Hauptseminar

Ant - das Java Build-Tool

Funktionalität, Mächtigkeit und Praxiserfahrungen

Betreuer: Dipl.Inf. Thorsten Strufe
Vortragender: Christoph Lühr

Gliederung

Build-Tools

Aufgaben und Probleme

Ant

Arbeitsweise und Features

Erweiterungen

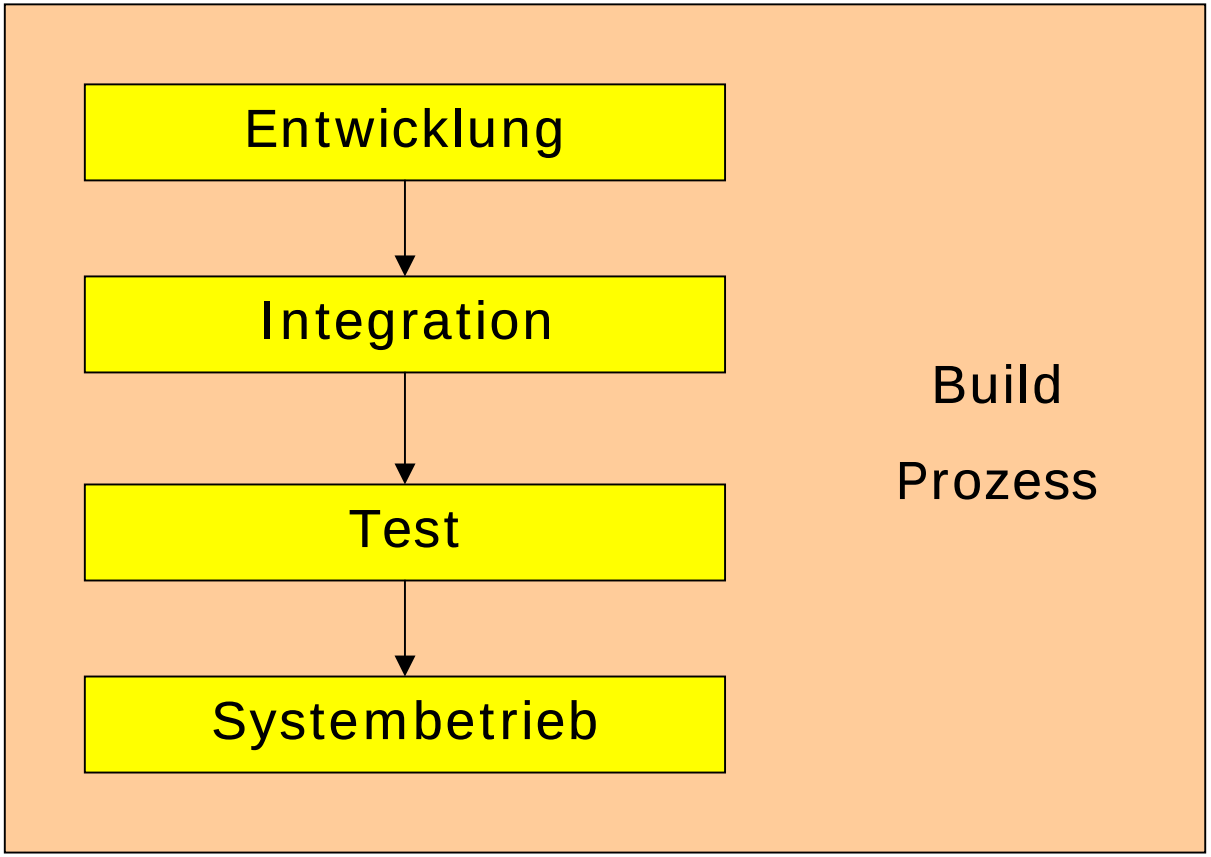
Java Build Alternativen

make, jmk & Ant im Vergleich

Build-Tools

Aufgaben und Probleme

Softwareentwicklung



Build-Tools: Aufgaben

Wozu braucht man Build-Tools?

