KaffeeKlatsch
Das Magazin rund um Software-Entwicklung

ISSN 1865-682X
04/2013

Verflixte Sieben?

Ein Blick auf den JBoss AS 7 – Von Domänen und Modulen

von Werner Eberling

Verflixte Sieben?

Ein Blick auf den JBoss AS 7 – Von Domänen und Modulen

von Werner Eberling

In regelmäßigen Abständen scheint *JBoss* (seit einigen Jahren ein Teil von *RedHat*) die Notwendigkeit zu sehen den gleichnamigen *Application-Server* auf technisch neue Beine zu stellen. Mit Version 7 stand wieder ein solcher *Re-Write* ins Haus. Einige der Neuerungen, die diese neue Version mit sich bringt, sollen in diesem Artikel beleuchtet werden.

Auch ein Applikations-Server ist nur ein Stück Software und wie bei jeder Software neigt auch der Code eines solchen Servers dazu sich zu „entwickeln". Dass dies nicht immer zwingend in die Richtung von statten geht, in die man es sich wünschen würde, wurde den Verantwortlichen bei *JBoss* nach dem *Release* der Version 6 ihres Servers wieder einmal schmerzlich bewusst. Zu spät fertiggestellt, viel zu langsam und mit so mancher Kinderkrankheit behaftet, erhielten sie jede Menge negatives Feedback für Ihre erste Implementierung eines *JEE 6 Servers*[1].

So stand also mit Version 7 wieder einmal eine komplette Neuimplementierung des Servers auf dem Plan. Alte Zöpfe sollten abgeschnitten werden, mit neuen Ideen und Ansätzen der verlorene Boden wieder aufgeholt und der gute Ruf wieder hergestellt werden. Anders als bei der Version 5, bei der sich vor allem die interne Architektur des Servers änderte, bringt die aktuelle Version Neuerungen mit sich, die sich sofort für den Anwender bemerkbar machen.

Installation und Ordnerstruktur

Nach dem Download der aktuellen Version (inzwischen die *7.1.1.Final*) von [1] zeigt sich die Installation noch im bekannt einfachen Stil. Es gilt lediglich das heruntergeladene *ZIP*- oder *TAR.GZ*-Archiv in den Ordner der Wahl zu entpacken. Eine nähere Betrachtung des Ergebnisses lässt aber schon die strukturellen Änderungen im neuen Server erahnen. Die Ordnerstruktur ähnelt nur noch in Ansätzen dem aus früheren Versionen bekannten Aufbau. Unter anderem ist das altbekannte *server*-Verzeichnis komplett verschwunden. An seine Stelle ist der Ordner *standalone* getreten, der mit *domain* dafür auch gleich noch ein neues Geschwisterchen mitbringt. Diese beiden Ordner stehen stellvertretend für die beiden Betriebsmodi, die der aktuelle JBoss-Server anbietet.

Neben den zwei genannten Ordnern, die die Server-Konfigurationen und Daten enthalten, gibt es noch zwei weitere Neulinge, die den bisher bekannten *lib*-Ordner ablösen:

- *bundles*: Hier finden sich die applikationsübergreifend verwendeten Bibliotheken, die als *OSGI-Bundles* eingebunden werden können.
- *modules*: Enthält applikationsübergreifend verwendbare Bibliotheken, die nach dem neuen modularen *Classloading*-Konzept *JBoss Modules* eingebunden werden können.

Von den übrigen bekannten Ordnern haben es nur noch *bin* und *docs* in die neue Version geschafft. Während der erste wie gewohnt Skripte zum Starten und Stoppen des Servers bzw. Hilfsskripte zur Konfiguration enthält, ist im *docs*-Ordner speziell der Unterordner *schema* interessant, da dieser die *XSDs* aller verfügbaren Konfigurationselemente enthält.

Ein *run.sh* wird man im *bin*-Ordner allerdings vergeblich suchen. An dessen Stelle sind die Skripte *standalone.sh* und *domain.sh* getreten. Mit der Wahl eines dieser Skripte entscheidet sich der Anwender für einen der beiden möglichen Betriebsmodi des JBoss 7.

