trivadis
Part of Accenture

Guten Morgen Geronimo

AUTOR

Kristian Köhler
Orientation in Objects GmbH

Veröffentlicht am: 18.1.2005

ABSTRACT

Das Apache Geronimo Projekt hat das Ziel, einen zertifizierten J2EE Application Server unter der Apache Software License zur Verfügung zu stellen. Dies soll über die Integration verschiedener OpenSource Projekte erreicht werden. Die wesentliche Neuimplementierung stellt der Geronimo Kernel dar, der eine Plug-in Architektur zur Verfügung stellt, mit der verschiedene Komponenten eingebunden werden können. Jede dieser Komponenten deckt einen einzelnen J2EE Spezifikationsbereich ab. Auf diese Weise soll der komplette J2EE Stack bereitgestellt werden. Dieser Artikel begrüßt den neuen Server und stellt ihn seinen Lesern kompakt vor.

) Schulung)

) Beratung)

) Entwicklung)

) Artikel)

Trivadis Germany GmbH

Weinheimer Str. 68
D-68309 Mannheim

Tel. +49 (0) 6 21 - 7 18 39 - 0
Fax +49 (0) 6 21 - 7 18 39 - 50

www.oio.dekontakt@trivadis.com

ALLGEMEINES ZU GERONIMO

JOnAS und JBoss stehen beide unter einer GPL (GNU General Public License) abgeleiteten Lizenz, was deren Integration in eigene nicht OpenSource Projekte verhindert. Im Gegensatz dazu ermöglicht die Apache Lizensierung von Geronimo die Integration des Servers in beliebigen kommerziellen Produkten. Zur Zeit ist kein anderer J2EE zertifizierter OpenSource Application Server verfügbar, der eine vergleichbare Lizensierung besitzt.

Unter den prominentesten OpenSource Projekten, die bereits in Geronimo integriert sind befinden sich:

- Jetty (WebContainer)
- OpenEJB (EJB Implementierung)
- ActiveMQ (JMS)
- Tranql (Persistenz)
- JOTM (Transaktionsmanagement)

Durch die bereitgestellte Plug-in Architektur ist es außerdem möglich beliebige Java Komponenten einzubinden. Geronimo bietet so zum Beispiel Unterstützung für Axis oder das Spring Framework an.

KONFIGURATIONEN

Die Konfiguration einzelner Serverinstallationen sowie deren Reproduzierbarkeit ist oft problematisch. Insbesondere bei vielen Serverinstanzen, die mit gleicher Konfiguration ausgeführt werden sollen, können kleinere Unterschiede fatale Folgen haben. Diese Problematik löst Geronimo durch einen eigenen Konfigurationsmechanismus. Mit ihm wird eine leichte Skalierbarkeit von kleinen Plattformen bis zu großen Rechnerinstallationen erreicht.

Basis für den Konfigurationsmechanismus sind die sogenannten GBeans (siehe weiter unten). Sie stellen die eigentlichen deploybaren Komponenten innerhalb Geronimo dar. Jeder Dienst besteht aus einem oder mehreren GBeans, die individuell konfiguriert werden können, sowie aus Beziehungen zwischen diesen. Die GBeans werden innerhalb eines Deploymentvorgangs zu Konfigurationen zusammengestellt und gepackt. Jedes dieser Archive enthält entweder Referenzen auf benötigte Komponenten oder enthält diese selbst. Somit ist jede Konfiguration für sich betrachtet eigenständig. Die so erzeugten Konfigurationen können anschließend ausgeführt bzw. verteilt werden. Konfigurationen können noch zur Laufzeit angepasst und wieder im sogenannten ConfigurationStore abgespeichert werden.

Im ConfigurationStore befindliche Konfigurationen werden nicht zwangsläufig sofort im Server ausgeführt. Über den Deployment Mechanismus können sie zur Laufzeit gestartet bzw. auch wieder gestoppt werden. Dies ermöglicht die Auslieferung verschiedener Konfigurationen die, je nach Bedarf, aktiviert werden können.

