Tomcat als Serviceunternehmen: erfolgreiche Dienstleistung unter Windows

Der feine Unterschied

Gute Dienstleistung hat ihren Wert: Wir liegen gern in der Hängematte in dem Wissen, dass jemand anders sich um alles kümmert. Wenn dieser Dienst nicht so teuer ist, leisten wir uns guten Service gern. Windows hat einen Dienstmanager und warum sollen wir eigentlich nicht unseren Tomcat mit allen Webanwendungen ständig zugreifbar haben? Mit dem Neustart der Maschine alle Webanwendungen direkt wieder verfügbar zu haben, davon träumt so mancher Entwickler und Administrator beim Einsatz von Java-Web-Containern. In dieser Kolumne zeigen wir verschiedene Möglichkeiten für die Serviceintegration des Tomcat in Windows.

Beim Start des Rechners den eigenen Wiki mit Tagebuch, die Bildergalerie oder einfach den letzten Projektstand auf der eigenen Maschine als Referenz zu haben ist verführerisch. Eine echte Integration in das Betriebssystem ist für Produktionsserver unbedingt erforderlich. Schnell ist die Lösung für diese Wünsche gefunden: eine Integration des Tomcat in den Dienstmanager des Betriebssystems Windows. Aber die Schnittstelle ist doch bestimmt nicht direkt von Java zugänglich, oder? Stimmt, aber es gibt nette Windows-Entwickler, die in verschiedenen Projekten entsprechende Programme geschrieben haben, die diese Aufgabe für uns bewältigen. Der Trick ist klar, man kann zwar nur schwer auf das Windows API direkt von Java zugreifen, aber innerhalb eines C-Programms eine virtuelle Java-Maschine zu starten ist leicht möglich.

Seit Geburt von Java ist leider der Aufruf der *main()*-Methode immer noch die einzige Möglichkeit, ein Java-Programm in einer JVM zu starten. Sicherlich eine gute und einfache Möglichkeit für Rich Clients. Für Server mit vielen aktiven Threads und entsprechenden Zuständen für ihre Clients ist die Situation nicht mehr unbedingt so übersichtlich. Gerade Web-Container stellen umfangreiche Anforderungen an ihre JVM-Umgebung. Während der Start- und Stoppphase des Containers müssen Zustände gespeichert, JDBC Connection Pools und Request Threads geeignet geschlossen werden. Der Tomcat öffnet dafür, um einen sauberen Stopp hinzubekommen, beispielsweise einen ServerSocket und wartet auf seinen Shut-down-Befehl. Seit J2SE 1.3.1 wird zusätzlich für das kontrollierte Herunterfahren des Tomcat ein entsprechender Signal-Handler registriert. Das hört sich alles nach einer sehr vernünftigen Lösung an. Nur die direkte Integration in das Betriebssystem fehlt noch. Wer seinen Tomcat also immer verfügbar haben möchte, muss sich in die spezifische Umgebung des jeweiligen Betriebssystems einklinken. Zwei hilfreiche Projekte für die Dienstbereitstellung unter Windows wollen wir in dieser Kolumne vorstellen. Das Jakarta-Commons-Daemon-Projekt ist während des Tomcat-Projekts entstanden [1]. Ein sehr erfolgreiches Open-Source-Projekt für die Serviceintegration ist das Java-Service-Wrapper-Projekt [2]. Auf der Heft-CD befinden sich für diese beiden Servicedienstleister entsprechende Beispiele. Die ausgewählten Kandidaten sind ebenso unter Unix verfügbar.

Jakarta Commons Daemon

Der Daemon besteht aus zwei Teilen. Der eine ist in C geschrieben und sorgt für die Integration in das verwendete Betriebssystem. Der Java-Teil besteht aus einer anpassbaren Schnittstelle zum Start und Stopp des Servers. Für Windows wird das Programm *procrun* verwendet und für Unix kommt *jsvc* zum Einsatz. Es gibt also leider sehr verschiedene Realisierungen und Konfigurationen für Windows und Unix.

In der *jakarta-tomcat.5.0.x.exe*-Windows-Distribution ist der Service gleich in den grafischen Installer integriert [3]. Schnell steht damit ein Service *Tomcat5* im eigenen Windows-System bereit. Natürlich kann man diese Serviceintegration

