Das Software-Projekt-Verwaltungspragramm -

Maven

Oberseminar Software-Entwicklung

Autor: Martin Hoffmann

Inhaltsverzeichnis

1. Einführung

Das Programm Maven ist eine Entwicklung der Apache Software Stiftung. Es wird größtenteils für die Arbeit an auf Java basierenden Projekten verwendet und wurde ebenfalls in Java geschrieben.

Die Programmiersprache Java wurde bei ihrer Entwicklung bereits mit einer einfachen Modularität versehen, um den Anspruch der Objektorientiertheit zu genügen. Dies umfasst die Fähigkeit Abhängigkeiten zwischen Klassen zuerkennen und eigenständig beim Übersetzen des Quelltextes zu überprüfen ob diese erfüllt werden.

Komplexere Dinge, wie zum Beispiel das Speichern der übersetzten Klassen in ein anderes Verzeichnis oder das Überprüfen des Quellcodes auf Korrektheit, können aber nur manuell oder gar nicht durchgeführt werden. Das wird von anderen Programmen übernommen, die dem Java-Übersetzer zuarbeiten oder dessen Ergebnisse aufbereiten.

Entwicklungsumgebungen bieten meist die Möglichkeiten solche Arbeiten automatisch zu erledigen. Neben diesen gibt es den Ansatz die Arbeitsschritte in Konfigurationsdateien zu beschreiben, welche von Programmen ausgelesen werden und diese dann gemäß den darin verfassten Vorschriften arbeiten.

Ein solches Programm für die Programmiersprache C/C++ ist „Make“ und für Java „Ant“. Die Vorteile für diese Programme liegt darin, dass man die Arbeit des Übersetzten, etc. automatisieren kann und so Zeit sparen und Fehlerquellen vermeiden kann.

Bei Maven handelt es sich um einen ähnlichen Ansatz, wobei es mit dem Anspruch entwickelt wurde, einfacher handhabbar und vielseitiger in seinen Möglichkeiten zu sein.

Auffällig ist, dass Maven von der gleichen Organisation entwickelt wurde, die auch Ant entwickelt und dass Maven weder den Anspruch hat Ant zu ersetzen, noch die Entwicklung an Ant eingestellt wurde. Die Zielgruppe von Maven ist eine andere als die von Ant.

Maven wurde entwickelt um bei umfangreichen Projekten eingesetzt zu werden, bei denen viele Entwickler beteiligt sind. Dabei wurde verstärkt darüber nachgedacht, wie man die Verwaltung und Kommunikation während der Entwicklung möglichst einfach gestalten kann.

Die Entwickler von Maven sahen sich Problemen gegenüber, die weit verbreitet sind, wenn es darum geht an einem großen Projekt zu arbeiten. Bei der Arbeit mit mehreren Personen an einem Projekt wird eine gut funktionierende Verwaltung des Quelltextes notwendig, was den Einsatz spezieller Programme wie zum Beispiel CVS notwendig macht. Desweiteren muss sichergestellt

werden, dass alle Entwickler eventuelle Hilfsprogramme verwenden, wenn diese für das Projekt notwendig sind. Die einzelnen Entwickler, aber vor allem die Leiter des Projektes müssen ständig einen Überblick haben, wie weit die einzelnen Teile des Projektes sind, um auf Engpässe und sich ankündigende Probleme schnell reagieren zu können.

Maven versucht diese Probleme auf einen einfache und allgemeingültige Art zu lösen, um den Einsatz für möglichst alle Projekte zu ermöglichen.

2. Der Ansatz von Maven

Um zu verhindern, dass ständig zwischen den Entwicklern Hilfsprogramme und Konfigurationsdateien herum geschickt werden und diese dann in unterschiedlichen Versionen vorliegen, kam man auf die Idee, diese an einer zentralen Stelle zu lagern und so allen Entwicklern zugänglich zu machen.

Die Konfiguration dieser Hilfprogramme sollte von ihnen selbst übernommen werden, damit die Entwickler schnell und einfach mit ihnen arbeiten können.

Es sollte möglich sein, dass die Programme mit Haupt- und Teilzielen ausgestattet werden können, um thematisch zusammenhängenden Aufgaben in einem Programm zusammenzufassen.

Deshalb kam man auf die Idee diese Programme in Form von Plugins zu realisieren. Sie werden in einem zentralen Verzeichnis abgelegt, und jeder Entwickler kann sie sich von dort herunterladen oder das Verzeichnis in seine Verzeichnisstruktur einbinden.

