Administration und Management des Tomcat entzaubert

Tomcat-Zauber

Dynamisches Management des Tomcat 4.1 ist eine feine Sache. Wer möchte nicht ohne Restart des Servers seine Änderungen aktivieren können? Als Gehilfen für diese Art magischen Zaubers dienen die beiden Web-Anwendungen „Manager“ und „Administrator“, welche standardmäßig im Tomcat Server enthalten sind. Wir zeigen in dieser Ausgabe der Kolumne verschiedene Möglichkeiten, sowohl einzelne Komponenten als auch gesamte Web-Anwendungen zur Laufzeit zu modifizieren. Wie bei jedem Zauber gibt es auch hier eine Menge zu staunen und magische Tricks zu lernen.

Die Anforderungen an das dynamische Management innerhalb eines Servlet Containers sind weit gestreut. Generell lassen sie sich allerdings in zwei grobe Aufgabengebiete unterteilen: Einerseits möchte man die zentrale Tomcat-Konfiguration, die durch die Datei *conf/server.xml* entstanden ist, zur Laufzeit des Servers aktiv verändern; es handelt sich also um die Administration des Servers. Anderseits besteht das Bedürfnis, vollständige Web-Anwendung zu stoppen, zu verändern, erneut zu starten, zu löschen oder gar neu zum Server hinzuzufügen. In diesem Fall handelt es sich um das Management der Web-Anwendungen des Servers.

Zur Durchführung der ersten Aufgabe – Administration des Servers – nutzt man die „Administrator“-Anwendung. Diese administriert die MBeans des zentralen JMX-MBeanServers, der die gesamte Konfiguration des Servers zur Verfügung stellt. Mit Hilfe des Java Management eXtension Ansatzes können zur Laufzeit zum Beispiel Ressourcen, Connectoren, Session Manager, neue virtuelle Hosts oder Attribute eines Context (Konfiguration einer Web-Anwendung) verändert werden [1], [2], [3], [4].

Für die Durchführung der zweiten Aufgabe – Management von Web-Anwendungen – gilt es die sehr unterschiedlichen Anforderungen für eine Entwickler- bzw. Produktionskonfiguration zu berücksichtigen. Entwickler möchten am liebsten, dass jede Änderung an einer Datei, Konfiguration oder Klasse ihrer Web-Anwendung sofort im laufenden Server sichtbar werden. Die lästigen Wartezeiten eines Server Restart „Round a Trip“ sind heute – aufgrund der Serverkomplexität – ein unangenehmer Beigeschmack der Java Server-Entwicklung geworden, welchen Entwickler nur ungern in Kauf nehmen. In der Produktion ist aus Sicherheitsgründen dagegen eine eher statische Konfiguration erwünscht. Ein Hot Fix einer der vielen Anwendungen auf einem Server soll allerdings schon gezielt und schnell durchführbar sein. Und dies, ohne dass dabei alle anderen Anwendungen automatisch stillgelegt werden.

Thema des Monats
Deployment und Administration

Seit dem Release 4.1.x ist es mit der Web-Anwendung */admin* aus dem Distributionsverzeichnis *%CATALINA.HOME% server/webapps/admin* möglich, die einzelnen Komponenten der Konfiguration *server.xml* zu administrieren (Abb. 1). Die Administrator-Anwendung ist mit Struts 1.1 sehr modular realisiert. Zu ihrer Aktivierung bedarf es einiger kleiner Handgriffe, welche im folgenden Verlauf aufgezeigt werden.

Im Rahmen unseres Beispiels nutzen wir zur Konfiguration des Tomcat Servers ein eigenständiges *%CATALINA.BASE%*-Verzeichnis (Beispiel *webdev-server* auf der Heft-CD). Ein solches Verzeichnis enthält die Server-Konfiguration und verweist in den Startup- und Shutdown-Skripten auf eine Tomcat-Distribution. Im Unterverzeichnis *webapps/* befinden sich wie gewohnt die Anwendungen und im *conf/*-Verzeichnis die Konfigurationen (*server.xml*, *web.xml*, *jk2.properties* und *tomcat-user.xml*) (Abb. 2).