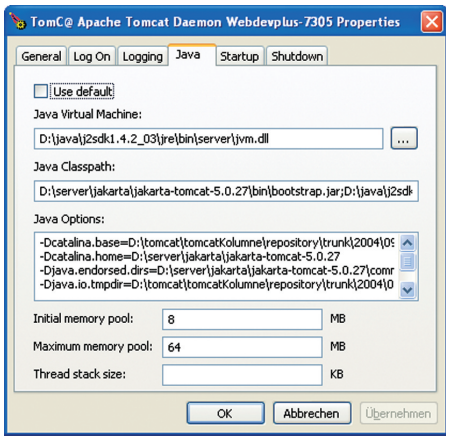

Abb. 1: Jakarta Commons Daemon als Windows Tray Integration

auch zu jedem späteren Zeitpunkt mit dem Skript %CATALINA.HOME%\bin\service.bat erreichen. Mit dem Skript kann man einen vorhandenen Dienst auch wieder entfernen oder die Bezeichnung ändern. Die Konfiguration kann über ein Tray Icon erreicht werden (Abb. 1). Hier können Startoptionen und Parameter eingesehen und verändert werden. Alternativ ist es möglich, mit dem Werkzeug Regedit die Parameter ebenfalls zu verändern oder einem Fehler auf die Spur zu kommen (Abb. 2). Natürlich kann man in Windows in gewohnter Weise mit einem Klick der rechten Maustaste auf das Tray Icon den Service starten bzw. stoppen.

Leider kann das Tomcat-Service-Skript keine Catalina-Base-Installation als Service verwalten [4], denn mit dem Skript *service.bat* kann man nur verschiedene komplette Tomcat-Distributionen als Service integrieren. Dies ist aber keine grundsätzliches Hürde. Einige kleine Änderungen am vorhandenen Skript führen bereits zum Erfolg (Heft-CD). Da sich ein Skript meist nur schwer in eine Entwicklungsumgebung integrieren lässt, haben wir uns gleich ein Ant-

Listing 1

Windows Procrun Service als Ant-Skript

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<project name="Procrun Windows Daemon"
                                default="install" basedir="." >
 <property file="build.properties" />
 <target name="install">
  <echo>Installing Tomcat service...</echo>
<!--1-->
  <exec dir="${catalina.home}\bin"

executable="${catalina.home}\bin\${catalina.executable}"
   failifexecutionfails="true"
   failonerror="true"
   os="Windows NT, Windows 2000, Windows XP"
   output="logs/install.log"
   logerror="true">
   <arg line="//IS//${daemon.app.name}" />
   <arg line="--DisplayName &quot;${daemon.app.
                                  displayname}&quot;" />
   <arg line="--Description &quot;${daemon.app.
                                   description}&quot;" />
   <arg line=
    "--Classpath
&quot;${catalina.java.home}\lib\tools.jar;${catalina.
                          home}\bin\bootstrap.jar&quot;" />
   <arg line="--Jvm &quot;${catalina.java.home}
                           \jre\bin\server\jvm.dll&quot;" />
   <arg line="--JavaHome
&quot;${catalina.java.home}&quot;" />
   <arg line=
    "--StartClass &quot;org.apache.catalina.startup.
                                    Bootstrap&quot;" />
   <arg line="--StartParams &quot;start&quot;" />
   <arg line="--StartMode &quot;jvm&quot;" />
   <arg line=
    "--StopClass &quot;org.apache.catalina.startup.
                                    Bootstrap&quot;" />
   <arg line="--StopParams &quot;stop&quot;" />
   <arg line="--StopMode &quot;jvm&quot;" />
  </exec>
  <echo>Setting service parameters...</echo>
<!--2-->
  <exec dir="${catalina.home}\bin"

executable="${catalina.home}\bin\${catalina.executable}"
   failifexecutionfails="true"
   failonerror="true"
   os="Windows NT, Windows 2000, Windows XP"
   output="logs/install.log"
   logerror="true">
   <arg line="//US//${daemon.app.name}" />
   <arg line=
    "--JvmOptions &quot;-Dcatalina.home=${catalina.
                                        home}&quot;" />
   <arg line=
    "++JvmOptions &quot;-Dcatalina.base=${catalina.
                                        base}&quot;" />
   <arg line=
    "++JvmOptions &quot;-Djava.endorsed.dirs=${catalina.
                        home}\common\endorsed&quot;" />
   <arg line=
    "++JvmOptions &quot;-Djava.io.tmpdir=${catalina.
                                     base}\temp&quot;" />
   <!-- BEGIN OF Java Security Manager setup -->
   <arg line="++JvmOptions &quot;-Djava.security.
                                      manager&quot;" />
   <arg line=
    "++JvmOptions &quot;-Djava.security.policy=
          ${catalina.base}\conf\catalina.policy&quot;" />
   <!-- END OF Java Security Manager setup -->
   <!-- ClassLoader Konfiguration -->
   <arg line=
    "++JvmOptions &quot;-Dcatalina.config=file:/${
        catalina.base}/conf/catalina.properties&quot;" />
   <arg line=
    "++JvmOptions &quot;-Xms${jvm.minmemory}m&
         quot;;&quot;-Xmx${jvm.maxmemory}m&quot;" />
   <arg line="--LogPath &quot;${catalina.base}\
                                          logs&quot;" />
   <arg line="--LogLevel &quot;Debug&quot;" />
   <arg line="--StdOutput
&quot;${catalina.base}\logs\stdout.log&quot;" />
   <arg line="--StdError
&quot;${catalina.base}\logs\stderr.log&quot;" />
  </exec>
 </target>