ARCHITEKTUR

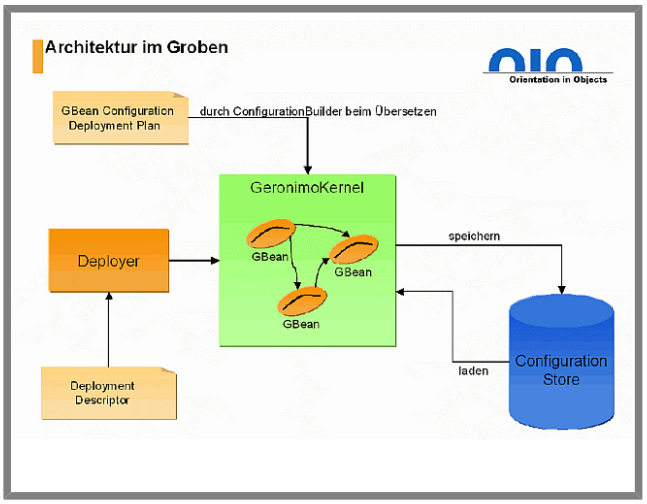

Abbildung 1: Gernonimo - Architektur Überblick

Die Eckpfeiler der Geronimo Architektur sind die schon oben erwähnten GBeans. Sie repräsentieren Verwaltungseinheiten, die innerhalb des Servers ausgeführt und verwaltet werden. Dienste wie z. B. das Lifecyle Management(JSR77) setzt an diesen kleinsten zu verwaltenden Einheiten an. Der Container ruft, dem Inversion of Control (IoC) Gedanken entsprechend, Methoden an ihnen auf.

Mit einer "Dependency Injection" werden Referenzen zwischen den Komponenten aufgelöst. Besitzt ein GBean eine Referenz zu einem anderen GBean, so wird diese Referenz beim Starten des Beans in Form eines Proxy Objektes von der Laufzeitumgebung gesetzt. Eine generische und somit (laufzeit-)fehleranfällig Variante über die Java Management Extensions (JMX), wie sie in anderen Servern zum Einsatz kommt, entfällt hierbei. Innerhalb des Servercodes kann typsicher programmiert werden.

```
server.invoke( new
ObjectName("oio.beispiel:name=BeispielDynamic"),
           "print",
           new Object[]{"Hallo"},
           new String[]{"java.lang.String"} );
```

Beispiel 1: Generischer Methodenaufruf mittels JMX

Beispiel eines Methodenaufrufes eines referenzierten Objektes innerhalb Geronimo über ein Proxy Objekt:

```
proxyObject.print("Hallo");
```

Beispiel 2: Methodenaufruf Proxy Objekt

Die Java Management Extensions (JMX) spielen mit Ihren Managable Beans (MBeans) in der Architektur des Kerns trotz alledem eine zentrale Rolle. Die einzelnen GBeans werden als MBeans an einem MBeanServer angemeldet und können somit verwaltet werden. Das GBean Framework unterstützt JMX, basiert allerdings nicht darauf.

Innerhalb des Deployment Descriptors werden die Referenzen zwischen den GBeans mit Hilfe von JMX ObjectNames konfiguriert.

```
...
<reference name="SimpleBean">
 de.oio.geronimo:type=SimpleGBean
</reference>
...
```

Beispiel 3: JMX ObjectNames

Im Zielbean muß eine entsprechende Methode vorhanden sein, wobei der Referenzname und der Attributname im Bean übereinstimmen müssen.

```
public void setSimpleBean(SimpleBean simpleBean) {
   simpleBean.print("Hallo");
}
```

Beispiel 4: Methode in der Zielbean

Die Proxy Objekte werden zur Laufzeit als Java-Bytecode einmalig über die CGLIB Bibliothek erzeugt und versprechen im Folgenden eine höhere Performance wie die Reflection basierten Dynamic Proxies aus dem Java Standardumfang.

INSTALLATION UND START DES SERVERS

Im Folgenden soll gezeigt werden, wie man den Server installieren und ausführen kann.

