Überblick

- **ANT** ist ein Build-Tool ähnlich wie **make** (?)
- Teilprojekt der Apache Software Foundation
- **ANT** ist Opensource
- Warum dann ein neues Build-Tool?
  - Plattformunabhängig
  - Im Vergleich zu make sind Build-Files einfacher zu handhaben
  - Kein eigenes Dateiformat sondern XML basiert
  - Unabhängigkeit von einer IDE gewährleisten

Überblick

- **ANT** ist ein Build-Tool ähnlich wie **make** (?)
- Teilprojekt der Apache Software Foundation
- **ANT** ist Opensource
- Warum dann ein neues Build-Tool?

Inhalt

- Überblick
- Installation und Grundlagen
- Beispiel
- Core-Tasks
  - Dateioperationen
  - Kompilierung, Versionierung und Deployment
  - SQL und XSLT
- Optionale Tasks und Erweiterungen
- Integration in IDE's
- Ant erweitern
- Zusammenfassung
- Ressourcen
- Folienübersicht

Apache Ant 1.6

Thomas Kraus

6. Mai 2003

Projekt

- Build-File wird eingeleitet mit:
  `<?xml version="1.0" encoding="ISO-8859-1"?>`
- Die Targets und Properties befinden sich alle zwischen:

```
<project name="Projektname" default="Default Target"
basedir="Projekt Basis Verzeichnis">
...
</project>
```

- Die Attribute sind:
  - **name** - gibt dem Projekt einen Namen
  - **default** - das Target, das ausgeführt wird, wenn kein anderes angegeben wird
  - **basedir** - das Wurzelverzeichnis des Projekts (meist .)

Aufbau

- **ANT** selbst muss nicht konfiguriert werden
- Wenn Dateiname **build.xml**, dann findet **ANT** diese Datei automatisch, sonst Aufruf:

```
ant -buildfile <buildfile>
```

- oder über Java:

```
java -Dant.home=c:\ant org.apache.tools.ant.launch.Launcher <options> <target>
```

- Ein Build-File besteht aus einer Menge von **Properties** und **Targets**
- Ein Target stellt eine Aktion, die man auf das Projekt ausführen können soll, dar
- Targets können von anderen Targets abhängig sein

Installation

- Gute Nachricht: **ANT** ist in Eclipse mit installiertem SDK bereits enthalten
- Konsolennutzer müssen etwas mehr tun:
  - Vorraussetzung:
    - Installiertes JDK1.2 (Java 2) oder höher
    - Umgebungsvariable `JAVA_HOME` auf Java Installation gesetzt
    - `JAVA_HOME/bin` befindet sich im Pfad
  - Aktuelle Version (1.6.1) von **ANT** herunterladen
  - Datei in ein beliebiges Verzeichnis entpacken
  - Umgebungsvariable `ANT_HOME` auf dieses Verzeichnis setzen
  - Verzeichnis `ANT_HOME/bin` in den Pfad aufnehmen

Funktionen (Ausschnitt)

- **ANT** steht den Makefiles trotz Plattformunabhängigkeit in nichts nach
- Beliebig durch Java Code erweiterbar
- **ANT** kann:
  - Java Code compilieren
  - Dokumentation aus dem Quellcode erstellen (Javadoc und andere)
  - Jars und komplette Releases erzeugen . . .
  - Archive erzeugen/entpacken (zip, gzip)
  - Jars und komplette Releases erzeugen
  - Dateioperation (Kopieren, Verschieben, Zugriffsrechte ändern) durchführen
  - SQL Operationen an Datenbanken abschicken
  - Sounds nach Beendigung des Build-Vorgangs abspielen :)
  - . . .
- Für einen kompletten Überblick **ANT** Dokumentation lesen!