Die Plugins selber können auch von Maven erstellt und dann in das Sammelverzeichnis verschoben werden, um sie anderen zur Verfügung zustellen. Dies vereinfacht es eigene Plugins zu erstellen, da man ihre Entwicklung wie ein normales Mavenprojekt handhaben kann.

Bei der Installation von Maven werden grundlegende und häufig verwendete Plugins gleich mitgeliefert, weitere können dann automatisch heruntergeladen oder von Hand installiert werden.

Maven verfügt über die Möglichkeit Projekte mit Versionsnummern zu versehen, was ebenfalls den Plugins zu Gute kommt, da sie so leicht auf ihre Aktualität überprüft werden können. Ebenso kann gefordert werden, dass das zu verwendende Plugin eine bestimmte Version haben und gegebenenfalls erneuert werden muss.

Maven verfügt über eine Standardkonfiguration, die es erlaubt ein Projekt in Maven zu realisieren, ohne eigene Konfigurationen vornehmen zu müssen. Natürlich kann man diese auch an das eigene Projekt anpassen, da selten die Standardkonfiguration mit der Struktur des Projektes voll übereinstimmen wird.

Da selten ein großes, umfassendes Projekt in einem Stück entwickelt wird, gibt es in Maven die Möglichkeit mehrere Unterprojekte zu erstellen und diese in eine gemeinsame Struktur zusammen zufassen.

3. Die Arbeitsweise bekannter Programme

Wie schon erwähnt gibt es schon seit längerer Zeit ähnliche Programme, wie Make für C/C++ und Ant für Java.

Beide verfahren ähnlich, in dem für jedes Projekt eine spezielle Konfigurationsdatei angelegt wird, in welcher die einzelnen Ziele beschrieben werden. Zu jedem Ziel werden Abhängigkeiten , Vor- und Nachbedingungen definiert, die dann vom dem Programm abgearbeitet werden.

Bei Make ist es unter anderem nötig zu testen, ob erforderliche Bibliotheken zur Verfügung stehen. Ant muss diese Prüfung nicht unbedingt selbst machen, da der Java-Übersetzer diese selbst finden würde, wenn sie in entsprechenden Verzeichnissen vorhanden sind.

Der Nachteil dieser beiden Programme ist, dass für jedes Projekt eine umfassende Konfigurationsdatei geschrieben werden muss, auch wenn sich im Vergleich zu anderen nur wenig ändert. Eine weiterer Nachteil an diesem Ansatz liegt darin, dass alles in einer Datei beschreiben wird und diese recht bald unübersichtlich werden kann, was die Pflegbarkeit verschlechtert.

Vom Entwickler wird verlangt selbst bei einfachen Projekten umfangreiche Konfigurationen vorzunehmen, was einen enormen Zeitaufwand erfordert, und die Arbeit am eigentlichen Projekt behindert.

Der Entwickler muss die Arbeitsweise der einzelnen Programme, die in der Konfigurationsdatei beschreiben werden gut kennen. Was die Bereitschaft neue Programme hinzuzuziehen verringert und eventuelle der Abeitsgeschwindigkeit und -qualität nachträglich ist.

4. Ein Projekt in Maven verwalten

4.1 Erste Schritte zur Eingewöhnung

Im weiteren werden zu Verdeutlichung folgende Verzeichnisumschreibungen benutzt:

${PROJECT_HOME}	Das Stammverzeichnis des einzelnen Projektes
${MAVEN_HOME}	Das Installationsverzeichnis von Maven
${USER_HOME}	Hauptverzeichnis des Benutzers

```
/
+- src/
| +- main/
| | +- java/
| | | +- ...
| | +- resources/
| |    +- ...
| +- test/
| | +- java/
| | | +- ...
| | +- resources/
| |    +- ...
| +- site/
|    +- xdoc/
|       +- ...
+- target/
| +- ...
+- project.xml
+- README.txt
+- LICENSE.txt
```

(Maven Standardverzeichnisstruktur)

So sieht die Standardverzeichnisstruktur von Maven aus, wenn nichts anderes angegeben wird, arbeiten die Plugins mit dieser. Es ist also möglich ein Mavenprojekt zu erstellen, ohne eigene Konfigurationen vorzu nehmen. Doch dies ist nicht zu empfehlen, da zumindest der Name und Version des Projektes angegeben werden sollte.