Wie bei vielen anderen Produkten bekannt besteht die Installation nur aus dem Entpacken eines Archives in ein Verzeichnis. Die aktuelle Version finden Sie unter: http://geronimo.apache.org/download.html. Im Folgenden wird davon ausgegangen, daß die Version 1.0-M3 eingesetzt wird.

Entpacken Sie den Server in ein Verzeichnis auf Ihrer Festplatte und wechseln Sie in das bin Verzeichnis der Installation. Durch Ausführen folgenden Befehls läßt sich der Server starten:

```
java -jar server.jar
org/apache/geronimo/DebugConsole
```

Die Angabe 'org/apache/geronimo/DebugConsole' bezeichnet hierbei die zu startende Konfiguration. Weitere Konfigurationen können durch Leerzeichen getrennt angegeben werden. Die DebugConsolen Konfiguration wird im Standardumfang mit ausgeliefert und stellt eine rudimentäre Debug Konsole unter http://localhost:8080/debug-tool/ zur Verfügung, mit der alle Server Dienste überwacht werden können.

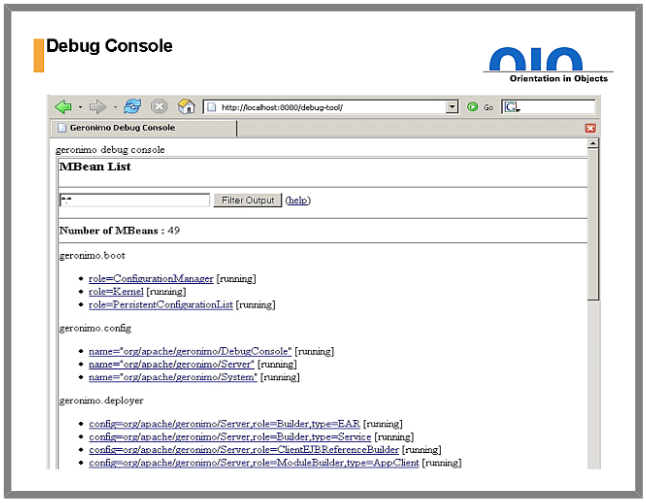

Abbildung 2: Debug Console

Einzelne Dienste können ebenfalls über Kommandozeile zur Laufzeit gestoppt bzw. wieder gestartet werden (Benutzername: system, Passwort: manager).

```
java -jar bin/deployer.jar stop
org/apache/geronimo/DebugConsole
```

```
java -jar bin/deployer.jar start
org/apache/geronimo/DebugConsole
```

GBEAN IN ACTION

Im Rahmen des folgenden Beispiels wird ein GBean implementiert, daß im Server ausgeführt werden soll. Es handelt sich um ein normales POJO (Plain Old Java Object), das mit einer Geronimo-eigenen Managementschnittstelle ausgestattet ist. Dieses Bean soll innerhalb einer eigenen Konfiguration im Server zur Verfügung gestellt werden.

Die Klasse SimpleBean verfügt über ein Attribut und eine Methode, die über die Managementschnittstelle veröffentlicht werden sollen. Hierzu wird die statische Methode getGBeanInfo() implementiert, die während des Deployments vom Server aufgerufen wird. Innerhalb dieser Methode wird ein GBeanInfo Objekt erzeugt, in dem sämtliche Attribut- und Methodenbeschreibungen enthalten sind.

```
/*
 * Copyright (c) 2005 Orientation in Objects GmbH
 * Weinheimer Str. 68, D - 68309 Mannheim, Germany
 * All rights reserved.
 */
package de.oio.geronimo;
import org.apache.geronimo.gbean.GBeanInfo;
import org.apache.geronimo.gbean.GBeanInfoFactory;
/**
 * @author kkoehler
 */
public class SimpleBean {
   public static final GBeanInfo GBEAN_INFO;
   static {
      GBeanInfoFactory infoFactory =
        new GBeanInfoFactory("SimpleGBean",
        SimpleBean.class);
      infoFactory.addAttribute("name", String.class, true);
      infoFactory.addOperation("printName", new Class[]{});
      GBEAN_INFO = infoFactory.getBeanInfo();
   }
   public static GBeanInfo getGBeanInfo() {
      return GBEAN_INFO;
   }
   private String name;
   /**
    * @return Returns the name.
    */
   public String getName() {
      return name;
   }
   /**
    * @param name The name to set.
    */
   public void setName(String name) {
      this.name = name;
   }
   public void printName() {
      System.out.println(name);
   }
}
```

