Grundlagen eines Tomcat Cluster

von Peter Roßbach

Magische Momente

Die Verfügbarkeit einer Webanwendung zu garantieren ist eine Herausforderung. Schon kleine Änderungen oder eine starke Nutzungszunahme der Anwendung forcieren Ausfälle. Gerade für den Betreiber einer erfolgreichen internationalen Website ist es dann schon problematisch, ein geeignetes Zeitfenster für geplante Release-Wechsel zu finden. Weiterhin kämpfen Administratoren oftmals erfolglos gegen Lastspitzen und Fehler, die den gesamten Server regelmäßig zum Absturz bringen. In dieser Kolumne wollen wir deshalb den technischen Grundlagen eines Tomcat 5 Cluster auf den Zahn fühlen.

Erfolg ist doch etwas Erstrebenswertes, oder nicht? Wenn Ihnen dieses frohe Ereignis aber unvorbereitet auf Ihrer Website passiert, wird das schnell zu einer echten Belastung aller Beteiligten. Weitere Hardware ist sicherlich schnell besorgt, aber wie wird die Last effektiv verteilt und welche Zustandsdaten einer Webanwendung müssen wirklich gesichert werden? Haben die Entwickler die Anwendung überhaupt skalierbar und verteilbar entworfen? Ein zustandsloser Web Service skaliert auch mit einem einfachen Lastverteiler, aber die meisten Webanwendungen sind zustandsbehaftet. Formular- oder Sachbearbeitersysteme, die mit Java-Webtechnologie entwickelt wurden, sind eben mittlerweile üblich. Um für solche Systeme durch Verteilung ein besseres Antwortverhalten und die Minimierung der Ausfallwahrscheinlichkeit zu erzielen, muss man sich den folgenden Fragen stellen:

- Welche Antwortzeiten sind zumutbar und überhaupt erreichbar?
- Welche Daten müssen bei einem Ausfall unbedingt sofort wieder vorhanden sein?
- Wie transparent darf ein Ausfall für die Anwender gestaltet werden?
- Ist eine Neuanmeldung vertretbar?
- Der Anwender bemerkt nichts und arbeitet eventuell etwas langsamer weiter.
- Wie viele Anwender sind von einem Ausfall betroffen?
- Warum kommt es und wie häufig zu einem Ausfall eines oder mehrerer Systeme?
- Sind der Web-Container oder ein Hintergrundsystem (Datenbank, fremder Dienst …) überlastet?
- Gibt es viele Fehler in der Anwendung oder ist eine hohe Änderungshäufigkeit zu erwarten?
- Verfügen Sie für Ihr Anwendungsszenario über eine in Qualität und Quantität angemessene Hardware?
- Wie groß sind die Lasten von Anfrage oder Ausführung auf welche Teile der Anwendung?

Für die Beantwortung dieser Fragen sind Zeit und Ehrlichkeit erforderlich. Ohne ein paar kleine Experimente, um die Machbarkeit zu prüfen, wird dies nur schwer gelingen. Der Tomcat 5.5 liefert zur Ausfallsicherung schon eine ansehnliche Lösung mit seinem Cluster-Modul. Für die Lastverteilung stellt das Tomcat-Projekt das Modul mod_jk zur Verfügung. Es wird dazu in den Webserver (Apache oder IIS) eingebunden, der dann als LoadBalancer für mehrere Tomcats dient (Abb. 1). Ein Beispiel zum Aufsetzen eines Tomcat Cluster befindet sich auf der Heft-CD.

Lastverteilung mit dem neuen mod_jk 1.2.7

Die Verteilung der Last auf mehrere Server kann im einfachsten Fall durch die Vergabe mehrerer IP-Adressen im DNS-Service erfolgen. Bei jeder Namensauflösung des Rechners wird dann jedem Client eine Liste von IP-Adressen geliefert. Meist rotiert der DNS-Server diese Liste pro Anfrage, da die meisten Clients nur die erste IP-Adresse einer DNS-Antwort verwenden. Wenn Ihre Anwendung allerdings serverseitige Zustände halten soll, kommen Sie um die Konfiguration eines eigenen Lastverteilers nicht mehr herum. Das mod_jk-

