Kurzanleitung Ant

Bei großen wie kleinen Softwareprojekten ist oft ein Werkzeug nötig, dass hilft Quellcode und andere Resourcen in ein ausführbares Programm oder eine Bibliothek zu verwandeln. Dazu sind meistens viele kleine Schritte nötig. In der C/C++-Welt hat sich dafür das Programm `make` als Standard etabliert. In der Java-Welt gibt es für diesen Zweck *Ant*.

Ant war zunächst ein Teil des Jakarta Tomcat, ist nun aber ein eigenes Projekt von Apache. Der Name Ant ist eine Abkürzung und steht für „Another Neat Toll“. Es ist komplett in Java geschrieben und steht auch im Quelltext zur Verfügung.

Diese Kurzanleitung soll nur einen Überblick über die wichtigsten Funktionen von Ant liefern. Der Nutzungsbereich von Ant ist nicht ausschließlich auf das Kompilieren von Software beschränkt, es kann auch für viele andere Dinge genutzt werden. Eine ausführliche Anleitung ist unter [1] zu finden.

Es ist typischerweise nicht nötig, Ant zu installieren. Jedes aktuelle JDK bringt auch ein aktuelles Ant mit.

1 Nutzung

Um anzugeben, welche Aufgaben Ant ausführen soll, muss eine Datei angelegt werden, die eine Beschreibung der Aufgaben enthält. Diese werden als XML-Datenstruktur abgelegt. Standardmäßig heißt diese Datei `build.xml`. Andere Namen sind auch möglich, müssen dann aber beim Aufruf von Ant mit angegeben werden.

Um Ant zu starten genügt die Eingabe von `ant` auf der Kommandozeile. Es sucht dann im aktuellen Verzeichnis die Builddatei `build.xml` und führt darin das Standardziel aus. Soll ein anderes Ziel ausgegeben werden, so wird der Name des Ziels einfach als Parameter angehängt.

2 Aufbau einer Builddatei

Jede XML-Datenstruktur hat genau einen Wurzelknoten. Dieser heißt bei Ant `project`. Hier kann dem Projekt ein Name gegeben werden indem das Attribut `name` verwendet wird. Dies ist zwar optional, wird jedoch empfohlen.

Innerhalb eines `project`-Tags können verschiedene weitere Tags verwendet werden. Die gebräuchlichsten sind `property`, `path` und `target`.

2.1 Eigenschaften

Eigenschaften haben die Funktion einer Variablen. So können am Anfang einer Builddatei einige Eigenschaften festlegen werden, auf die im weiteren Verlauf des Skripts zugegriffen wird. Als Beispiel seien hier Pfadangaben oder auch Versionsnummern genannt. Das nötige XML-Tag sieht folgendermaßen aus:

```
<property name="destdir" value="dist"/>
```

Hiermit wird eine Eigenschaft mit dem Namen `destdir` angelegt und mit dem Wert `dist` initialisiert. Der Zugriff auf diese Eigenschaft erfolgt über `${destdir}`. Dies kann an allen Stellen im Skript verwendet werden. So könnte man zum Beispiel eine weitere Eigenschaft mit dem Wert `${destdir}/bin` belegt werden. `${destdir}` würde dann durch `dist` ersetzt werden.

2.2 Vordefinierte Eigenschaften

Es gibt eine Reihe von vordefinierten Eigenschaften. Alle Eigenschaften, die man unter Java durch den Aufruf der Methode `System.getProperties()` abrufen kann, stehen so zur Verfügung. Des weiteren gibt es einige Werte, die von Ant direkt geliefert werden. Dazu zählt insbesondere die Eigenschaft `basedir`. Sie enthält das Wurzelverzeichnis des Ant-Laufes. So arbeitet es sich mit relativen Pfadangaben.

Zusätzlich ist es möglich, die Umgebungsvariablen des Systems einzubinden. Dazu ist folgende Zeile nötig:

```
<property environment="env" />
```

Ab jetzt stehen alle Umgebungsvariablen unter der Eigenschaft `env.`*`<Name der Umgebungsvariablen>`* zur Verfügung.