Normalerweise werden die Web-Anwendungen eines virtuellen Hosts einfach in das Verzeichnis *webapps/* entweder als Verzeichnis oder Archiv (*war*) kopiert. Alternativ kann durch eine entsprechende *<context>.xml*-Datei eine komplette Web-Anwendung mit ihren Tomcat-spezifischen Komponenten definiert werden (Listing 1). Damit muss die zentrale Server-Konfiguration *conf/server.xml* nicht für jede Anwendung angepasst werden. Die Administrator-Anwendung erhält mit dem Context-Attribute *priviledged=true* Zugang zu dem Classloader des Servers. Mit Hilfe dieser Einstellung kann die Anwendung auf das interne API des Tomcat zugreifen. Wie bereits weiter oben erwähnt, werden alle Komponenten des Tomcat in einem zentralen JMX-MBeanServer zur Verfügung ge-

Abb. 1: Die Administration-Anwendung des Tomcat 4.1.x

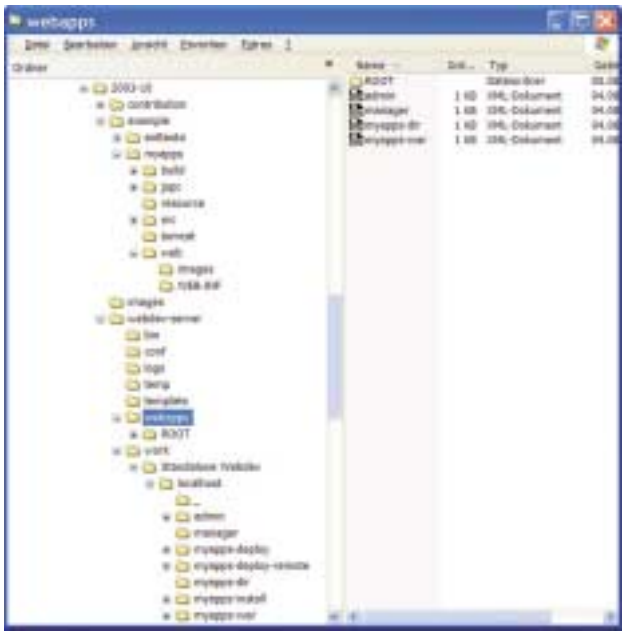

Abb. 2: Verzeichnisstruktur einer eigenständigen *%CATALINA.BASE%* Tomcat-Konfiguration

stellt. Als JMX-Implementierung kommen dabei MX4J und das Jakarta Commons Modeler Framework zum Einsatz [5], [6].

Im nächsten Schritt wird ein Valve definiert und festgelegt, dass nur Requests von IP Interface 127.0.0.1 zugelassen werden.

Der Zugriff auf die Administrator-Anwendung darf natürlich nur mit einer entsprechenden Autorisierung erfolgen. Deshalb muss für den zugehörigen Context ein Realm eingerichtet werden. In unserem Beispiel ist das *UserDatabaseRealm* ausgewählt worden, welches über JNDI die Daten für Rollen, Gruppen und Nutzer zur Verfügung stellt. Als Quelle der „Nutzerdatenbank" dient die Datei *conf/tomcat-users.xml*. In dieser müssen die Rollen „manager" und „admin" angelegt und mindestens einem Nutzer zugewiesen werden (Listing 2). Die Pflege dieses Realms kann innerhalb der Administrator-Anwendung erfolgen.

Listing 1: Definition einer Web-Anwendung durch eine Context-Descriptor *webapps/admin.xml*

```
<Context  path="/admin"
    docBase="c:\server\jakarta\jakarta-tomcat-4.1.27/
                                server/webapps/admin"
    debug="0"
    privileged="true">

  <Valve  className="org.apache.catalina.valves.
                                        RemoteAddrValve"
    allow="127.0.0.1"/>

  <Logger className="org.apache.catalina.logger.
                                        FileLogger"
    prefix="localhost_admin_log."
    suffix=".txt"
    timestamp="true"/>

  <Realm  className="org.apache.catalina.realm.
                                    UserDatabaseRealm"
    debug="0"
    resourceName="UserDatabase"
    validate="true"/>

</Context>
```

Listing 2: Rollen und Rechte für die Web-Anwendungen Administrator und Manager

```
<?xml version='1.0' encoding='UTF-8'?>
<tomcat-users>
 <role rolename="tomcat"/>
 <role rolename="manager"/>
 <role rolename="admin"/>
 <user username="tomcat" password="tomcat"
roles="tomcat"/>
 <user username="manager" password="tomcat"
roles="manager,admin"/>
</tomcat-users>
```