Jeder für sich – der standalone mode

Im *standalone* Modus wird ein einzelner Server-Knoten gestartet. Dies entspricht dem aus früheren Versionen bekannten Verhalten des Servers. Sämtliche Konfigurationen, *Deployments* und Datenverzeichnisse befinden sich unterhalb des *standalone*-Ordners. Anzumerken ist hierbei, dass die Konfiguration nicht mehr in verschiedenen *XML*-Dateien verstreut, sondern in einer XML-Datei pro Server-Profil zusammengefasst ist. Diese, im Unterordner *configuration* abgelegten verschiedenen XML-Dateien (*standalone.xml*, *standalone-full.xml* etc.), sind die Entsprechung der vormals vorhandenen unter-

[1] Der schlechte Ruf des JBoss 6 ging dabei sogar so weit, dass sich JBoss-Offizielle auf öffentlichen Veranstaltungen für diese Serverversion entschuldigten und mit Version 7 Besserung versprachen!

schiedlichen Ordner pro Server-Konfiguration (*default*, *all*, etc.).

Auf den genauen Aufbau der Konfigurationsdatei soll hier nicht im Einzelnen eingegangen werden, da man ihr, dank ihres Umfangs, einen eigenen Artikel widmen könnte. Es sei nur darauf hingewiesen, dass die zugrundeliegenden Schema-Definitionen im Ordner *docs/schema* zu finden sind und diese in vielen Fällen eine wertvolle Hilfe darstellen, wenn es darum geht wie nun genau einem *Web-Container* eine *jvmRoute* gegeben werden kann oder wie ein *Cluster-User* für *JMS* zu konfigurieren ist.

Ein Deployment in eine solche *Standalone*-Instanz ist wie gewohnt über das Kopieren der entsprechenden Artefakte in den *deployment*-Ordner, über die inzwischen *GWT*-basierte *Web-Console* oder über das neue Kommandozeilentool (*bin/jboss-cli.sh*) möglich.

Geschwindigkeit ist keine Hexerei

Hat man sich nun für den *standalone mode* entschieden, so meldet sich der Server wenige Sekunden nach dem Aufruf von *standalone.sh* mit der Meldung:

> [org.jboss.as] (Controller Boot Thread) JBAS015874: JBoss AS 7.1.1.Final "Brontes" started in 8915ms - Started 138 of 213 services (74 services are passive or on-demand)

Dieser schnelle Startvorgang ist einer hohen Parallelisierung innerhalb des Servers zu verdanken. Als Herz des Systems sorgt der *Modular Service Controller (MSC)* dafür, dass nur die benötigten Dienste, sowie deren Abhängigkeiten gestartet werden. Dies geschieht weitgehend parallel, um unnötige Wartezeiten zu vermeiden. Ein *Reflection Cache* und ein optimierter *Annotationsscanner*, der seine Arbeit verrichtet ohne die Klassen über einen *Classloader* laden zu müssen, tun ihr Übriges um Startzeiten und Speicherverbrauch des Servers auf ein Minimum zu beschränken.

Von Haus aus sicher

Möchte man nun auf das gestartete *Standalone*-System über den Browser zugreifen, so geschieht dies über die altbekannte Adresse *http://localhost:8080*. Hier meldet sich der Server mit einer schicken *JBoss 7-Welcome Page* und Links u.a. zur Dokumentation, zu Beispielen für den schnellen Einstieg und zur neuen Administrationskonsole.

Möchte man allerdings auf letztgenannte zugreifen, so scheitert man an der standardmäßig aktiven Sicherheitsbarriere. Ohne einen eingerichteten *Management-User* geht beim neuen JBoss gar nichts. Das passende *Realm* ist bereits eingerichtet und das im *bin*-Verzeichnis mitgelieferte *add_user.sh*-Skript ermöglicht eine schnelle Einrichtung des benötigten Users.

Was für die Management-Konsole ein nachvollziehbares Feature zu sein scheint, wird beim ersten Versuch auf eine *EJB*-Anwendung oder *JMS-Queue* zuzugreifen sehr schnell zum nervigen Hindernis. Auch hier ist standardmäßig der Security-Service aktiv und verlangt diesmal nach einem *Application-User*. Geschickter Weise lassen sich über das oben genannte Skript auch diese User anlegen. Ist dies geschehen, steht dem Aufruf der Anwendung (die richtige *JNDI*-Konfiguration vorausgesetzt) nichts mehr im Wege.[2]

Einfacheres Management dank Domänen

Sollen im Rahmen einer Server-Farm oder eines *Clusters* mehrere Server-Instanzen mit der gleichen Konfiguration und den gleichen Anwendungen versorgt werden, ist der *domain mode* das Mittel der Wahl. Hier werden mehrere Server-Instanzen zu einer logischen Domäne zusammengefasst. Dabei können die Server auf ein und demselben oder verschiedenen Netzwerkknoten angesiedelt sein.