2.3 Pfade

Häufig werden Pfade benötigt, um ein Projekt zu erstellen. Ein sehr häufig benötigter Pfad ist der Klassenpfad. Dabei handelt es sich nicht um einen einzelnen Pfad, der zum Beispiel auf eine Datei oder ein Verzeichnis zeigt, sondern um eine Auflistung von vielen Dateien und Verzeichnissen. Um solche Strukturen darzustellen gibt es das Tag `path`. Ein Klassenpfad könnte zum Beispiel so angegeben werden:

```
<path id="classpath">
  <pathelement path="${basedir}/lib" />
  <fileset dir="${basedir}/external_libs">
    <include name="*.jar" />
  </fileset>
</path>
```

Hier wird ein Pfad mit dem Namen `classpath` angelegt. Dort wird ein Verzeichnis eingefügt (`${basedir}/lib`). Anschließend werden alle jars aus dem Verzeichnis `${basedir}/external_libs` hinzugefügt.

2.4 Ziele

In einem Ant-Skript können mehrere *Ziele* angegeben werden. Ein Ziel könnte es zum Beispiel sein, alle für ein Programm nötigen Teile gebaut zu haben. Es ist beim Aufruf von Ant möglich, ein Ziel anzugeben,

welches ausgeführt werden soll. Auch ist es möglich, Abhängigkeiten unter den Zielen anzugeben. Die Art und Weise *wie* ein Ziel erreicht wird, muss dabei aber immer explizit angegeben werden. Ein Ziel wird folgendermaßen definiert:

```
<target name="build" depends="clean" description="Baut die Anwendung neu">
  ...
</target>
```

Hier wird ein Ziel mit dem Namen „build“ definiert. Bevor dieses Ziel jedoch ausgeführt werden kann, muss das Ziel „clean“ ausgeführt werden. Die Beschreibung des Ziels ist optional, wird jedoch empfohlen. Innerhalb des Tags werden die Aufgaben angegeben, die ausgeführt werden müssen, um das Ziel zu erstellen. Hier könnten dann die Anweisungen stehen, die die Anwendung erstellen.

2.5 Aufgaben

Bisher wurden nur Tags besprochen, die im engeren Sinne keine Aktionen durchführen. Es fehlt also noch die Möglichkeit Ergebnisse zu produzieren. Es gibt eine große Menge an so genannten Tasks, die bereits in Ant eingebaut sind. Da jeder Task anders aufgebaut ist, kann hier keine vollständige Beschreibung folgen. Es folgt aber eine Tabelle mit häufig benötigten Tasks.

Task	Beschreibung
javac	kompiliert Java-Quellecode
mkdir	legt ein Verzeichnis an
copy	kopiert eine Datei oder ganze Verzeichnisstrukturen
zip	komprimiert Dateien oder Verzeichnisse
junit	führt JUnit-Tests aus
antcall	ruft ein anderes Ziel in der selben Build-Datei auf
jar	erstellt ein Jar
echo	gibt eine Meldung aus

3 Beispiel

Es folgt ein sehr einfaches Beispiel, mit dem ein Java-Projekt übersetzt wird.

```
<?xml version="1.0"?>
<project name="Buzzwordgenerator" default="build">
  <!-- Einige Eigenschaften setzen -->
  <property name="srcdir" value="src" />
  <property name="bindir" value="bin" />
  <property name="destdir" value="dist" />
  <property name="jarfile" value="${destdir}/Buzzwordgenerator.jar" />

  <!-- Klassenpfad setzen -->
  <path id="classpath">
    <pathelement path="${basedir}/lib" />
    <fileset dir="${basedir}/external_libs">
      <include name="*.jar" />
    </fileset>
  </path>

  <!-- Ziel -->
  <target name="build" description="Baut die Anwendung">
    <!-- Quelltext kompilieren -->
    <javac srcdir="${srcdir}"
           destdir="${binsrc}"
           debug="false"
           deprecation="true"
           optimize="true" >
      <classpath refid="classpath" />
    </javac>

    <!-- Baut die JAR-Datei -->
    <jar jarfile="${jarfile}">
      <fileset dir="${bindir}"/>
    </jar>
  </target>

  <target name="clean" description="Räumt auf">
    <delete file="${jarfile}" />
    <delete>
      <fileset dir="${bindir}" />
    </delete>
  </target>

  <target name="rebuild" depends="clean,build" description="Baut die Anwendung neu" />
</project>
```

Literatur

[1] URL `http://ant.apache.org/manual/index.html`.

Autor: Malte Ried, Institut für SoftwareArchitektur.