Listing 3: Zugang zum Tomcat MBeanServer.

```
MBeanServer server = (MBeanServer)
  getServletContext().getAttribute
                    ("org.apache.catalina.MBeanServer");
Registry registry = (Registry)
  getServletContext().getAttribute
                    ("org.apache.catalina.Registry");
```

Der Zugriff auf den MBeanServer und die zentrale MBean-Registratur wird der Anwendung durch Context-Attribute ermöglicht (Listing 3). Mit diesen Service Instanzen ist die Administrator-Anwendung in der Lage, sowohl neue MBeans zu erzeugen und bestehende im MBeanServer zu ändern als auch JMX-Operation auf diesen auszuführen. Der MBeanServer wird durch den *ServerLifecycleListener* der Tomcat Service-Komponente gestartet [3], [7].

Wem der Luxus dieser Administrations-Console zu viel erscheint oder wer das Deployment einer eigenen Web-Anwendung zur Administration im Produktionsserver nicht für die richtige Wahl hält, dem bieten sich Alternativen: Der Coyote JK2 Connector stellt zum Beispiel einen JMX Adpator (mx) zur Verfügung. Dieser startet wahlweise die MX4J oder Sun JMX-Adaptoren, die via HTTP oder RMI den MBeanServer verfügbar machen (siehe Kasten „Zauber: Administration mit alternativen JMX Clients"). Ein alternativer Client ist MC4J, der verschiedene Java Server innerhalb einer JMX-Console zusammenführt (Abb. 3).

Nachdem wir im bisherigen Verlauf einen Blick auf die Administrator-Anwendung geworfen haben, wollen wir uns nun den Manager einmal genauer anschauen.

Der Tomcat-Manager dient dazu, Web-Anwendungen zu starten, zu stoppen, hinzuzufügen und zu entfernen. Die oben ge-

Abb 3: Die JMX-Console MC4J

nannten Aktionen des Managers erfordern keinen Neustart des Tomcat. Durch diese Eigenschaft wird der Manager sowohl in der Entwicklungsphase als auch in der Produktion zu einem interessanten Instrument. Um die beschriebene Funktionalität zu nutzen, gibt es verschiedene Möglichkeiten:

- Die Manager-Anwendung direkt und schnell mit einer textuellen Ausgabe ansprechen,
- das HTML-Frontend des Manager-Anwendung oder
- Zugang der Aktionen durch spezifische Ant-Tasks.

Um den Manager direkt und schnell zu nutzen, werden die Aktion und die für sie benötigten Argumente als URL-Parameter übergeben (*http://host:port/manager/[aktion]?[parameter]*). Die einfachste Aktion des Managers ist das Auflisten der installierten Web-Anwendungen über die Aktion *list*. Da diese Aktion keine Parameter benötigt, erfolgt der Aufruf über die URL *http://localhost:7380/manager/list*. Eine Übersicht über alle Aktionen findet sich in Tabelle 1. In der HTML-Anwendung des Managers stehen diese Befehle ebenfalls zur Verfügung, grafisch aufbereitet (*http://localhost:7380/manager/html*), wie man es von einem Frontend erwartet (Abb. 4).

Für den Zugriff über Ant sind dieselben Aktionen als Task verfügbar, wobei hier zu beachten gilt, dass das Archiv *catalina-ant.jar* für den Taskdef benötigt wird. Die einfachste Möglichkeit, dies in einem Ant-Skript zu ermöglichen, ist einen Pfad für dieses JAR zu definieren und in der Task-Definition zu referenzieren. Je nach Befehl können dann verschiedene Attribute in dem entsprechenden Ant-Task gesetzt werden. Dies hat den unbeschreiblichen Vorteil, dass man im Grunde ein allgemeines Ant-Skript nutzen kann, und die Parameter nur noch über Ant setzt (Listing 4 und in dem Beispiel auf der Heft-CD). Damit nicht für jeden Task eine eigene Taskdef-Definition angeben werden muss, haben wir das Mapping der Namen der Task und der Implementierungsklassen in der *catalina-ant.properties*-Datei hinterlegt. Die Parameter der Targets sind in der *build.properties* hinterlegt und nur die spezifischen Argumente z.B. *${app.path}* werden dann im aufrufenden Ant-Skript explizit genutzt.