Dieses neue Konzept benötigt ein etwas komplexeres *Setup*. Hier wird über das Skript *domain.sh* nicht die eigentliche Server-Instanz gestartet, sondern lediglich ein sogenannter *Host Controller*. Dieser steuert, auf jedem physischen Knoten, das Starten und Stoppen der einzelnen Server-Instanzen, die für eine einheitliche Konfiguration in logischen Gruppen (*server groups*) zusammengefasst sind.

Um das Management zu vereinfachen, wird einer der Knoten als *Domain Controller* konfiguriert. Dieser hält die zentrale Konfiguration der Domäne vor und verteilt diese, wie auch anstehende Deployments, an die anderen in der Domäne beteiligten Knoten. Adressaten für Konfigurationen und Deployments sind dabei immer die bereits angesprochenen Server-Gruppen. Somit bildet der *Domain Controller* eine zentrale Konfigurationsinstanz und erspart das händische „nachziehen“ der einzelnen Applikations-Server-Instanzen.[3] (siehe Abbildung 1)

Domain mode – Konfigurationsdetails

Durch ihren komplexeren Aufbau kann die Domäne leider nicht direkt, ohne weitere Vorarbeit, gestartet werden.

---

[2] Hierbei ist allerdings zu beachten, dass das neu zum Einsatz kommende *JBoss Remoting* eine eigene, proprietäre(!) Konfigurationsdatei verlangt [2].

[3] Das Domänenkonzept darf hier nicht mit dem Aufbau eines Server-*Clusters* verwechselt werden. Es handelt sich lediglich um einen Zentralisierungsansatz zur Vereinfachung der Konfiguration. Die Definition eines *Clusters* und die Auswahl des Betriebsmodus sind zueinander orthogonale Konzepte!

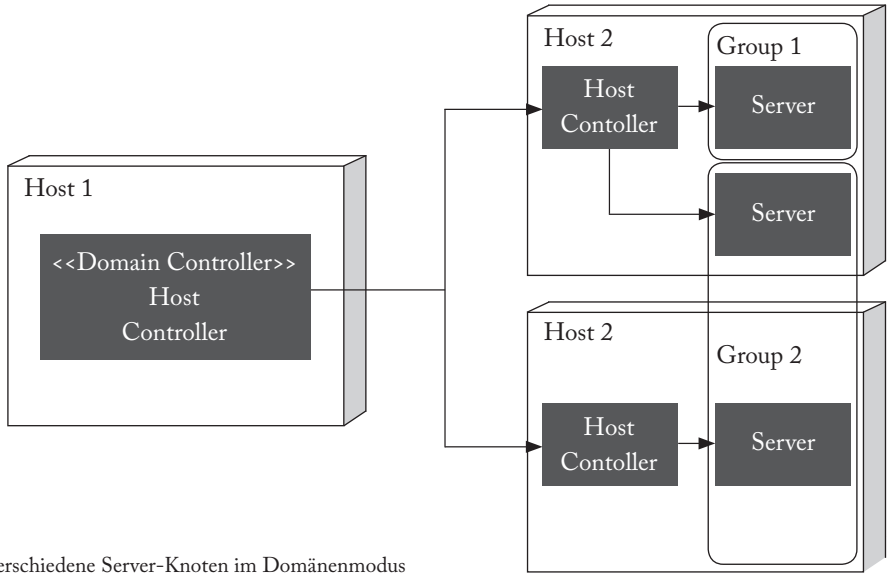

Abbildung 1: Verschiedene Server-Knoten im Domänenmodus

Vorab müssen die Details der einzelnen *Host Controller* im *host.xml* des entsprechenden Knotens konfiguriert werden. Dort werden die Netzwerkverbindungen der jeweiligen Knoten sowie die Verbindung zum *Domain Controller* eingerichtet.

Damit sich nicht jeder Server einfach in die Domäne einklinken kann, ist vorab ein *Management User* auf dem *Domain Controller* anzulegen. Mit diesem User melden sich die abhängigen *Slave*-Knoten bei ihrem *Domain Controller* an, um dort die notwendigen Konfigurationen abzurufen.

Nachdem die Verbindung zum *Domain Controller* eingerichtet ist, bleibt noch die Definition der auf jedem Knoten verfügbaren Server-Instanzen und deren Organisation in Server-Gruppen. Dies geschieht ebenfalls im bereits erwähnten *host.xml*. Ist all dies geschehen, kann erst der *Domain Controller*-Knoten und anschließend jeder *Slave*-Knoten mit *bin/domain.sh* gestartet werden. Alles weitere handeln die Knoten untereinander aus:

```
[Host Controller] 10:23:46,784 INFO  [org.jboss.
as.domain] (domain-mgmt-handler-thread - 1)
JBAS010918: Registered remote slave host "slave",
JBoss AS 7.1.1.Final "Brontes"
```

Die eigentliche Konfiguration der Applikations-Server-Instanzen wird über die Datei *domain.xml* vorgenommen. Anders als im *standalone mode*, finden sich hier alle Konfigurationsprofile (*default*, *full*, *full-ha*, etc.) in einer einzigen XML-Datei, die zentral auf dem *Domain Controller* vorgehalten wird.