Beispiel

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<project name="Example" default="compile" basedir=".">
    <property name="src" value="src"/>
    <property name="build" value="build"/>
    <property name="dist" value="dist"/>

    <target name="compile">
        <javac srcdir="${src}" destdir="${build}" debug="on">
            <classpath refid="project.class.path"/>
        </javac>
    </target>

    <target name="dist" depends="compile">
        <mkdir dir="${dist}"/>
        <jar jarfile="${dist}/${project}.jar" basedir="${build}">
            <manifest>
               <attribute name="Main-Class" value="edu.ant.StartClass"/>
            </manifest>
        </jar>
    </target>
</project>
```


Properties

- Mit Properties kann man häufig benutzte Strings definieren (z.B.Verzeichnisse, Namen)
- Definition wie folgt:

```
<property name="Ein Name" value="Ein Wert"/>
```

- z.B.

```
<property name="src" value="~/projects/src"/>
```

- Zugriff auf Properties wie folgt:

```
${Property Name}
```

- Auslagern in Property-Dateien

```
<property file="build.properties" />
```


Targets(2)

- Ein Beispiel:
  `<target name="echo">`
  `<echo>HalloWelt</echo>`
  `</target>`
  Würde 'Hallo Welt' auf den Bildschirm schreiben
- Targets werden auf der Kommandozeile wie folgt aufgerufen:
  `ant [-buildfile <Build File>] <Target Name>`
- Lässt man
  `<Target Name>`
  weg, wird das **Default Target** ausgeführt

Targets

- Die Targets beschreiben die eigentlichen Aktionen auf dem Projekt
- Alles was zwischen:

```
<target name="Target Name">
...
</target>
```

steht wird von **ANT** interpretiert und ausgeführt(Tasks)

Core Tasks - SQL und XSLT

```
<sql
    driver="org.database.jdbcDriver"
    url="jdbc:database-url"
    userid="sa"
    password="pass"     <xslt in="doc.xml" out="build/doc/output.xml"
    src="data.sql"            style="style/apache.xsl">
    rdbms="oracle"       <outputproperty name="method" value="xml"/>
    version="8.1."       <outputproperty name="encoding"
    >                                    value="iso8859_1"/>
insert                   <outputproperty name="indent" value="yes"/>
into table some_table   </xslt>
values(1,2,3,4);

truncate table some_other_table;
</sql>
```


Core Tasks - Kompilierung und Versionierung

```
<javac destdir="${build}"
       classpath="xyz.jar"
       debug="on">
  <src path="${src}"/>
  <src path="${src2}"/>
  <include name="mypackage/p1/**"/>
  <include name="mypackage/p2/**"/>
  <exclude name="mypackage/p1/testpackage/**"/>
</javac>

<cvs cvsRoot=":pserver:anoncvs@cvs.apache.org:/home/cvspublic"
     package="ant"
     dest="${ws.dir}" />
<cvs command="update -A -d"/>

<war destfile="myapp.war" webxml="src/metadata/myapp.xml">
    <fileset dir="src/jsp/myapp"/>
    <lib dir="thirdparty/libs">                    WEB-INF/web.xml
       <exclude name="jdbc1.jar"/>                 WEB-INF/lib/jdbc2.jar
    </lib>                                         WEB-INF/classes/com/myco/myapp/Servlet.class
</war>                                             META-INF/MANIFEST.MF
                                                   index.html
                                                   front.jsp
```


Core Tasks - Dateioperationen

```
<mkdir dir="${dist}/lib"/>

<copy todir="../backup/dir">
   <fileset dir="src_dir">
    <exclude name="**/*.java"/>
   </fileset>
   <filterset>
       <filter token="TITLE" value="Foo Bar"/>
   </filterset>
</copy>

