me saaame ka luua testiaruende, mida me ise või keegi teine saaks lugeda peale konsooli sulgemist. Siin on kaks sammu:

1. Las <junit> logib informatsiooni
2. Teisendada see loetavale (lehitsetavale - *browsable*) kujule

```
...
<property name="report.dir" value="${build.dir}/junitreport"/>
...
<target name="junit" depends="jar">
      <mkdir dir="${report.dir}"/>
      <junit printsummary="yes">
      <classpath>
            <path refid="classpath"/>
            <path refid="application"/>
      </classpath>
      <formatter type="xml"/>
      <batchtest fork="yes" todir="${report.dir}">
            <fileset dir="${src.dir}" includes="*Test.java"/>
      </batchtest>
      </junit>
</target>

<target name="junitreport">
      <junitreport todir="${report.dir}">
            <fileset dir="${report.dir}" includes="TEST-*.xml"/>
            <report todir="${report.dir}"/>
      </junitreport>
</target>
```

Kui nüüd käivitada

```
ant junit junitreport
```

, siis luuakse kausta build\junitreport hulk faile (.xml, .css, .html). XML failid on junitreport tulemus. Selleks et neid oleks mugavam vaadata on loodud ka html failid mis sarnanevad ülesehituselt ja väljanägemiselt javadoc'le. Raporti vaatamiseks tuleb käivitada fail index.html.