JBoss Modules

Ein weiteres neues Konzept, das mit der Version 7 eingeführt wurde, ist das modulare *Classloading* auf Basis von *JBoss Modules*. Waren in bisherigen Servern die einzelnen *Classloader* hierarchisch angeordnet, so verwendet JBoss Modules einen modularen Ansatz. Hier wird für alle Module je ein eigener Classloader erzeugt, die durch explizit definierte Abhängigkeiten miteinander verknüpft werden.

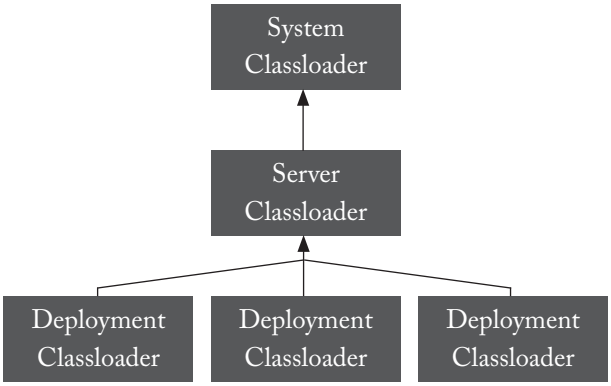

Abbildung 2: Hierarchisches Classloading

Dieser modulare Ansatz ermöglicht nicht nur ein Höchstmaß an Isolierung der Anwendungen, durch die Versionierung der Module ist es möglich die gleichen Bibliotheken in verschiedenen Versionen zentral im Server vorzuhalten und somit die Deployments der einzelnen Applikationen deutlich zu verschlanken (siehe Abbildung 3).

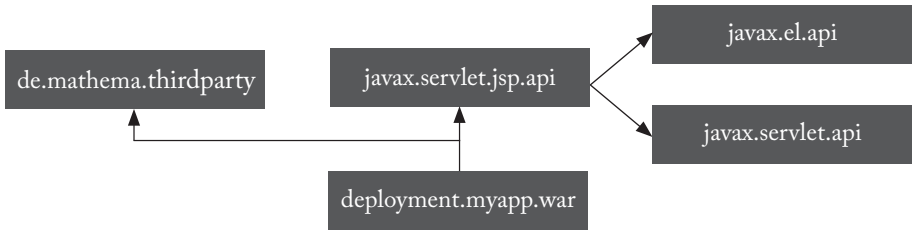

Abbildung 3: Modulares Classloading mit JBoss Modules

Definition von Modulen

Module werden im Verzeichnis *modules* abgelegt. Die Organisation erfolgt dabei hierarchisch nach dem Modulnamen, der sich an einer paketartigen Gruppenstruktur, wie sie beispielsweise aus *Maven* bekannt ist, orientiert. Zusätzlich wird ein Moduldeskriptor (*module.xml*) benötigt, der die nötigen *JAR*-Dateien und evtl. transitive Abhängigkeiten des Moduls beschreibt.

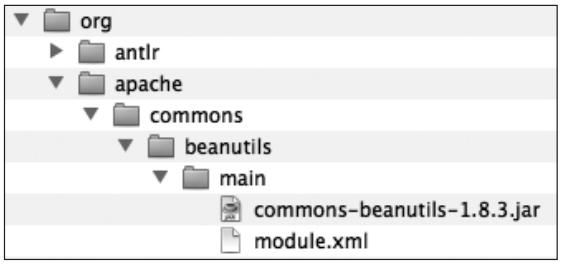

Abbildung 4: *Commons-beanutils* als JBoss Module

```
<module xmlns="urn:jboss:module:1.1" name=
  "org.apache.commons.beanutils">
  <resources>
    <resource-root path="commons-beanutils-1.8.3.jar"/>
    <!-- Insert resources here -->
  </resources>
  <dependencies>
    <module name="org.apache.commons.logging"/>
    <module name="org.apache.commons.collections"/>
  </dependencies>
</module>
```

module.xml des Moduls org.apache.commons.beanutils

Jedes Modul stellt mindestens eine Hauptversion im Unterverzeichnis *Main* zur Verfügung. Beliebige weitere Versionen (im JBoss-Jargon: *Slots*) sind dabei ebenfalls möglich.