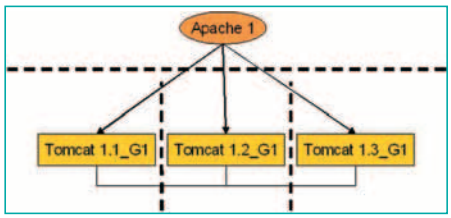

Abb. 1: Apache-mod_jk-Anbindung an mehrere Tomcat-Knoten

Modul bietet gerade diese Funktionalität. Eine Anfrage über HTTP kann an einen oder mehrere Tomcat-Knoten weitergeleitet werden. Wenn der Tomcat eine neue Session generiert, wird diese in der HTTP-Antwort durch einen Session Cookie oder per URL Rewriting markiert [1]. Diese Information ermöglicht dem LoadBalancer im mod_jk die Zuordnung weiterer Anfragen zu dem Server, der den Zustand des entsprechenden Clients hält. Weiterhin lenkt der LoadBalancer die Anfrage um, sollte der entsprechende Server nicht mehr erreichbar sein. Zwischen Webserver und Tomcat werden die Daten über das Apache Java Protocol (AJP) ausgetauscht [2], indem dazu eine stehende TCP/IP-Verbindung erzeugt wird. Damit entfällt für jede weitere Anfrage der teure Aufbau einer TCP/IP-Verbindung. Außerdem wird die schon vom Webserver vorverarbeitete HTTP-Anfrage in eine kompaktere binäre Form überführt und erst dann weitergesendet. Eigene Messungen zeigen, dass AJP eine Steigerung von 30 bis 50 Prozent gegenüber dem normalen HTTP Forwarding herkömmlicher Proxies erbringt.

Nun ist es offiziell: Die Entwicklung des mod_jk2 Moduls ist gestoppt worden und die weitere Entwicklung wird ausschließlich am altgedienten mod_jk-Modul weiter betrieben [2]. Die wesentlichen Gründe liegen in der sehr schleppenden Marktakzeptanz des mod_jk2 und der zu hoch gesteckten Realisierungsziele, die erhebliche Wartungsprobleme nach sich gezogen haben. Sehr erfreulich ist allerdings, dass einige der Errungenschaften des mod_jk2 nun in das mod_jk ihren Einzug finden werden. Ein erster Realisierungsschritt stellt die neue mod_jk-Version 1.2.7 mit folgenden Änderungen dar:

- neuer verbesserter Lastverteilungsalgorithmus mit Unterstützung für Cluster Domains
- direkte Definition der Worker (Tomcat-Knoten) in der Apache-Konfigurationsdatei *httpd.conf*
- verbesserte Protokollausgaben zur leichteren Fehleranalyse
- URL-Angaben mit Mustern in den Befehlen JkMount und JkUnMount
- Überarbeitung der gesamten Dokumentation
- Refactoring und Redesign des gesamten Moduls zur Verbesserung der Abarbeitungsgeschwindigkeit

Der neue Lastverteilungsalgorithmus verteilt die Last weiterhin nach einem gewich-

Listing 1

Lastverteilungsdefinition in der Apache-Datei *httpd.conf* mit mod_jk 1.2.7

```
<IfModule !mod_jk.c>
 LoadModule jk_module "modules/mod_jk.dll"
</IfModule>

JkWorkerProperty worker.list=balancer

JkWorkerProperty worker.node1.port=9012
JkWorkerProperty worker.node1.host=127.0.0.1
JkWorkerProperty worker.node1.type=ajp13
JkWorkerProperty worker.node1.cachesize=10
JkWorkerProperty worker.node1.recycle_timeout=300
JkWorkerProperty.worker.node1.lb_factor=1

JkWorkerProperty worker.node2.port=9022
JkWorkerProperty worker.node2.host=127.0.0.1
JkWorkerProperty worker.node2.type=ajp13
JkWorkerProperty worker.node2.cachesize=10
JkWorkerProperty worker.node2.recycle_timeout=300
JkWorkerProperty.worker.node2.lb_factor=1

JkWorkerProperty worker.node3.port=9032
JkWorkerProperty worker.node3.host=127.0.0.1
JkWorkerProperty worker.node3.type=ajp13
JkWorkerProperty worker.node3.cachesize=10
JkWorkerProperty worker.node3.recycle_timeout=300
JkWorkerProperty.worker.node3.lb_factor=1

JkWorkerProperty worker.balancer.balanced_workers=
                                    node1,node2,node3
JkWorkerProperty worker.balancer.type=lb

JkLogFile "logs/mod_jk.log"

JkLogLevel debug

JkMount /ClusterTest balancer
JkMount /ClusterTest/* balancer
```