Beispiel 5: Class Simple Bean

Zusätzlich zur imperativen Implementierung wird ein Deployment Descriptor benötigt, in dem das GBean dem Server bekanntgegeben wird. Dieser Deployment Descriptor (auch als plan bezeichnet) beschreibt die komplette Konfiguration, die später im Server zur Verfügung steht und gestartet werden kann. Anzugeben sind Bibliotheksabhängigkeiten und die eigentlichen GBeans. In unserem Fall befindet sich die Klasse SimpleBean in der Datei gbean-example.jar. Die Konfiguration "erbt" von der Konfiguration org/apache/geronimo/System und heißt de/oio/geronimo/Example. Mit diesem Namen kann später die Konfiguration gestartet werden.

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration
   xmlns="http://geronimo.apache.org/xml/ns/deployment"
   configId="de/oio/geronimo/Example"
   parentId="org/apache/geronimo/System">
 <dependency>
  <uri>oio/jars/gbean-example.jar</uri>
 </dependency>
   <gbean name="de.oio.geronimo:type=SimpleGBean"
      class="de.oio.geronimo.SimpleBean">
      <attribute name="name"
               type="java.lang.String">Kristian</attribute>
   </gbean>
</configuration>
```

Beispiel 6: gbean-example.xml

Über folgende Befehle kann die Konfiguration zur Laufzeit im ConfigurationStore installiert und im Server gestartet werden (die JAR Datei muss sich hierzu am angegebenen Ort im repository Verzeichnis des Servers befinden (oio/jars/gbean-example.jar)).

```
cd %GERONIMO_HOME% java -jar bin/deployer.jar
start PATH_TO_DD/gbean-example.xml
```

```
java -jar bin/deployer.jar distribute
PATH_TO_DD/gbean-example.xml
```

FAZIT

Die bisherige Entwicklung des Geronimo Servers läßt auf einen zukünftigen starken Konkurrenten der bestehenden Application Server schließen. Der neue IOC Kernel verspricht mit den eingesetzten generierten Proxy Klassen eine hohe Performance und bessere (da typsichere) Entwicklung. Diese Aussagen müssen sich natürlich erst noch in der Praxis bewähren, es handelt sich allerdings um einen sehr vielversprechenden Ansatz.

Mit der Konfigurationsverwaltung zielt Geronimo auf grössere sowie kleinere Installationen gleichermassen ab. Dieses Vorgehen vereinfacht die Reproduzierbarkeit einzelner Systeme was sich im produktiven Einsatz sicher als Vorteil erweisen wird.

Der aktuelle Stand der Entwicklung ist noch nicht für den vollständigen produktiven Einsatz bereit. Dies wird auch durch die aktuelle Versionsnummer (M3) deutlich. Allerdings sollte man Geronimo nicht aus den Augen verlieren, da einige neue, gute und interessante Ideen integriert sind.

REFERENZEN

- Geronimo, J2EE Server Projekt
  (http://geronimo.apache.org/)
- Jetty, Web Server & Servlet Container
  (http://jetty.mortbay.org/)
- Tomcat Servlet Container
  (http://jakarta.apache.org/tomcat/)
- HOWL - High-speed ObjectWeb Logger
  (http://howl.objectweb.org/)
- Open Source Messaging Fabric
  (http://activemq.apache.org/)
- TranQL, open source framework for building persistence engines
  (http://tranql.codehaus.org/)
- Spring, java/j2ee Application Framework
  (http://www.springframework.org/)
- OpenEJB, Open Source EJB Container System and EJB Server
  (http://www.openejb.org/)
- cglib, Code Generation Library
  (http://cglib.sourceforge.net/)