Gerade die Aktion *deploy* ist hochinteressant. Im Gegensatz zur Aktion *install*, die ein auf der Maschine vorhanden Verzeichnis oder WAR-Archiv im Server installiert, wird mit der Aktion *deploy* ein

Zauber: Administration mit alternativen JMX Clients

Der JK2 Connector – zur Anbindung des Tomcat an einem Web Server – kann verschiedene Handler zur Bearbeitung von Anfragen starten. Nicht nur, dass es die bekannten Channels für Remote Socket (*channelSocket*), JNI (*channelJNI*) und Shared Memory (*shm*) gibt, auch ein JMX Adaptor (*mx*) kann erzeugt werden. Auch hier kommt, wie schon bei dem Tomcat JMX-MBeanServer, die erprobte Realisierung des MX4J Projekts oder, falls installiert, die Sun Referenz-Implementierung zum Einsatz [1], [5]. Als weiteren Client für verschiedene JMX Server lässt sich auch das MC4J Projekt nutzen [10].

Hier die „Step by Step" Anleitung zur Nutzung eines alternativen JMX Client:

- Anlage der Datei *%CATALINA.BASE%\conf\jk2.properties* mit dem Inhalt „mx.port=9000"
- Anlage eines Coyote JK2 Connector in der *%CATALINA.BASE%\conf\server.xml*:

```
<Connector className="org.apache.coyote.tomcat4.CoyoteConnector"
  port="8009" minProcessors="0" maxProcessors="5"
  enableLookups="false"
  acceptCount="10" debug="0" connectionTimeout="0"
  useURIValidationHack="false"
  protocolHandlerClassName="org.apache.jk.server.JkCoyoteHandler"/>
```

- Kopieren der *mx4j-tools.jar* der MX4J Distribution ins Verzeichnis *%CATALINA_HOME%\server\lib*. Alternativ kann die Sun JMX-Referenzimplementierung mit den beiden Archiven *jmxri.jar* und *jmxtools.jar* genutzt werden. **Achtung**: Ein gleichzeitiger Einsatz beider Adaptoren ist nicht vorgesehen.
- Nun kann mit unserem *%CATALINA.BASE%\bin\startup* Skript der Tomcat erneut gestartet werden.
- Mit einem beliebigen Browser kann via *http://<tomcathost>:9000/* die MX4J HTML JMX Console erreicht werden.
- Alternativ kann der JMX Client MC4J genutzt werden:
- Nach dem Download und der Installation von MC4J kann der Client mit dem Programm *bin\runide.exe –jdkhome <JAVA_HOME>* gestartet werden.
- Mit dem Tab MC4J FOR MX4J wird zu der Konfiguration eines MX4J Server verzweigt.
- Mit dem Rechts-Klick und der Wahl von CONNECT können die Verbindungsparameter angelegt werden.
- Eingabe eines Namens und Bestätigung der Parameter (*localhost:1099*).
- Mit einem Doppelklick auf den erstellten Eintrag wird die Verbindung zum JMX Server Tomcat hergestellt.
- Lassen sie sich von den alternativen Wegen der Remote Administration des Tomcat bezaubern.

Einziger Wermutstropfen dieser Methode ist zur Zeit, dass ein normaler Shutdown des Tomcat mit gestarteten JMX-Adaptoren nicht mehr den JVM-Prozess korrekt beendet. Der Tomcat scheint den RMI-Adpatoren nicht von dem Shutdown zu benachrichtigen oder im MX4J-Adaptor ist ein aktiver Thread gestartet.

WAR-Archiv auf den Server hochgeladen (upload). Dies kann sogar indirekt via URL auf einen weiteren Server geschehen. So kann man zum Beispiel für einen Cluster leicht ein zentrales Anwendungsrepository mit verschiedenen Versionen schaffen und durch ein Ant-Skript auf allen Maschinen nahezu gleichzeitig freigeben.

Eine sehr elegante Ergänzung hierzu ist: Man kann die Datei *context.xml* im *META-INF/*-Verzeichnis des WAR-Archivs unterbringen. Damit können die notwendigen Services der Anwendung immer aktuell gehalten werden. Ein solches Deployment bewirkt als Seiteneffekt, dass die zentrale Server-Konfiguration in der Datei *conf/server.xml* angelegt wird. Es wird der gesamte aktuelle Serverzustand gespeichert, also auch die Einstellungen, die evtl. nur temporär durch den Administrator geändert wurden. Natürlich wird die aktuell gültige Version der Datei in ein Backup überführt. Ein derartiges Deployment überlebt einen Server-Restart somit problemlos.

