Der Tod der Java Application Server: Sind sie noch zeitgemäß? ▸55

S&S

JAVA Mag

Deutschland €9,80 Österreich €10,80 Schweiz sFr 19,50 Luxemburg €11,15 7.2014

Java magazin

Java • Architekturen • Web • Agile

www.javamagazin.de

RoboVM
iOS-Apps mit Java entwickeln ▸115

ServiceMix
Schlanke Systemintegration ▸91

CLOJURE

Quo vadis Clojure 1.6 ▸14

Zugriff auf relationale Datenbanken ▸32

Polyglotte Entwicklung mit Clojure und Java ▸24

Fieldings Vermächtnis: Wer REST will, muss mit HATEOAS ernst machen ▸64

IoT mit Java 8 und Tinkerforge: Baukasten für große Kinder ▸38

Mächtig flexibel: Konzepte des Ceph-Storage-Clusters ▸103

4 194586 709801 07

Webvisualisierung im Zusammenspiel mit Pax Web

Webanwendungen mit Apache Karaf

Im dritten Teil dieser Artikelserie über Apache Karaf wird ein weiterer Aspekt der Enterprise-Features vorgestellt. Aufbauend auf dem letzten Artikel, der den grundsätzlichen Aufbau einer JPA-Anwendung gezeigt hat, geht es heute darum, den im letzten Artikel erzeugten Service zu verwenden, um in einer Webanwendung die Entitäten darzustellen.

von Achim Nierbeck

Auch in diesem Artikel stehen vor allem die von Apache Karaf mitgebrachten Funktionalitäten im Vordergrund. Daher ist das Design der Anwendung eher einfach und schlicht gehalten.

Karaf vorbereiten für das Web

Wie bereits im letzten Artikel muss Apache Karaf für den Einsatz in Enterprise-Anwendungen vorbereitet werden. Damit der Benutzer sich aber nicht selbst in eine Installationsarie stürzen muss, gibt es einfach zu installierende Features zum Aufsetzen der entsprechenden Umgebung. So wird Apache Karaf mit der Installation des *war*-Features zum Webcontainer. Hierzu den Karaf-Container mit *bin/karaf* starten und im Anschluss das Feature *war* installieren.

Durch die Installation des *war*-Features werden dreißig weitere Bundles installiert. Dieses Feature baut auf drei anderen Features auf, die automatisch mitinstalliert werden. Das *http*-Feature bringt alles Nötige für den Basis-OSGi-HTTP-Service mit. Aufbauend auf dem *http*-Feature installiert das *http-whiteboard*-Feature einen erweiterten HTTP-Service (*WebContainer*) und dazu passend einen Whiteboard Pattern Extender. Dieser Extender lauscht auf als Services installierte Servlets, Filter, EventListener, Ressourcen etc., um diese dann über den *WebContainer*-Service zu registrieren.

Zum Visualisieren der Daten per JPA aus dem vorhergehenden Artikel erzeugen wir ein neues Webprojekt, das am Ende ein OSGi Web Application Bundle (WAB) mit annotierten Servlets enthält. Da wir auch hier wieder Maven für das Erstellen verwenden, hilft uns an dieser Stelle wieder das *maven-bundle-plugin* zum Aufbauen eines OSGi-fähigen *war*-Archivs.

„war“ als Bundle aufbauen

Möchte man ein Maven-Projekt als OSGi Bundle erstellen, kommt man am *maven-bundle-plugin* des Apache-Felix-Projekts [1] nicht mehr vorbei. Dieses Projekt verwendet intern die von Peter Kriens entwickelten Bndtools, die eine Bytecodeanalyse des gerade erstellten Bytecodes durchführen. Die so erhaltenen Informationen ergeben unter Zuhilfenahme des Konfigurationsteils des *maven-bundle-plugin* die notwendigen Inhalte für das Erstellen des *Manifest.mf*.