Listing 2

Minimal-Cluster-Definition des Tomcat 5.5.5

```
<Server port="9005" shutdown="SHUTDOWN" >
 <Service name="Catalina">
  <Connector port="9013" protocol="AJP/1.3"/>
  <Engine name="Catalina" defaultHost="localhost"
                                      jvmRoute="node1">
  <Host name="localhost"
   appBase="webapps"
   unpackWARs="false">
   <Cluster
    className="org.apache.catalina.cluster.tcp.
      SimpleTcpCluster"managerClassName="org.apache.
                catalina.cluster.session.DeltaManager"
    expireSessionsOnShutdown="false"
    notifyListenersOnReplication="true"
    useDirtyFlag="true">

    <Membership
     className="org.apache.catalina.cluster.mcast.
                                          McastService"
     mcastAddr="228.0.0.4"
     mcastBindAddress="127.0.0.1"
     mcastPort="45564"
     mcastFrequency="500"
     mcastDropTime="3000"/>

    <Receiver
     className="org.apache.catalina.cluster.tcp.
                                      ReplicationListener"
     tcpListenAddress="auto"
     tcpListenPort="9015"
     tcpSelectorTimeout="100"
     tcpThreadCount="6"/>

    <Sender
     className="org.apache.catalina.cluster.tcp.
                                      ReplicationTransmitter"
     replicationMode="pooled"
     ackTimeout="15000"/>

    <Valve
     className="org.apache.catalina.cluster.tcp.
                                      ReplicationValve"
     filter=".*\.gif;.*\.js;.*\.css;.*\.png;.*\.jpeg;
      .*\.jpg;.*\.htm;.*\.html;.*\.txt;"/>

    <Deployer
     className="org.apache.catalina.cluster.deploy.
                                      FarmWarDeployer"
     tempDir="${catalina.base}/cluster/temp"
     deployDir="${catalina.base}/webapps"
     watchDir="${catalina.base}/cluster/watch"
     watchEnabled="true"/>

    </Cluster>
   </Host>
  </Engine>
 </Service>
</Server>
```