- Erstellung eines Projektes
  - Nachvollziehbarer, immer identischer Vorgang
- Build Prozess wird
  - Strukturiert
  - Dokumentiert
  - Automatisiert

⇨ Verbindung von Entwicklungs- und Integrationsprozessen

Build-Tools: Probleme

Herausforderungen für Java Build-Tools:

- Cross-Plattform Kompatibilität
  - Dateipfadtransformation ("/" vs. "\")
  - externe Tools
- Integration Entwicklungsprozess
  - Entwicklung
  - Integration
  - Test
  - Einsatz
- IDE Einbindung

Ant

Arbeitsweise und Features
Erweiterungen

Ant: Übersicht

Aufgabe

make-Ersatz für Java Projekte

Geschichte

Apache Group Jakarta Projekt (Tomcat JSP)

Lizenz

Open Source (frei)

Implementierung

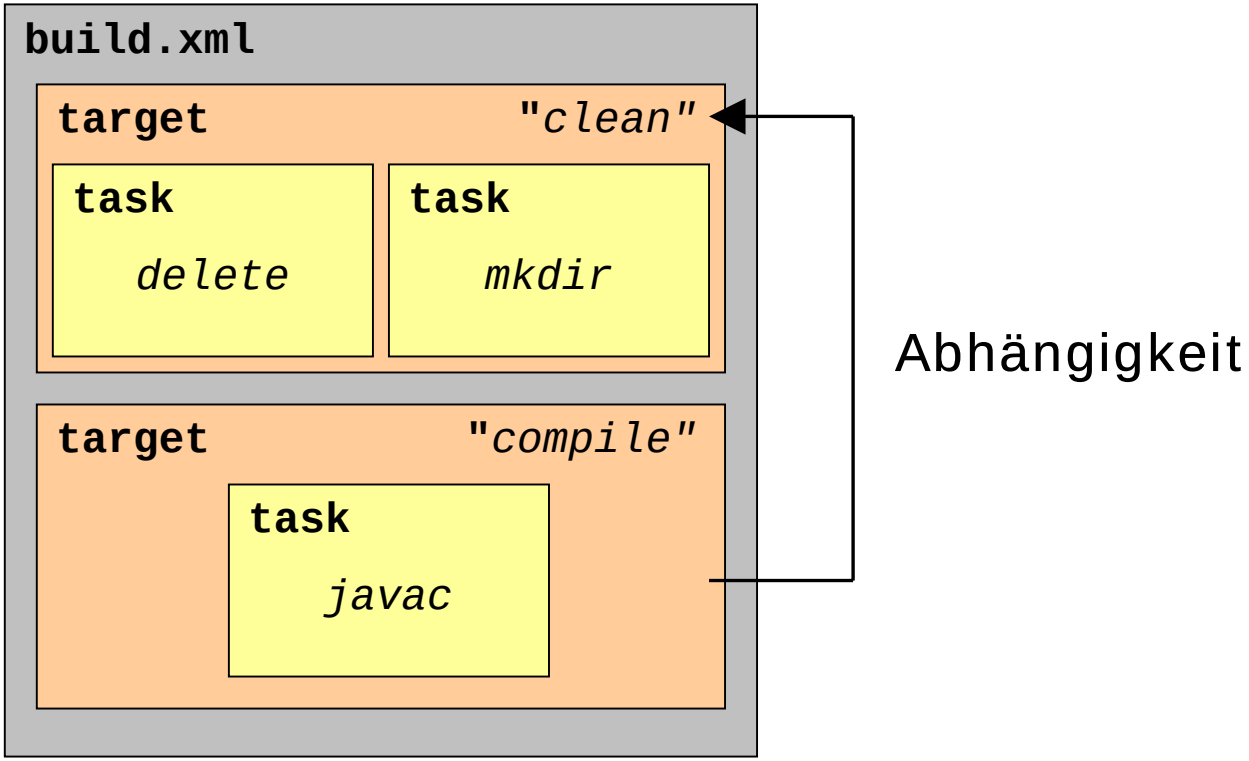
Java

IDE-Integration

JBuilder, NetBeans, VisualAge, div. Editoren

Ant: Arbeitsweise 1/2

Projektbeschreibung: XML Datei



Ant: Arbeitsweise 2/2

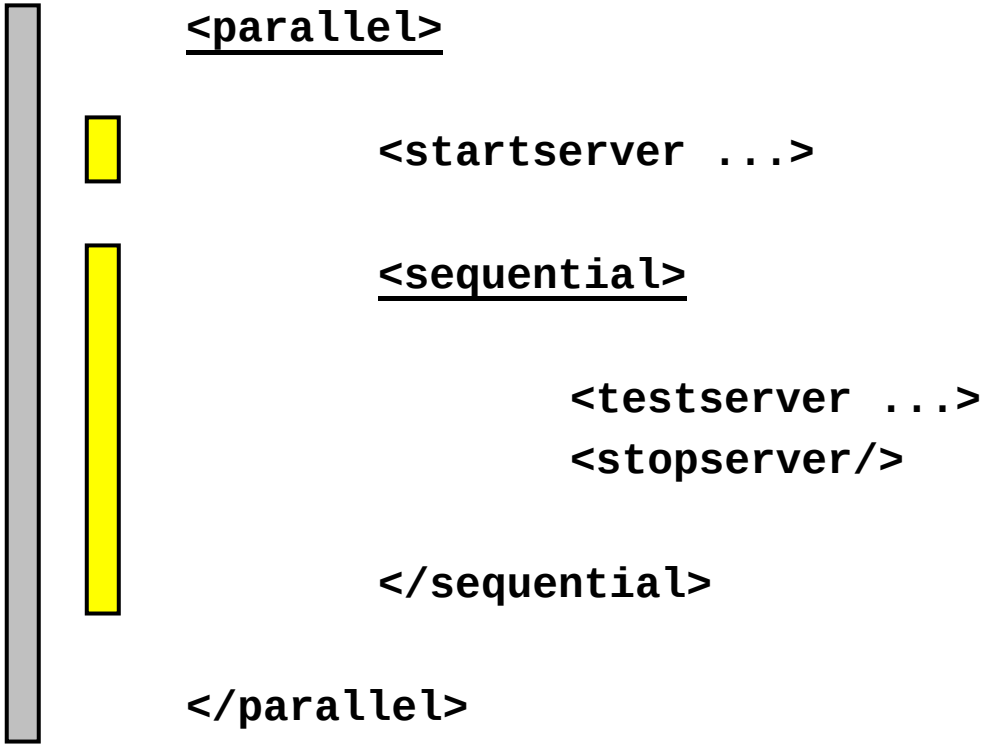
- Targets
  - Können über Kommandozeile spezifiziert werden
  - Ant unterbricht Bearbeitung bei Fehlern
  - Jedes Target wird nur einmal bearbeitet
- Tasks
  - Ausführung nur, falls notwendig ("Intelligenz")
  - Als Java Klassen in Ant integriert
  - Weitere Tasks können erstellt und verwendet werden

Ant: Beispiel **build.xml**


```
<project name="DemoProjekt" default="deploy" basedir=".">

   <target name="init">
      <property name="sourceDir" value="src"/ >
      <property name="outputDir" value="classes" />
      <property name="deployDir" value="/web/deploy/" />
   </target>

   <target name="clean" depends="init">
      <delete dir="${outputDir}" />
      <mkdir dir="${outputDir}" />
   </target>

   <target name="compile" depends="clean">
     <javac srcdir="${sourceDir}" destdir="${outputDir}" />
   </target>

   <target name="deploy" depends="compile,init">
     <copy file="${jsp}" todir="${deployDir}"/>
  </target>

</project>
```