Für unser Beispiel ist es erst einmal nicht notwendig, irgendwelche Import-Package- oder Export-Package-Anweisungen zu geben, da diese automatisch berechnet werden. Hier ist in aller Regel meist nur noch das Export-Package einzuschränken, damit nicht alle Packages exportiert werden. Wichtig für unser *war*-Archiv ist es,

Artikelserie

Teil 1: Grundlagen und Neuigkeiten
Teil 2: Aufbau einer JPA-Anwendung
Teil 3: Webvisualisierung im Zusammenspiel mit Pax Web

Starten von Karaf

```
        __ __                  ____
       / //_/ ____ __________/ __ /
      / ,<  / __ `/ ___/ __ `/ /_
     / /| |/ /_/ / /  / /_/ / __/
    /_/ |_|\__,_/_/   \__,_/_/

  Apache Karaf (3.0.0)

Hit '<tab>' for a list of available commands
and '[cmd] --help' for help on a specific command.
Hit '<ctrl-d>' or type 'system:shutdown' or 'logout' to shutdown Karaf.

karaf@root()> feature:install war
```

den Web-*ContextPath* anzugeben (Listing 1), denn ohne diesen kann ein OSGi WAB nicht deployt werden. Im Vergleich zu einem klassischen Webcontainer entspricht dieser Pfad dem extrahierten *war*-Namen. Dieses Verzeichnis dient dann immer auch als Wurzelverzeichnis für Webanfragen, da es im OSGi-Kontext keinen *deploy*-Ordner gibt, in dem ein extrahiertes *war* liegt.

Besonders wichtig für das Erstellen des *WebApplicationBundles* (WAB) ist es, dem *maven-bundle-plugin* das Verzeichnis mitzuteilen, das als Wurzelverzeichnis innerhalb des WABs fungiert. Die <*_wab*>-Anweisung übernimmt diese Funktion für das *maven-bundle-plugin*.

Das Ergebnis ist ein WAB, allerdings auch nur ein Bundle. Es ist also explizit kein hybrides *war*, das sowohl auf einem Standardwebcontainer als auch einem OSGi-Container lauffähig ist. Dazu werden noch weitere Einstellungen benötigt, wie sie in Listing 2 zu sehen sind.

Zum einen wird im *maven-bundle-plugin* noch die Unterstützung für *war* als Projekttyp konfiguriert, zum anderen wird durch das Setzen der *execution* konfiguriert, dass es nur ein Manifest erzeugen soll. Alles Weitere übernimmt dann wieder das *war*-Plug-in, das das nun erzeugte Manifest mit einbindet. Details dazu können den Sourcen zu diesem Beispiel unter [2] entnommen werden. Schlussendlich haben wir nun ein Webprojekt, das uns bequem *war*-Archive inklusive OSGi-Manifest erzeugt.

Rendern der JPA-Anwendung

Unser Ziel ist es nun, die im vorigen Artikel erstellte JPA-Anwendung über eine Weboberfläche auszugeben. Da wir damit aber keinen Schönheitspreis gewinnen, sondern uns nur auf die verwendeten Technologien im Einsatz mit Apache Karaf konzentrieren wollen, geht es jetzt erst einmal darum, ein Servlet für die Darstellung der Daten aufzubauen.

Wie man Listing 3 unschwer entnehmen kann, ist das Servlet für einen modernen Entwickler, der sich immer auf die neuesten Technologien stürzt, nicht sonderlich spektakulär. Allerdings ergibt sich an dieser Stelle doch eine Besonderheit. Denn schaut man das Servlet genauer an und legt dabei nochmal die OSGi-Spezifikation bzgl. des HTTP-Service daneben, dann steht in der OSGi-Spezifikation, dass mindestens Servlets in API-Version 2.1 unterstützt werden. Mit Apache Karaf 3 und dem verwendeten Pax Web 3 ist es nun möglich, Servlet 3.0 Annotations zu nutzen und damit sogar auf alle weiteren *war*-Eigenschaften wie *web.xml* zu verzichten und ein rein mit annotierten Servlets versehenes *jar* zu deployen.