Zum Testen aber kann diese Eigenschaft recht nützlich sein, da so schnell ein Einblick in die Arbeitsweise von Maven gewonnen werden kann. Es soll angenommen werden, dass die Standardkonfiguration verwendet wird und einige Java-Klassen als Quelltext im Verzeichnis ${PROJECT_HOME}/src/main/java vorliegen. Will man diese jetzt übersetzen und als JAR-Datei speichern, so braucht man nur in ${PROJECT_HOME}/ den folgenden Befehl ausführen:

```
maven jar
```

Die übersetzten Klassendateien und die JAR-Datei liegt danach im Verzeichnis ${PROJECT_HOME}/target vor. Die JAR-Datei wird in der Form projekt-X.Y.Z.jar erstellt.

Will man nur übersetzen kann man dies ebenso einfach:

```
maven java:compile
```

Die Struktur des Befehls ist absichtlich einfach gehalten. Zuerst kommt der Name des Plugins, der auch ausreicht, wenn man das Hauptziel des Plugins verwenden will. Soll ein Teilziel ausgeführt werden, folgt dies nach einem Doppelpunkt. Die zur Verfügung stehenden Plugins und deren Ziele findet man auf der Maven-Webseite bzw. in der Dokumentation des einzelnen Plugins. Eine Übersicht über die Ziele eines Plugins erhält man bei folgendem Befehlsaufruf.

```
maven -P pluginname
```

Maven kann zur Erstellung einer Projektdokumentation benutzt werden, die über die einfache Möglichkeiten von JAVADOC hinausgeht. Jedes Plugin liefert eigene Dokumentationen, die über ein spezielles Plugin zu einem gemeinschaftlichen Bericht zusammen gefasst wird.

```
maven site
```

Zum Abschluss löscht man überflüssige Datei, die während des Übersetzungsprozess angefallen sind mit einem weiteren Plugin.

```
maven clean
```

4.2 Maven an ein eigenes Projekt anpassen

Die wichtigste Datei für ein Mavenprojekt ist ${PROJECT_HOME}/project.xml. In dieser Datei wird die Verzeichnisstruktur des Projektest beschrieben, falls diese vom Standard abweicht. Ebenso Name, Version und Abhängigkeiten des Projektes und weiteres wird dort definiert.

Versionen können im Format X.Y.Z[-SNAPSHOT] angegeben werden, wobei X,Y und Z für Zahlenwerte stehen. Der Zusatz Snapshot soll verdeutlichen, das es sich zwar um eine bestimmte Version, aber eine nicht vollständig getestete und damit eventuell instabile Vorabveröffentlichung handelt.

Die „groupId“ ermöglicht es mehrere Projekte einer gemeinsamen Gruppe zuzuordnen, dann werden diese nach ihrer Erstellung in einem gemeinsamen Verzeichnis abgespeichert.

Anhängigkeiten zu anderen Plugins können auch Forderungen an die Versionen dieser Stellen, um sicher zugehen, das Funktionen vorhanden sind.

Weiter führende Informationen zu den einzelnen Komponenten der project.xml finden sie auf der Webseite von Maven im Bereich Referenzen.

In der project.xml finden desweiteren Angaben über die Organisation, die Entwickler, und Mailinglisten platz, um die Herkunpft von Projekten schnell herausfindbar zu machen und die Kommunikation zwischen Anwender und Entwickler zu erleichter, damit Fehler schnell beseitigt werden und Verbessungsvorschläge zielgerichtet versendet werden können.

Desweiteren ist es auch möglich systemspezifische Daten, die für alle Mavenprojekte gelten sollen in einer Datei zuspeichern, um etwa den Pfad zu einer besonderen Bibliothek oder Javaversion zu setzen, die nicht oder abweichend in Umgebungsvariablen gespeichert sind. Dies geschieht über ${USER_HOME}/build.properties. So kann man beispielsweise standardmäßig in seinem System die Javaversion 1.4.2 verwenden für Maven aber 1.5. Durch das Anlegen einer ${PROJECT_HOME}/build.properties wird die allgemeine ${USER_HOME}/build.properties überschrieben bzw. erweitert, und Systemregeln nur für dieses Projekt festgeschreiben.

```
<project>
  <groupId>sample</groupId>
  <artifactId>sample-echo</artifactId>
  <version>1.0-SNAPSHOT</version>
  <dependencies>
    <dependency>
      <groupId>log4j</groupId>
      <artifactId>log4j</artifactId>
      <version>1.2.8</version>
    </dependency>
  </dependencies>
  <build>
    <sourceDirectory>src/main/java</sourceDirectory>
    <unitTestSourceDirectory>src/test/java</unitTestSourceDirectory>
    <resources>
      <resource>
        <directory>src/main/resources</directory>
      </resource>
    </resources>
    <unitTest>
      <includes>
        <include>**/*Test.java</include>
      </includes>
    </unitTest>
  </build>
</project>
```

(Beispiel für eine project.xml)

Die grundlegenden Mavenkonfigurationen finden sich in der Datei ${USER_HOME}/.maven. So kann hier das Verzeichnis für die Plugins gespeichert werden, falls dessen Position vom Standardwert ${MAVEN_HOME}/plugins abweicht.