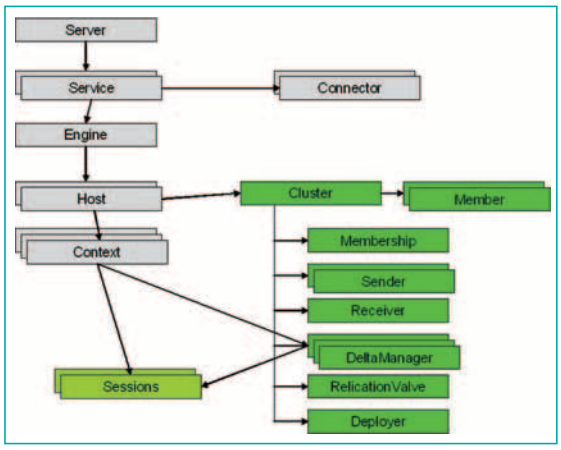

Abb. 2: Übersicht der Cluster-Infrastruktur

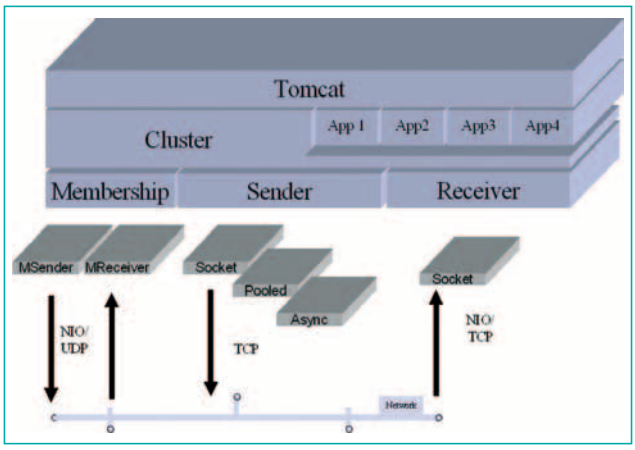

Abb. 3: Cluster-Infrastruktur im Detail

teten Verfahren und berücksichtigt dabei endlich mehrere ausgezeichnete Tomcat-Knoten (Local Worker) und die Zugehörigkeit zu einer Cluster Domain. Endlich lassen sich auch die Anzahl der Verbindungsversuche und das Prüfintervall für ausgefallene Knoten konfigurieren. Sogar den aktiven Abbau ungenutzter Verbindungen zum Tomcat kann man nun einstellen (Listing 1, Abb. 1). Im Beispiel sind drei Tomcat-Knoten auf einem Rechner definiert, die alle dieselbe Belastung zugeteilt bekommen sollen. Die Verteilung der Last wird nur auf Basis der ersten Anfrage eines Clients entschieden. Die Definition eines Verbindungs-Cache (cachesize > 0) ist natürlich nur für Multithread-Server sinnvoll. Wer wirklich den gesamten Datenverkehr und detaillierte Informationen zur Arbeitsweise des mod_jk sehen möchte, kann dies mit der *JkLogLevel*-Einstellung *trace* erreichen. Allerdings sollte man sich dann auf die Ausgabe wirklich vieler Informationen einstellen.

Die Definition eines Clusters

Der Tomcat 5 besitzt eine einfache Cluster-Lösung [3]. Das Clustering bezieht sich in diesem Fall auf die Zustandsreplikation auf alle Knoten im selben Cluster (Listing 2). Die Verteilung der Sessions erfolgt für jede Webanwendung, die als „distributable" in der Datei *WEB-INF/web.xml* markiert ist. Für solche verteilbaren Anwendungen müssen alle Objekte, die mit einer Session assoziiert werden, die Schnittstelle *java.io.Serializable* implementieren. Diese Anforderung hört sich einfach an, ist aber oft für eigene ältere Klassen, die von Drittherstellern- oder Open-Source-Bibliotheken stammen, nicht immer erfüllbar. Die bekannte „grüne Wiese" ist auch im Java-Umfeld nicht mehr immer umsetzbar, trotzdem ist für eine Zustandsreplikation die Speicherbarkeit eines Objekts Voraussetzung für die Cluster-Fähigkeit.

Die Verteilung erfolgt auf der Basis von Sessionänderungen, die in einer Anfrage auftreten. Nur die Objekte, die in der Session tatsächlich geändert wurden, werden auf den anderen Cluster-Knoten repliziert. Eine Veränderung an der Session wird ausschließlich durch die Nutzung der Methoden *setAttribute()* und *removeAttribute()* festgestellt. Eine direkte Manipulation des Objekts kann der Cluster *DeltaManager* nicht feststellen und führt auch nicht zur Replikation der Zustandsdaten. Wenn ein neuer oder ausgefallener Knoten gestartet wird, fordert jede Anwendung einzeln alle Sessions des gesamten Clusters von einem anderen Knoten an. Die Zugehörigkeit (Membership) zu einem Cluster wird durch regelmäßiges Senden eines Membership Ping beantragt. Als Information sendet jeder Knoten seine Verbindungsdaten, auf dem er Cluster-Nachrichten empfangen kann (Abb. 4). Der Empfänger eines solchen Ping schafft beim Empfang der ersten Membership-Nachricht die entsprechende Infrastruktur zur Replikation. Bei längerem Ausbleiben der Bestätigung der Zugehörigkeit wird der Sender für diesen Knoten dann wieder entfernt. Für die Netzwerkinfrastruktur der Zugehörigkeit und des Cluster-Nachrichtenempfangs wird auf das Java 1.4 NIO API vertraut (Abb. 2). Der Cluster ist als offener Kommunikationsbus realisiert, das heißt, es können auch eigene Nachrichten gesendet und empfangen werden. Für das Senden der Nachrichten gibt es drei verschiedene Sender: Zwei Sender senden synchron nach der Bearbeitung der HTTP-Antwort ihre Session-Nachrich-