PAX CDI installieren

Jetzt haben wir zwar ein Version-3-annotiertes Servlet in unserem Projekt, aber immer noch keine Möglichkeit, auf die Entitäten und den Dienst zur Persistierung zuzugreifen. Welche Möglichkeiten gibt es denn nun, aus einem WAB auf andere OSGi-Dienste zuzugreifen? Zum einen gibt es die Möglichkeit der Ansprache über JNDI. Dies setzt allerdings voraus, dass Aries JNDI installiert ist. Der Aufruf über diesen Weg ist bereits im vorhergehenden Artikel beschrieben, um auf die *DataSource* durch den *EntityManager* zuzugreifen. Zum anderen ist es möglich, die Servlets per Blueprint zu instanziieren und hierbei die Referenzen auf äußere Dienste injizieren zu lassen. Diese Variante setzt allerdings voraus, dass die Servlets nicht annotiert, sondern tatsächlich auch in der *blueprint.xml* konfiguriert sind. Die in diesem Fall eleganteste Lösung setzt auf Pax CDI auf. Dieses Framework aus der OPS4j-Familie (Open Participation for Java) bringt Contexts und Dependency Injection in die OSGi-Welt.

Listing 1

```
<instructions>
  <_wab>src/main/webapp</_wab>
  <Web-ContextPath>javamagazin</Web-ContextPath>
</instructions>
```

Listing 2

```
<plugin>
  <groupId>org.apache.felix</groupId>
  <artifactId>maven-bundle-plugin</artifactId>
  <version>2.4.0</version>
  <extensions>true</extensions>
  <executions>
    <execution>
      <id>bundle-manifest</id>
      <phase>process-classes</phase>
      <goals>
        <goal>manifest</goal>
      </goals>
    </execution>
  </executions>
  <configuration>
    <supportedProjectTypes>
      <supportedProjectType>jar</supportedProjectType>
      <supportedProjectType>bundle</supportedProjectType>
      <supportedProjectType>war</supportedProjectType>
    </supportedProjectTypes>
    <instructions>
      <_wab>src/main/webapp</_wab>
      <Web-ContextPath>javamagazin</Web-ContextPath>
    </instructions>
  </configuration>
</plugin>
```

Listing 3

```
@WebServlet(name = "JavaMagazinVisualizer", urlPatterns = "/visualize")
public class VisualizerServlet extends HttpServlet {
  @Override
  protected void doGet(final HttpServletRequest request,
    final HttpServletResponse response) throws ServletException {
    // TODO: this is where the magic happens
  }
}
```

Für den Einsatz in Karaf gibt es wieder ein Feature, welches das Zusammensuchen von Abhängigkeiten und die entsprechende Fehlersuche reduziert. Durch ein beherztes *feature:install pax-cdi-web-openwebbeans* werden dabei 24 weitere Bundles installiert. So wird es möglich, CDI im OSGi Context zu verwenden. Damit aber auch das aktuelle Bundle vom CDI Extender als CDI Bundle erkannt wird, ist es notwendig, das Bundle als CDI Bundle zu markieren. Hierbei setzt Pax CDI vor allem auf die neue OSGi-5-Möglichkeit *Require-Capabilities*, um sicherzustellen, dass CDI verwendet wird. Die in Listing 4 gezeigten Anpassungen sind notwendig, damit das generierte Manifest alle Einträge enthält.

Das resultierende Manifest sieht dann wie in Listing 5 aus. Wichtig sind die beiden Einträge zu *Pax-ManagedBeans* und *Require-Capabilities*. Der Eintrag *Pax-ManagedBeans* zeigt auf das Verzeichnis, das die *beans.xml* enthält. In unserem Fall reicht es, eine leere Datei dort abzulegen. Allerdings gibt es eine Einschränkung mit Pax CDI bezüglich des Web Application Bundles: Es muss tatsächlich ein Bundle sein und kein *war*-Archiv.

Nach all diesen Vorarbeiten kann endlich auch der OSGi Service zum Ansprechen der Datenbank benutzt werden. Dazu muss das Servlet um diesen Dienst und seine Injektion erweitert werden, wie in Listing 6 zu sehen.

Durch den injizierten *entityBroker* kann jetzt auf die Objektstruktur zugegriffen werden. Mit den im vorigen Artikel gezeigten Karaf-Shell-Kommandos können die Daten nun angelegt werden, um sie über die Weboberfläche anzuzeigen. Natürlich ist es auch möglich, über die Weboberfläche eine Pflegeapplikation für die Daten zu erstellen. Diese Fingerübung überlasse ich aber dem ambitionierten Leser.