<delete file="/lib/ant.jar"/>
```

- chmod, touch, move, sync, get ...

Beispiel - Ausgabe

- Der Aufruf von `ant dist` würde dazu führen das:
  - der Quellcode aus `${src}` nach `${build}` kompiliert wird (`<javac/>` Task)
  - aus den Kompilaten ein Jar-Archiv mit der zugehörigen Manifest Datei nach `${dist}` erzeugt wird (`<jar/>` Task)

Erweiterungen

- XDoclet - Attributbasierte Codeerzeugung
- AntContrib - weitere Tasks(if, cc)
- AndroMDA - Unterstützung für Model Driven Architecture
- Speech4J - Sprachausgabe während des Builds
- Generics Compiler - Unterstützung für Java 1.5
- . . .

Optionale Tasks - Scripting (BSF)

```
<project name="squares" default="main" basedir=".">
  <target name="setup">

    <script language="javascript"> <![CDATA[

      for (i=1; i<=10; i++) {
        echo = squares.createTask("echo");
        main.addTask(echo);
        echo.setMessage(i*i);
      }

    ]]> </script>

  </target>

  <target name="main" depends="setup"/>
</project>
```


Optionale Tasks - Compilergeneratoren

```
<javacc target="src/Parser.jj"
        outputdirectory="build/src"
        javacchome="c:/program files/JavaCC"
        static="true"
/>

<antlr target="etc/java.g"
       outputdirectory="build/src"
/>

<junit printsummary="yes" fork="yes" haltonfailure="yes">
  <formatter type="plain"/>
  <test name="my.test.TestCase"/>
</junit>
```


Optionale Tasks - SCP, SSH, TELNET

```
<sshexec host="somehost"
         username="dude"
         keyfile="${user.home}/.ssh/id_dsa"
         command="touch somefile"/>

<scp todir="user:password@somehost:/home/chuck">
  <fileset dir="src_dir">
    <include name="**/*.java"/>
  </fileset>
</scp>

<telnet port="80" server="localhost" timeout="20">
  <read/>
  <write>GET / http/0.9</write>
  <write/>
  <read timeout="10">&lt;/HTML&gt;</read>
</telnet>
```

Ant erweitern (2)

```
import org.apache.tools.ant.Task;
import org.apache.tools.ant.BuildException;

public class HelloWorld extends Task {

    String message;
    public void setMessage(String msg) {
        message = msg;
    }

    public void execute() {
        if (message==null) {
            throw new BuildException("No message set.");
        }
        log(message);
        // use of the reference to Project-instance
        String project = getProject().getProperty("ant.project.name");
        // Task's log method
        log("Here is project '" + project + "'.");
        // where this task is used?
        log("I am used in: " +  getLocation() );
    }
}
```

```
<target name="use" description="Use the Task" depends="jar">
    <taskdef name="helloworld"
             classname="HelloWorld"
             classpath="${ant.project.name}.jar"/>
    <helloworld message="Hello World"/>
</target>
```

```
[helloworld] Hello World
[helloworld] Here is project 'MyTask'.
[helloworld] I am used in: C:\tmp\anttests\MyFirstTask\build.xml:23:
```

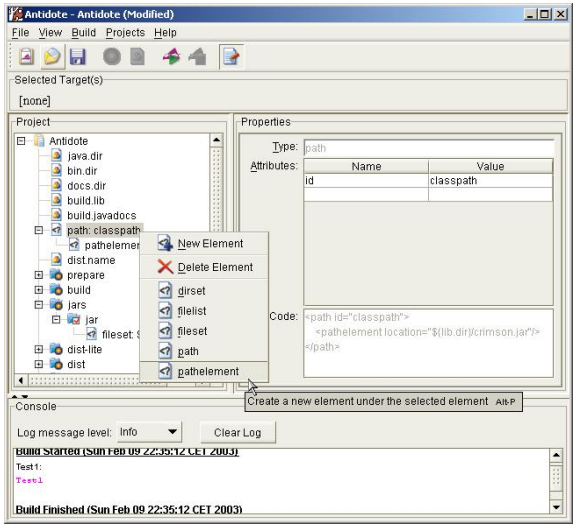

Ant erweitern