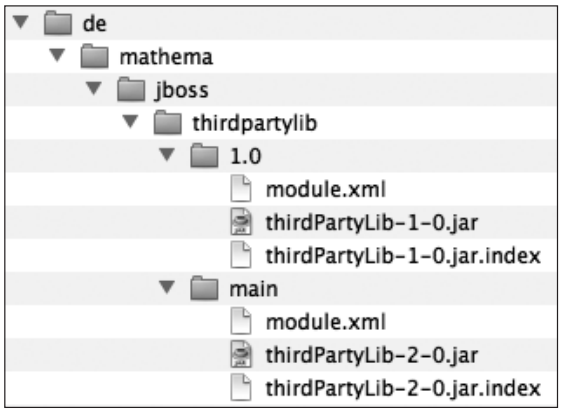

Abbildung 5: Zwei Versionen der selben Bibliothek

Explizite Abhängigkeiten in den Deployments

Während des Deployment-Vorgangs wird jede Anwendung ihrerseits zu einem *Classloading*-Modul. Die Abhängigkeiten für dieses Deployment-Modul ergeben sich dabei einerseits aus der Container-Umgebung (d.h. *JEE*-Abhängigkeiten werden implizit mit eingebunden) und andererseits aus im Deployment explizit angegebenen, benötigten Modulen. Die Definition dieser Abhängigkeiten erfolgt entweder in der Manifest-Datei des Deployments oder in einem JBoss-spezifischen Deployment-Deskriptor.

```
Dependencies: de.mathema.jboss.thirdpartylib:1.0,
org.apache.commons.beanutils
```

Abhängigkeitsdefinition im Manifest

Wird bei der Abhängigkeitsdefinition kein *Slot* (i.e. keine Version) angegeben, so wird der *Main Slot* verwendet. Die dort enthaltene Version des Moduls ist sozusagen die Standardversion, die im Server verwendet werden soll.

Schöne neue JBoss 7-Welt?

Mit JBoss Modules und dem Domänen-Konzept wurden in diesem Artikel die beiden Neuerungen der aktuellen

JBoss-Version vorgestellt, die dem Anwender sicher als erstes ins Auge stechen werden. Beide bieten neue Möglichkeiten, erfordern aber auch ein Umdenken.

Wo es beim Domänen-Konzept noch die „klassische" Alternative mit dem immer noch unterstützten *Standalone Mode* gibt, bleibt dem Anwender beim neuen modularen Classloading nichts anderes übrig, als sich damit anzufreunden. Hier bleibt aber festzuhalten, dass diese Art der expliziten Definition der Abhängigkeiten zu einem deutlich sauberen Deployment-Aufbau und damit zu einer besseren Applikatonsstruktur führt.

Aufwand der sich lohnen könnte

Nichtsdestotrotz ist bei einem Umstieg von JBoss 4 – 6 auf JBoss 7 mit größeren Aufwänden zu rechnen, als es noch bei den Vorgängerversionen der Fall war. Aber das ist ein Thema für einen eigenen Artikel und sollte niemanden davon abhalten, sich den aktuellen JBoss-Server anzusehen und ggf. auf diesen umzusteigen. In diesem Zusammenhang sei dem geneigten Leser die JBoss AS 7.1-Dokumentation [3] als umfangreiche Informationsquelle ans Herz gelegt.

Referenzen

[1] JBoss *Application Server Downloads*
http://www.jboss.org/jbossas/downloads

[2] JBoss *AS 7.1 - EJB invocation from a remote client*
https://docs.jboss.org/author/display/AS71/EJB+invocations+from+a+remote+client+using+JNDI

[3] JBoss *AS 7.1 Dokumentation*
https://docs.jboss.org/author/display/AS71/Documentation

Kurzbiografie

Werner Eberling ist Principal Consultant und Technical Lead bei der MATHEMA Software GmbH in Erlangen. Seit 1999 beschäftigt er sich mit verteilten Systemen, sein Fokus liegt dabei auf CORBA und der JEE. Neben seiner Projekttätigkeit hält er Technologie-Trainings und ist als Sprecher auf nationalen und internationalen Konferenzen anzutreffen. Sein aktuelles Steckenpferd ist die Anwendungsentwicklung für mobile Endgeräte bzw. deren Anbindung an Enterprise Systeme. Weiterhin ist er u. a. Autor des Buches „Enterprise JavaBeans 3.1 Das EJB-Praxisbuch für Ein- und Umsteiger", erschienen im Hanser Verlag.