Listing 4

```
<configuration>
 <instructions>
  <_wab>src/main/webapp</_wab>
  <Web-ContextPath>javamagazin</Web-ContextPath>
  <Pax-ManagedBeans>WEB-INF/beans.xml</Pax-ManagedBeans>
  <Require-Capability>
   org.ops4j.pax.cdi.extension;
   filter:="(&amp;(extension=pax-cdi-extension)(version&gt;=${version;==;${pax.cdi.
     osgi.version.clean}})(!(version&gt;=${version;=+;${pax.cdi.osgi.version.clean}})))",
   osgi.extender; filter:="(osgi.extender=pax.cdi)"
  </Require-Capability>
 </instructions>
</configuration>
```

Listing 5

```
Manifest-Version: 1.0
Archiver-Version: Plexus Archiver
Created-By: Apache Maven Bundle Plugin
Built-By: anierbeck
Build-Jdk: 1.7.0_17
Bnd-LastModified: 1394879134880
Bundle-ManifestVersion: 2
Bundle-Name: Karaf-Web-Sample
Bundle-SymbolicName: de.inovex.javamagazin.Karaf-Web-Sample
Bundle-Version: 1.0.0.SNAPSHOT
Import-Package: javax.servlet;version="[3.0,4)",javax.servlet.annotati
 on;version="[3.0,4)",javax.servlet.http;version="[3.0,4)"
Pax-ManagedBeans: WEB-INF/beans.xml
Require-Capability: org.ops4j.pax.cdi.extension;filter:="(&(extension=
  pax-cdi-extension)(version>=0.6)(!(version>=0.7)))",osgi.extender; fi
  lter:="(osgi.extender=pax.cdi)"
Tool: Bnd-2.1.0.20130426-122213
Web-ContextPath: javamagazin
```

Listing 6

```
@WebServlet(name = "JavaMagazinVisualizer", urlPatterns = "/visualize")
public class VisualizerServlet extends HttpServlet {
  @Inject
  @OsgiService
  InventoryEntityBroker entityBroker;
  @Override
  protected void doGet(final HttpServletRequest request,
    final HttpServletResponse response) throws ServletException {
    List<InventoryItem> allItems = entityBroker.getAllItems();
    // TODO: this is where the magic happens
  }
}
```

Fazit

Mit Apache Karaf lassen sich schnell und einfach moderne Enterprise-Anwendungen erstellen. Unterstützung findet man hier bei den eingebauten Hilfestellungen sowie den zahlreichen Kommandos: z. B. JDBC zum Erstellen von Data Sources oder den zahlreichen Features, die die Installation von eigenen Applikationen deutlich vereinfachen.

Aber mit Apache Karaf ist noch mehr möglich. So gibt es bereits zwei Subprojekte, die Apache Karaf deutlich erweitern: zum einen Apache Karaf Cellar, welches Karaf nochmals um Cluster-Funktionalitäten erweitert. So können mit Cellar, Bundles, Features und Konfigurationen im Cluster verteilt, aber auch Dienste im Cluster per DOSGi angesprochen werden. Und zum anderen das zweite Subprojekt Apache Karaf Cave. Mit Cave wird es möglich, einen Karaf-Server als OSGi Bundle Repository (OBR) einzusetzen – gerade im Zusammenhang mit Cellar eine durchaus interessante Kombination.

Achim Nierbeck ist Seniorentwickler/-architekt/-Scrum-Master bei der inovex GmbH und beackert das Feld der EE-Entwicklung schon seit über zwölf Jahren. Seit nunmehr drei Jahren beschäftigen ihn Apache Karaf PMC OSGi und vor allem das Thema Karaf.

Links & Literatur

[1] http://felix.apache.org/site/apache-felix-maven-bundle-plugin-bnd.html

[2] https://github.com/ANierbeck/Karaf-Web-Sample.git

[3] http://karaf.apache.org/

[4] https://ops4j1.jira.com/wiki/display/paxweb/Pax+Web

[5] https://ops4j1.jira.com/wiki/display/PAXCDI/Pax+CDI