Einführung in Maven und AppFuse Light

© Prof. Christoph Knabe
Beuth-Hochschule Berlin
13.05.2009

Gliederung

- Motivation
- Maven
- Spring
- AppFuse
- Demonstrationen
- Fazit

Motivation

- Projektstruktur für SWP vorgeben
- Build-Prozess (Gesamterzeugung) erleichtern
- Musterarchitektur für Web-Applikationen
- Projekt-Dokumentation und -Metriken

Ziele von Maven

- Vereinfachung des Build-Prozesses
  - Standardsysteme nach Standardverfahren bauen
- Vereinheitlichung des Build-Prozesses
  - Datei `pom.xml` (Projekt-Objekt-Modell) beschreibt, woraus das Projekt besteht und was es benötigt.
  - Verschiedene Plugins können daraus Artefakte generieren.
- Generierung guter Projektdokumentation
  - aus redaktionellen Textstücken und
  - aus dem Quellcode
- Best Practices als Standardmodell
  - Bsp. Phasen generate-sources→compile→test→package→deploy
- Großes, öffentliches Komponentenarchiv

Begriffe in Maven

- POM: Project Object Model
  - Welches Artefakt ist das Build-Ergebnis
  - Aus welchen Artefakten wird es zusammengebaut
  - Welche Plugins wirken dabei mit
  - Welche Reports sollen generiert werden
- groupId, artifactId, version
  - Kompletter Name, um ein Artefakt finden zu können
- Repository
  - Zentraler oder lokaler Speicher für Artefakte
- Lifecycle, Phase
  - Ein Lifecycle hat mehrere Phasen der Ausführung
- Plugin, Goal
  - Ein Plugin bietet mehrere Goals zur Ausführung an
- Dependency
  - Angabe eines Artefakts, welches zum Build benötigt wird

Benutzung in 5 Minuten

- Maven in 5 Minutes
  - http://maven.apache.org/guides/getting-started/maven-in-five-minutes.html
  - Dieses in Eingabeaufforderung nachvollziehen
  - `archetype`-Plugin, `create`-Goal
  - Generierter `src`-Baum
  - Generierte Datei `pom.xml`
  - `mvn package`
  - Generierter `target`-Baum
  - `mvn site`
  - Generierter `target/site`-Baum

Ziele von Spring

- Dependency Injection
- Aspect Oriented Programming (AOP)
- Declarative transaction management
- Persistence accessing templates
- Transactional tests

Dependency Injection

- Programmieren gegen Interfaces
  - Vereinfacht Austausch benötigter Dienste in UserManagerImpl:

```
UserDao dao = new AustauschbarDao();
return dao.getUser(userId);
```

- Dependency Injection (DI)
  - Dienste werden vom DI-Container (Spring) injiziert ⇒ Keine Abhängigkeit des Codes von Dienstimplementierung:

```
@Autowired UserDao dao;
```

- Spring-Managed Services (*Spring Beans*)
  - Klasse stellt sich Spring als Dienst zur Verfügung:

```
@Repository(value = "userDao")
public class UserDaoHibernate implements
UserDao { ... }
```

Ziele von AppFuse [Light]

- Musterarchitektur für Java-Web-Anwendungen:
  - webapp → service → dao → model
- Flexibilität der Oberflächentechnik zeigen:
  - JSF, Spring MVC, Struts 2, Tapestry
- Für/durch Maven vorbereitet:
  - Eclipse-Integration mit Source-Download
  - Testsuite
  - Umfangreiche Reports, z.B.:
    Cobertura Test Coverage,
    Copy Paste Detector (CPD)
    Richtlinienprüfer PMD
    Source Xref (Farbiger und verlinkter Quelltext)
  - Kurzer Testzyklus, Deploy zum Server

Fazit

- Maven+Spring: Mächtiges Ökosystem
- Einstieg aufwändig