```
public class HelloWorld {
    public void execute() {
        System.out.println("Hello World");
    }
}
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<project name="MyTask" basedir="." default="use">
    ...
    <target name="use" description="Use the Task" depends="jar">
        <taskdef name="helloworld" classname="HelloWorld"
                 classpath="${ant.project.name}.jar"/>
        <helloworld/>
    </target>
</project>
```

```
Buildfile: build.xml
use:
[helloworld] Hello World
BUILD SUCCESSFUL
Total time: 3 seconds
```


Antidote



Integration in IDE's

- Integriert in
  - Eclipse/Webshere Application Developer
  - NetBeans/SunOne Studio
  - JBuilder Enterprise, JDeveloper
  - IntelliJ
  - JDEE(Emacs)
- Plugins für
  - JEdit(AntFarm), JBuilder(AntRunner), Jext(AnTwork)
- Apache Antidote

Zusammenfassung

- Plattformunabhängiges Buildtool
- weit verbreitet in der Java Welt
- unzählige Tasks und Erweiterungen verfügbar
- leicht erweiterbar
- Es existiert immer noch der `<exec />` Task!!

Zusammenfassung

- Plattformunabhängiges Buildtool
- weit verbreitet in der Java Welt
- unzählige Tasks und Erweiterungen verfügbar
- leicht erweiterbar

Tasks ausserhalb von Ant nutzen

```
import org.apache.tools.ant.Project;
import org.apache.tools.ant.Target;
import org.apache.tools.ant.taskdefs.Expand;
import java.io.File;

static public void unzip(String zipFilepath, String destinationDir) {

    final class Expander extends Expand {
        public Expander() {
          project = new Project();
        project.init();
        taskType = "unzip";
        taskName = "unzip";
        target = new Target();
        }
    }
    Expander expander = new Expander();
    expander.setSrc(new File(zipfile));
    expander.setDest(new File(destdir));
    expander.execute();
}
```


Antlib

- Bibliotheksdatei zur Gruppierung von Tasks
- enthält Definitionstasks(Typedef, Taskdef, Erweiterungen von `org.apache.tools.ant.taskdefs.AntlibDefinition`)

```
<?xml version="1.0"?>
<antlib>
   <typedef name="if" classname="org.acme.ant.If"/>
   <typedef name="scriptpathmapper"
            classname="org.acme.ant.ScriptPathMapper"
            onerror="ignore"/>
   </antlib>
```

- Benutzung im Build-Script:

```
<typedef file="sample.xml"/>

<sample:if valuetrue="${props}" xmlns:sample="sample.xml">
   <sample:scriptpathmapper language="beanshell">
      some bean shell
   </sample:scriptpathmapper>
</sample:if>
```

Folienübersicht(2)

- Core Tasks - SQL und XSLT
- Optionale Tasks - SCP, SSH, TELNET
- Optionale Tasks - Compilergeneratoren
- Optionale Tasks - Scripting (BSF)
- Erweiterungen
- Integration in IDE's
- Antidote
- Ant erweitern
- Ant erweitern(2)
- Antlib
- Tasks ausserhalb von Ant nutzen
- Zusammenfassung
- Ressourcen

Folienübersicht

- Inhalt
- Überblick
- Funktionen (Ausschnitt)
- Installation
- Aufbau
- Project
- Targets
- Targets (2)
- Properties
- Beispiel
- Beispiel - Ausgabe
- Core Tasks - Dateioperationen
- Core Tasks - Kompilierung, Versionierung und Deloyment

Ressourcen

- `http://ant.apache.org` - Offizielle Prokektseite
- `http://ant.apache.org/faq.html` - Offizielle FAQ
- `news://de.comp.lang.java` - Deutsche Java Newsgroup
- Java Development with Ant
  - Von den Ant Committern verfasstes Buch
  - `http://www.manning.com/antbook/`
- Ant in Anger: Using Ant in a Production Development System
  - `http://ant.apache.org/ant_in_anger.html`