Die Definition eigener Ziele für das Projekt wird in der ${PROJECT_HOME}/maven.xml vorgenommen. Dies ist besonderst von Bedeutung, wenn aus dem Projekt später ein Plugins werden soll oder man lokal Ziele verwenden will.

Lokale Ziele sind Ziele, die nur für dieses Projekt gelten sollen. Es ist jedoch ratsam von der Verwendung lokaler Ziele abzusehen und lieber ein Plugin zu entwickeln, um zukünftig dies bei anderen Projekten zur Verfügung zu haben oder für Andere nutzbar zu machen, denn Probleme die man selbst hat, haben Andere mit Sicherheit auch.

4.3 Gruppenarbeit mit Maven

Maven wurde entwickelt um die Arbeit an gemeinsamen Projekten zu erleichtern und die notwendige Kommunikation zwischen den Entwicklern effzient zu gestalten. Dazu ist es wichtig, nur wirklich notwendige Daten zu verschicken und gemeinsam genutzte Programme auf einem einheitlichen Stand zu halten.

Bei Maven wird dies dadurch erreicht, dass die gemeinsamen Plugins an zentraler Stelle abgelegt werden, wo alle Entwickler sie finden können. Entweder speichert man sie auf einem allgemein zugänglichen Server oder bei Linux-/Unixsystem bindet man das Verzeichnis in seinen Verzeichnissbaum ein.

Es muss so nur einmalig eine project.xml erstellt und diese nur gelegtlich bei einem Versionswechsel aktualisiert werden. So ist es auf einfache Weise möglich eine einheitliche Umgebung zu erstellen. Mit der Verwendung eines Quelltextkontrollprogramms wie etwa CVS gelangt man so zu einer guten Projektumgebung.

Für leitende Entwickler, aber auch anderen sind die Möglichkeiten, die Maven bietet einfach einen umfassenden Überblick über das Projekt zu erhalten, von enormer Bedeutung. Wie schon erwähnt kann man die einzelnen Berichte der Plugins zu einem grossen, zusammensetzten lassen.

Ist ein Projekt in mehrere Unterprojekte zerteilt, wird in dem Hauptverzeichnis des jeweiligen Unterprojektes eine project.xml erstellt, die Maven verdeutlicht, dass es sich dabei um ein neues Projekt handelt. Die Verzeichnisstrukturen müssen dann nur an die Umgebung angepasst werden.

Jedes Projekt muss einen einzigartigen Identifikator erhalten, da nach diesem später die JAR-Datei benannt wird und das Projekt darüber bestimmt wird. Dieser Identifikator setzt sich aus zwei

Komponenten zusammen, einmal der groupId, die die Projektgruppe und der artifactId, die das Projekt innerhalb der Gruppe idenifiziert.

5. Eigene Mavenplugins erstellen

Zum erstellen eigener Plugins, wird die Herangehensweise an ein Mavenprojekt nicht grundlegend geändert, nur einige Faktoren kommen hinzu.

Es müssen eine Reihe von Konfigurationsdateien angelegt werden und wenige zusätzliche Schritte vollzugen werden. Neben den schon erwähnten Dateien project.xml, maven.xml und biuld.properties kommen noch weiter Dateien zum Einsatz. ${PROJECT_HOME}/plugin.jelly beinhaltet Informationen für die Skriptsprache Jelly, die in den Mavenversionen 1.* noch zum erstellen der Plugins verwendet wird. In der kommenden Version 2 soll das Maven intern gelöst werden. In ${PROJECT_HOME}/plugin.properties können Standardwerte für das Plugin gesetzt werden.

```
<project>
  <pomVersion>3</pomVersion>
  <artifactId>hello-maven-plugin</artifactId>
  <groupId>hello</groupId>
  <name>Maven Hello World Plugin</name>
  <currentVersion>1.0</currentVersion>
  <build>
     <!-- Diese Angaben sind pflicht -->
    <resources>
      <resource>
        <directory>${basedir}/src/plugin-resources</directory>
        <targetPath>plugin-resources</targetPath>
      </resource>
      <resource>
        <directory>${basedir}</directory>
        <includes>
          <include>plugin.jelly</include>
          <include>plugin.properties</include>
          <include>project.properties</include>
          <include>project.xml</include>
        </includes>
      </resource>
    </resources>
  </build>
</project>
```