In der Entwicklung hat sich herausgestellt, dass das oben beschriebene Verhalten nicht immer gewünscht ist. Manche temporären Anwendungen bzw. alle Anwendung im Verzeichnis *webapps/* befinden sich bei diesem Verfahren unerwünschter Weise in der *server.xml*. Außerdem werden alle Elemente in der *server.xml* mit allen Attributen vollständig abgebildet. Eine Änderung in einer *webapps/<context>.xml* wird beim nächsten Restart somit nicht mehr berücksichtigt. Dies kann zu einiger Verwunderung während der Ent-

serverinfo	http://localhost:7380/manager/serverinfo
	Zeige die aktuelle Server Information.
sessions	http://localhost:7380/manager/session?path/myapps-xxx
	Zeige für eine Web-Anwendung einige Sessioninformationen.
roles	http://localhost:7380/manager/roles
	Zeige alle Rollen der globalen JNDI UserDatabase.
resources	http://localhost:7380/manager/resources?type=javax.sql.DataSource
	Zeige alle globalen JNDI Ressourcen des gewählten Typs.
List	http://localhost:7380/manager/list
	Liste der aktuell verfügbaren Web-Anwendungen mit Aktivitätszustand und Anzahl der geöffneten Sessions.
start	http://localhost:7380/manager/start?path=/myapps-xxx
	Start der Web-Anwendung */myapps-xxx*.
Stop	http://localhost:7380/manager/stop?path=/myapps-xxx
	Stopp der Web-Anwendung */myapps-xxx*.
install	http://localhost:7380/manager/install?path=/myapps-*install&* config=file:/C:/ tomcatkolumne/03/10/webapps/myapps-install.xml& war=file:/ C:/tomcatkolumne/ 03/10/webapps/myapps-war.war
	Installation einer neuen Web-Anwendung */myapps-install* aus einem Verzeichnis auf dem Server. Der Context wird mit Hilfe der Datei *myapps-install.xml* konfiguriert.
remove	http://localhost:7380/manager/remove?path=/myapps-install
	Entferne Web-Anwendung aus dem Server. Das Verzeichnis bleibt erhalten.
deploy	http://localhost:7380/manager/deploy?path=/myapps-deploy*&war=file:/C:/ tomcatkolumne/03/10/webapps/myapps-war.war* http://localhost:7380/manager/deploy?path=/myapps-deploy-remote&war= http://localhost:7380/myapps-war-remote.war
	Eine neue Web-Anwendung wird auf den Server geladen und ausgepackt im Verzeichnis *webapps/* gespeichert. Es ist auch möglich, das Archiv von einem entfernten HTTP-Server zu laden.
undeploy	http://localhost:7380/manager/undeploy?path=/myapps-deploy
	Die Web-Anwendung */myapps-deploy* wird vom Server gelöscht, d.h., die Anwendung wird im Verzeichnis *%WORK%* des Hosts vollständig gelöscht.

Tab. 1: Tomcat Manager Aktionen

wicklung führen: „Alles kopiert und der Server akzeptiert es einfach nicht mehr." Daher unser Tipp: Wenn ein Server mit der *deploy*-Aktion gemanaged wird, dann besser vollständig! Eine Alternative, die sich in der Praxis bewährt hat, ist das zusätzliche Hinterlegen einer manuell gepflegten Datei namens *conf/default-server.xml*, die bei Bedarf einen „kontrollierten" Zustand des Servers widerspiegelt.

Einige Adminstratoren sind nach dem gerade Gelesenen sicher ein wenig unruhig geworden oder schlagen gar die Hände über dem Kopf zusammen. Zu Recht, denn die genannten Anwendungen können etliche Sicherheitsbedenken auslösen. Remote Management und Administration sollte daher nur von internen Servern erlaubt werden. Es ist auf jeden Fall ratsam und wird von uns dringend empfohlen für die Manager- und Administrator-Anwendung ein eigenes Access Log zu schreiben und aktiv zu kontrollieren.

Wie so vieles im Tomcat sind natürlich auch die beiden Anwendungen Administrator und Manager nicht ganz frei von Fehlern. Im Tomcat 4.1.27 befindet sich ein behobener Fehler für das aktive Reloading von geänderten Web-Anwendungsklassen und Archiven. Ein Hot Fix steht bereit und eine weitere Tomcat 4.1.x-Version wird Anfang September verfügbar sein.