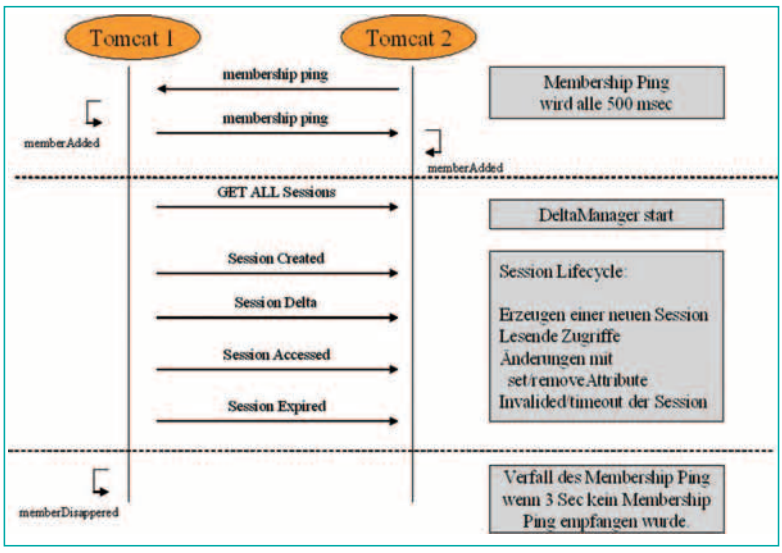

Abb. 4 Cluster-Nachrichten im Tomcat 5

ten. Der Synchronous-Sender sendet seine Nachrichten dabei über genau einen Socket pro Cluster-Mitglied. Der Pooled-Sender kann hingegen bis zu 25 parallele Sockets bedienen. Beide lassen sich die erfolgreiche Verarbeitung bestätigen und senden ihre Nachricht im Fehlerfall erneut. Der Asynchronous-Sender sammelt die Nachrichten in einer Queue pro Cluster-Mitglied und sendet die Nachrichten in einem eigenständigen Hindergrund-Thread. Mit diesem Replikationsmodus werden die eigentlichen Connector Threads des Tomcat schneller für die weitere Anfragebearbeitung frei. Das Risiko, dass bei einem Ausfall eines Primärknotens die Backup-Knoten allerdings nicht ganz aktuelle Daten haben, steigt natürlich, wenn die Queues gut gefüllt sind.

Bei einer Replikation werden nicht nur einfach die Zustandsdaten in die Backups kopiert, sondern die mögliche Signalisierung von Session-Events des Servlet API 2.4 wird auf allen Backups vollständig nachgezogen. Auf allen Knoten werden also alle in der *WEB-INF/web.xml*-Datei registrierten Listener mit den Schnittstellen *HttpSessionActivationListener*, *HttpSessionAttributeListener* oder *HttpSessionListener* aufgerufen [4] [1]. In der Klasse *SimpleTcpCluster* kann man mit *notifyListenerOnReplication=false* die Signalisierung für die Änderungen von Attributen auf den Backups verhindern.

Die Zugehörigkeit zum Cluster wird über Multicast-UDP/IP-Pakete verbreitet. Alle Knoten im Cluster müssen dafür auf derselben MultiCast-Adresse (228.0.0.4) und demselben Port (45564) konfiguriert werden. Der *ReplicationListener* arbeitet mit nicht blockierenden TCP/IP-Sockets und einer durch den Wert des Attributs *tcpThreadCount* einstellbaren Anzahl paralleler Verarbeiter-Threads. Für jeden Knoten im Netz gilt die Faustregel, dass ein Receiver Worker Thread pro Knoten im Cluster bereitgestellt werden sollte. Damit wird sichergestellt, dass die Backup-Knoten bei einer Lastspitze anderen Knoten nur moderat mit der Datenreplikation belastet werden. Das *ReplicationValve* im Tomcat erkennt pro Anfrage, ob eine Zustandsänderung stattgefunden hat, und veranlasst den jeweiligen *DeltaManager* zum Senden der Replikationsnachrichten der Sessions (Abb. 4). Für statische Ressourcen wie Bilder oder Texte kann die Prüfung mit dem Attribut *filter* unterdrückt werden.

Der Cluster kann auch dazu genutzt werden, das Deployment von Anwendungen clusterweit zu veranlassen. Der *FarmWarDeployer* überwacht das im Attribut *watchDir* angegebene Verzeichnis auf Veränderungen. Beim Start des Deployer-Knotens (*watchEnabled=true*) wird einmalig jede Anwendung an alle Knoten versandt. Damit alle Anwendungen auf allen Knoten auf demselben Stand sind, wird der Deployer-Knoten immer zuletzt gestartet. Die Anwendungen müssen als WAR-Archiv vorliegen. Das Entfernen der Anwendung erreicht man durch das Löschen des Archivs aus dem *watchDir*-Verzeichnis. Jede Aktualisierung zieht ein Redeployment der Anwendung auf allen Knoten des Clusters nach sich. Der einzige Nachteil des *FarmWarDeployer* ist, dass ein später zugeschalteter Cluster-Knoten keine Aktualisierung der Anwendung mitbekommt.