(Beispiel project.xml für ein Plugin)

```
<project xmlns:ant="jelly:ant">
  <goal name="hello:hello" description="Say Hello">
    <ant:echo>Hello from a Maven Plug-in</ant:echo>
  </goal>
</project>
```

(Beispiel plugin.jelly für ein Plugin)

In der maven.xml werden zuerst das Hauptziel und die Nebenziele des Plugins beschreiben. Diese müssen durch die project.xml und Javacode mit „Leben“ gefüllt werden. In der plugin.jelly werden die Ziele des Plugins definiert, diese Datei ist identisch mit der maven.xml in einem normalen Projekt.

Hat man diese beiden Dateien erstellt folgt das Übersetzen des Plugins

```
maven plugin
maven plugin:install
```

So wird eine Plugin-JAR-Datei erstellt und die Ziele installiert. Sie liegt lokal in dem target-Verzeichnis vor und kann sofort benutzt werden. Nun kann getestet werden ob alles problemlos ablief und das Plugin funktioniert.

```
maven -P hello
maven hello:hello
```

Jetzt kann man noch in der plugin.properties Standardwerte definieren.

Als nächstes kann man sich daran machen das Plugin allgemein zugänglich zu machen. Ob diese Weltweit über das Internet oder lokal über ein Netzwerk passiert ist jedem selbst überlassen. Für den lokalen Gebrauch reicht es das Plugin einfach in das Pluginverzeichnis von Maven zu kopieren und eventuell die anderen Entwickler über dieses Plugin zu informieren.

Die weltweite Veröffentlichung kann entweder über eine spezielle maveneigene Umgebung passieren oder über einschlägig bekannte Foren für Open-Source-Projekte, sowie die eigene Webseite und das werben in geeigneten Diskusionsforen. Um ein Paket für das Veröffentlichen im Internet vorzubereiten nutz man diesen Befehl und läd es dann auf einen Server hoch.

```
maven create-upload-bundle
```

6. Zusammenfassung

Zwar gibt es bereits viele Entwicklungsumgebungen mit umfangreichen Möglichkeiten zur Erstellung von Javaprojekten, doch bietet Maven einige interessante Ansätze die bisher noch nicht überall verbreitet sind.

Vorallem die Konzentration auf Unix/Linux typische Möglichkeiten, wie das Nutzen gemeinsamer Verzeichnisse über einfaches Einbinden in den Verzeichnisbaum zeigt an welche Zielgruppe man sich wendet. Wie manche vielleicht schon feststellen konnte braut jeder Unix-/ Liunxbenutzer gern sein eigenes Süppchen, was den Einsatz einheitlicher Entwicklungsumgebungen erschwert. Hier setzt Maven an. Es fordert nicht von den Benutzern ihre gewohnte Umgebung zu verlassen, sondern nur ein einfaches System zu verwenden um diese zu erweitern. Es entspricht auch dem Anspruch Javas Systemübergreifend zu sein.

Das Plugins sich einfach wie ein normales Mavenprojekt entwickeln lassen, trägt zur problemefreieren Anpassung von Maven an zukünftige Anforderungen bei. Ohne lange Umgewöhnung ist man in der Lage neue Ideen umzusetzen und mit anderen zu teilen.

Maven liegt zur Zeit noch in der Version 1 vor. Einige Funktionen werden noch von mavenfremden Programmen bewerkstelligt. Dies soll in zukünftigen Versionen in Maven übernommen werden, um alles unter einer einheitlichen Lizenz zu halten. Bei der Erstelung von Plugins zum Beispiel wird noch die Skriptsprache Jelly verwendet. In Maven 2.x soll dies Verwendung entfallen.

Projektverantwortliche Personen erhalten durch Maven eine Unterstützung einen objektiven und schnellen Überblick über das Projekt zu erhalten. Sie sind nicht mehr alleinig auf die Aussagen der Entwickler angewiesen oder müssen sich durch eine Vielzahl von Ausgabedatein arbeiten um zu erfahren wie der Stand ist und wo es Probleme gibt.

Literatur

Maven ist ein recht neues Projekt und es ist noch keine Literatur vorhanden. Bei O'Reilly erscheint Juli 2005 ein englischsprachiges Buch zum Thema.

Quellen

Webseiten:

http://maven.apache.org (englisch sprachig)