Listing 4: Der Start- und der Deploy-Task des Tomcat

```
<project name="Tomcat Admin" default="usage">
 <property file="build.properties"/>
 ...
 <taskdef resource="catalina-ant.properties">
   <classpath>
     <pathelement location="${catalina.home}/server/
                                    lib/catalina-ant.jar"/>
     <pathelement location="resource"/>
   </classpath>
 </taskdef>
 ...
 <target name="start"
   description="start application on servlet container">
   <start url="${manager.url}"
     username="${manager.username}"
     password="${manager.password}"
     path="${app.path}"/>
 </target>
 <target name="deploy"
   description="Deploy application on servlet container">

   <deploy url="${manager.url}"
     username="${manager.username}"
     password="${manager.password}"
     war="file://${app.war}"
     path="${app.path}"/>

 </target>
 <target name="remote-deploy"
   description="Deploy remote application on
                                    servlet container">

   <deploy url="${manager.url}"
     username="${manager.username}"
     password="${manager.password}"
     war="${remote.war}"
     path="${app.path}"/>

 </target>
 ...
</project>
```

Gerüchteküche
Tomcat

Auch in dieser Ausgabe wollen wir wieder einmal das eine oder andere Gerücht rund um das Thema Tomcat in den Raum werfen.

Im aktuellen Release (zur Zeit des Schreibens 4.1.27) hat sich wieder einmal einiges getan. Insbesondere der doch nicht ganz so leicht zu handhabende JK2-Adaptor ist um verschiedene Bugs bereinigt worden. So sind zum Beispiel die Schwierigkeiten, die es bei der Anbindung des Apache Webservers via mod JK2 geben konnte, beseitigt. Darüber hinaus wurde die Möglichkeit geschaffen, im Rahmen einer JK2-Verbindung auch mit SSL-Zertifikaten zu arbeiten. Auch im Bereich Client Certification Handling hat sich einiges getan, wodurch die Anzahl der Fehlermeldungen – Stichwort: Device DEBUG – deutlich reduziert wird.

Tomcat 5 kommt indes immer näher. Während in der letzten Ausgabe des *Java Magazins* das Release-Datum noch relativ wage mit Ende 3. Quartal/Anfang 4. Quartal betitelt wurde, heißt es inzwischen, dass mit einem Release Mitte Oktober gerechnet werden kann. Dieses Datum passt auch in den Zeitrahmen, wenn man davon ausgeht, dass der offizielle Start der Beta Phase auf Mitte August festgesetzt wird.

Zu guter Letzt soll natürlich nicht verschwiegen werden, dass für Ende September ein weiteres Tomcat 4.1.x Release geplant ist. Endlich wieder einmal eine Möglichkeit, auf Bug Suche zu gehen.

Und beim nächsten Mal...

In der nächsten Ausgabe der Tom C@-Kolumne werden wir uns eingehend mit der Thematik der Realms befassen. Ein kleiner Eindruck zu deren Anwendung und Nutzen konnte bereits in der aktuellen Kolumne durch die Verwendung des *UserDatabaseRealm* gewonnen werden.

Abb. 4: Die Manager Console des Tomcat 4.1

Neben der generellen Idee, welche sich hinter den Realms verbirgt, werden wir die wichtigsten Realms mit ihren Anwendungsparametern vorstellen und durch ein praktisches Beispiel verdeutlichen. Natürlich wird es wie gewohnt auch den einen oder anderen Trick zu sehen geben.

Wir freuen uns schon auf Kommentare und Anregungen – besuchen sie also unsere Tom C@-Site [7] und das Tom C@-Forum [8] oder die Tomcat Mailing-Listen [9].

Links & Literatur

[1] java.sun.com/products/jmx

[2] jakarta.apache.org/tomcat/

[3] Peter Roßbach, Lars Röwekamp & Michael Kloss: „Kater à la carte", Java Magazin 9.2003

[4] Peter Roßbach (Hrsg.); Andreas Holibek, Thomas Pöschmann, Lars Röwekamp, Peter Tabatt: „Tomcat 4X: Die neue Architektur und moderne Konzepte für Webanwendungen im Detail", Software und Support Verlag, Frankfurt 2002

[5] mx4j.sourceforge.net/

[6] jakarta.apache.org/commons/modeler/

[7] tomcat.objektpark.org/

[8] www.javamagazin.de/tomcat/

[9] www.mail-archive.com/tomcat-user@jakarta.apache.org/
www.mail-archive.com/tomcat-dev@jakarta.apache.org/

[10] mc4j.sourceforge.net/