Fazit

Der vorgestellte Tomcat 5-Cluster ist eine schlanke Lösung, um die Ausfallsicherheit von Webanwendungen zu erhöhen. Die *InMemory*-Replikation auf *alle* Knoten im Cluster ist ein stark begrenzender Faktor hinsichtlich der Anzahl der möglichen Knoten. Je nach verfügbarem Speicher und Speicherbedarf der Sessions sind zwischen drei und sechs Knoten in einem Cluster betreibbar. Mit der neuen mod_jk-Version 1.2.7 kann man Gruppen von Cluster-Knoten (Workern) zu Domains zusammenfassen, die vorrangig bei einem Ausfall als Backup herangezogen werden sollen. Mit diesem Konzept von Rainer Jung lassen sich mehrere Cluster-Partitionen schaffen und die Skalierung auf wesentlich mehr Server verteilen. In der nächsten Ausgabe der Kolumne gibt es dazu einen Erfahrungsbericht über das Clustern mit Tomcat in einem realen Projekt. Aus diesem Projekt entstammen weitere sinnvolle Erweiterungen, die hoffentlich bald in den Tomcat Cluster integriert werden.

Der Tomcat Cluster in Verbindung mit dem Apache-Modul mod_jk kann stabil hohe Lasten verteilen und Ausfälle verbergen. Es soll hier allerdings nicht verschwiegen werden, dass für die Ausfallsicherheit meist ein enormer zusätzlicher Aufwand betrieben werden muss. Nicht jede Ihrer Webanwendungen ist wirklich effektiv verteilbar. Meist fehlen nach der Veränderung der Session-Objekte die Aufrufe der Methoden *setAttribute()* und *removeAttribute()*. Weiterhin werden zu viele Daten mit der Session assoziiert. Ein umfangreiches Refactoring und oftmals ein Redesign der Anwendung sind die Folge. Die Vorbereitung von Installation, Wartung und Überwachung mehrerer Apache- und Tomcat-Server ist eine weitere Herausforderung, die individuell gelöst werden muss. Der Lohn der Mühen ist eine ausfallsichere und hochperformante Tomcat-Cluster-Lösung, die hoffentlich auch bei Ihnen für mehr Kundenzufriedenheit sorgt.

Ich freue mich auf Kommentare und Anregungen – besuchen Sie also die TomC@ Site und das TomC@-Forum oder die Tomcat-Mailing-Listen [5] [6] [7]. ■

Peter Roßbach (pr@objektpark.de) ist als freier J2EE-Systemarchitekt, Entwickler und Trainer tätig. Er ist Committer der Apache Software Foundation und arbeitet aktiv im Projekt Tomcat.

Tomcat-News

- Tomcat 5.5.5 Beta ist veröffentlicht.
- Tomcat 5.0.31 ist veröffentlicht.
- Neues mod_jk 1.2.7: neuer LoadBalancing-Algorithmus mit Unterstützung für mehrere LocalWorker, Domainkonzept für Worker zur Verbesserung der Cluster-Fähigkeit, direkte Worker-Definition in der Apache-Konfigurationsdatei *httpd.conf*, Pattern Matching in JkMount und JKUnMount Mappings, Ausschluss bestimmter URIs von der Weiterleitung an die Tomcat-Server, gesamte Dokumentation überarbeitet.
- Referenzkarte zum Tomcat 5.0.29 auf tomcat.objektpark.org/referenzkarte veröffentlicht.

Links & Literatur

[1] Peter Roßbach: Tomcat als Manager. Sessionmanagement im Tomcat, in *Java Magazin* 1.2005

[2] jakarta.apache.org/tomcat/connectors-doc

[3] jakarta.apache.org/tomcat

[4] Java-Servlet 2.4-Spezifikation: www.jcp.org/en/jsr/detail?id=154, Kapitel 7

[5] www.javamagazin.de/tomcat

[6] www.mail-archive.com/tomcat-user@jakarta.apache.org,

[7] tomcat.objektpark.org