Ant: Parallele Builds

Nutzung von Multiprozessor-Ressourcen



```
<parallel>

        <startserver ...>

        <sequential>

                <testserver ...>
                <stopserver/>

        </sequential>

</parallel>
```

Ant: Tasks

Weitere integrierte Funktionen

- SCM (Versionsverwaltung)
  - CVS, PVCS, patch
- Netzwerk
  - FTP / HTTP Dateitransfer
  - EMail
- Organisation / Paketverwaltung
  - Packen / Entpacken: JAR, RPM, GNUZip, MS CAB, ...
- XML / Dokumentation
  - XML Validator, JavaDoc

Ant: Erweiterungen 1/3

Entwicklung eigener Tasks

Beispiel: Kaffee kochen

- Task: coffee
- Parameter: extra (Milch / Zucker)

```
<coffee extra="sugar"/>
```

Ant: Erweiterungen 2/3

Implementation:

```
makeCoffee extends org.apache.tools.ant.Task ...
setExtra()...
execute() ..
```

Integration in build.xml:

```
<target name="init">
 <taskdef name="coffee" classname="local.makeCoffee"/>
 <coffee extra="sugar, cream"/>
</target>
```

Ant: Erweiterungen 3/3

Anbindung an IDE: `BuildListener` Interface

```
buildStarted(BuildEvent event);
targetStarted(BuildEvent event);
taskFinished(BuildEvent event);
```

BuildEvent Objekt Methoden:

```
getProject();
getTarget();
getTask();
```

```
> ant -listener org.apache.tools.ant.XmlLogger
```

Java Build Alternativen

make, jmk & Ant im Vergleich

Kurzprofil: make

- Portabilität
  - Implementation für viele Plattformen verfügbar
  - Abhängigkeit von externen Tools
  - keine Pfadtransformationen
- Handhabung
  - kryptische Syntax
  - Symbole (Tab, $@, $^)
- Erweiterbarkeit
  - nur externe Tools
- Performance / Leistung
  - Alle Aktionen über externe Prozesse

Beispiel: make Makefile


```
srcdir=.

JFLAGS=-O
JAVA=part1.java part2.java part3.java

all build: testprog

testprog: $(JAVA)
 <TAB>  javac $(JFLAGS) $@

clean:
 <TAB>  rm -f $(JAVA)
 <TAB>  mkdir $(srcdir)/classes
```

Kurzprofil: jmk

- Portabilität
  - benötigt nur Java VM 1.2+
  - keine weiteren betriebssystemspezifischen Tools
- Handhabung
  - Build-Dateien ähnlich make
  - Integration von Filesystemoperationen
  - Programmsteuerung (Schleifen, if, ..)
- Erweiterbarkeit
  - Java Klassen, externe Tools
  - Funktionen
- Performance / Leistung
  - Javac wird extern aufgerufen

Beispiel: jmk Makefile


```
javaflags = "-O";

srcdir = ".";
srcs = (glob (join srcdir, "/*.java"));

testprog:    srcs;
{
  exec "javac" javaflags ?;
}

"clean":;
{
  delete (join srcdir, "/*.class");
}
```

Kurzprofil: Ant

- Portabilität
  - benötigt nur Java VM 1.1+
  - keine weiteren betriebssystemspezifischen Tools
- Handhabung
  - XML-Beschreibungssprache
  - Integration von CVS, FTP, JavaDoc, ...
- Erweiterbarkeit
  - Java Klassen / externe Tools
  - Listener Interface (IDE Anbindung)
- Performance / Leistung
  - alle Tasks werden von einer VM ausgeführt

Fazit

- make
  - weite Verbreitung
  - hohe Flexibilität durch externe Tools
  - ungeeignet für Cross-Plattform Entwicklungen
- jmk
  - mächtig durch integrierte Sprache
  - portabel
- Ant
  - viele nützliche, integrierte Funktionen
  - einfache Handhabung
  - schnelle Build-Vorgänge

Ant ist einen Blick wert!

Fragen?

Vielen Dank für Ihre Aufmerksamkeit