Build-Systeme: Apache Ant

Carsten Gips (FH Bielefeld)

Automatisieren von Arbeitsabläufen

Works on my machine ...

Automatisieren von Arbeitsabläufen

Works on my machine ...

- Build-Tools:
  - **Apache Ant**
  - Apache Maven
  - Gradle

Aufbau von Ant-Skripten: Projekte und Targets

```
<project name="Vorlesung" default="clean" basedir=".">
    <target name="init" />
    <target name="compile" depends="init" />
    <target name="test" depends="compile" />
    <target name="dist" depends="compile,test" />
    <target name="clean" />
</project>
```

Aufgaben erledigen: Tasks

```
<target name="simplecompile" depends="clean,init">
    <javac srcdir="src" destdir="build" classpath="." />
</target>
```

- Beispiele: `echo`, `javac`, `jar`, `javadoc`, `junit`, ...
- Je Target mehrere Tasks möglich
- Quellen:
  - Eingebaute Tasks
  - Optionale Task-Bibliotheken
  - Selbst definierte Tasks

=> Überblick: ant.apache.org/manual/tasksoverview.html

Konsole/IDE: ant -f hello.xml

Properties: Name-Wert-Paare

```
<property name="app"       value="MyProject" />
<property name="build.dir" location="build" />

<target name="init">
    <mkdir dir="${build.dir}" />
</target>
```

- Properties beim Aufruf setzen mit Option "`-D`":
  `ant -Dwuppie=fluppie`

Tasks zum Umgang mit Dateien und Ordnern

```
<target name="demo">
    <mkdir dir="${build.dir}/lib" />

    <delete dir="${build.dir}" />
    <delete file="${dist.dir}/wuppie.jar" />

    <copy file="myfile.txt" tofile="../bak/mycopy.txt" />
    <move file="src/file.orig" tofile="bak/file.moved" />
</target>
```

Nutzung von Filesets in Tasks

```
<copy todir="archive">
    <fileset dir="src">
        <include name="**/*.java" />
        <exclude name="**/*.ba?" />
    </fileset>
</copy>

<delete>
    <fileset dir="." includes="**/*.ba?" />
</delete>
```

- "`*`" für beliebig viele Zeichen
- "`?`" für genau ein Zeichen
- "`**`" alle Unterverzeichnisse

Pfade und externe Bibliotheken

- Als Element direkt im Task:

```
<classpath>
    <pathelement location="${lib}/helper.jar" />
    <pathelement path="${project.classpath}" />
</classpath>
```

- Wiederverwendbar durch ID und "`refid`":

```
<path id="java.class.path">
    <fileset dir="${lib}">
        <include name="**/*.jar" />
    </fileset>
</path>

<classpath refid="java.class.path" />
```

Ausblick: Laden von Abhängigkeiten mit Apache Ivy

```xml
<!-- build.xml -->
<project xmlns:ivy="antlib:org.apache.ivy.ant">
    <target name="resolve">
        <ivy:retrieve/>
    </target>
</project>
```

```xml
<!-- ivy.xml -->
<ivy-module version="2.0">
    <dependencies>
        <dependency org="commons-cli" name="commons-cli" rev="1.5.0" />
        <dependency org="junit" name="junit" rev="4.13.2" />
    </dependencies>
</ivy-module>
```

Demo: ivydemo.xml

Ausblick: Weitere Build-Systeme

- Maven
  - War als Nachfolger von Ant gedacht
  - Statt wie bei Ant explizit Targets zu formulieren, geht Maven von einem Standardprojekt aus – nur noch Abweichungen müssen formuliert werden
  - Zieht Abhängigkeiten in zentralen `.maven`-Ordner
- Gradle
  - Eine Art Mischung aus Ant und Maven unter Nutzung der Sprache Groovy
- Make
  - DER Klassiker, stammt aus der C-Welt. Kann aber natürlich auch Java.
  - Analog zu Ant: Aktionen und Ziele müssen explizit definiert werden

Wrap-Up

Apache Ant: ant.apache.org

- Automatisieren von Arbeitsabläufen
- Apache Ant: Targets, Tasks, Properties
  - Targets sind auswählbare Teilziele
  - Abhängigkeiten zwischen Targets möglich
  - Tasks erledigen Aufgaben (innerhalb Targets)
  - Properties sind nicht änderbare Variablen
  - Umfangreiche Operationen auf Filesystem möglich

LICENSE