 <target name="uninstall" >
  <echo>Uninstalling Tomcat service...</echo>
  <exec dir="${catalina.home}\bin"

executable="${catalina.home}\bin\${catalina.executable}"
   failifexecutionfails="true"
   os="Windows NT, Windows 2000, Windows XP"
   output="logs/uninstall.log"
   append="true">
  <arg line="//DS//${daemon.app.name}" />
 </target>

 <target name="console" >
  <echo>Console start Tomcat service...</echo>
  <exec dir="${catalina.home}\bin"

executable="${catalina.home}\bin\${catalina.executable}"
   failifexecutionfails="true"
   os="Windows NT, Windows 2000, Windows XP"
   output="logs/console.log"
   append="true">
   <arg line="//TS//${daemon.app.name}" />
  </exec>
 </target>

 <target name="start" >
  <echo>Start Tomcat service...</echo>
<!--3-->
  <exec dir="${catalina.base}\bin"
   executable="cmd.exe"
   failifexecutionfails="true"
   os="Windows NT, Windows 2000, Windows XP">
   <arg line="start /MIN cmd /c start.bat"/>
  </exec>
 </target>

 <target name="stop" >
  <echo>Stop Tomcat service...</echo>
  <exec dir="${catalina.base}\bin"
   executable="cmd.exe"
   failifexecutionfails="true"
   os="Windows NT, Windows 2000, Windows XP">
   <arg line="start /MIN cmd /c stop.bat"/>
  </exec>
 </target>

 <target name="monitor" >
  <echo>Monitor Tomcat service...</echo>
<!--4-->
  <exec dir="${catalina.home}\bin"
   executable="${catalina.home}\bin\tomcat5w.exe"
   os="Windows NT, Windows 2000, Windows XP"
   spawn="true">
   <arg line="//MS//${daemon.app.name}" />
  </exec>
 </target>

</project>
```

Skript für die Serviceaufgabe geschrieben. Ein weiterer Grund für diese Maßnahme ist, dass die aktuelle im Netz vorhandene Commons-Daemon-Dokumentation nicht mit der im Tomcat genutzten Version übereinstimmt. Der Tomcat nutzt seit dem Release 5.0.2x immer den aktuellen CVS Head des Daemon-Projekts. Da sich Parameter und Funktionsweise grundsätzlich geändert haben, waren schon ein paar Experimente notwendig, um die gewünschte Konfiguration für mehrere Tomcat-Services auf Basis einer Tomcat 5.0.27-Installation zu realisieren (Listing 1).

Für eine erfolgreiche Daemon-Installation benötigt man vier Schritte. Als Erstes wird der Service dem Windows-Dienstmanager mitgeteilt (1). Jeder Service benötigt dafür einen eindeutigen Namen. Im nächsten Schritt können verschiedene Optionen des Service gesetzt werden. Hier kann man die Werte mit ++<*Option*> erweitern und mit --<*Option*> ersetzen. Alle Werte müssen hier mit doppelten Hochkommata eingeschlossen notiert werden. Im Ant-Skript erfolgt dies durch den Einschluss mit der XML Entity *"* (2).

Nun kann der Service mit dem Befehl ant *-f daemonservice.xml* start gestartet werden (3). Leider funktionieren die dafür vorgesehenen Parameter *//RS// (run service)* und *//SS// (stop service)* des Programms *%CATALINA.HOME%\bin\tomcat5.exe* (procrun) nicht. Mit dem net-Befehl kann man unter Windows den Service aber dennoch steuern:

```
net start $ServiceName
net stop  $ServiceName
```

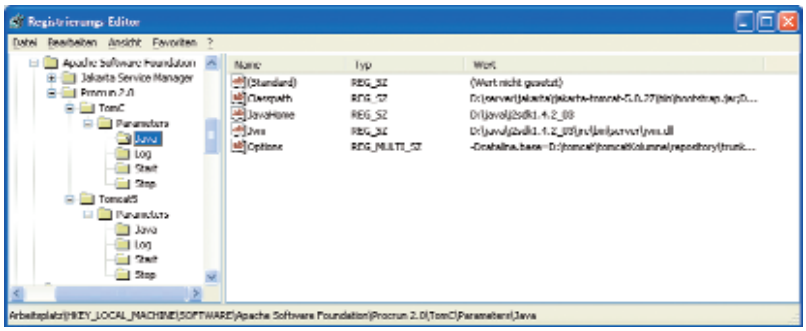

Abb. 2: Jakarta Commons Daemon mit Regedit verändern

Listing 2

```
wrapper.java.command=D:/j2sdk1.4.2_03/bin/java

# Java Main class des Wrappers.
wrapper.java.mainclass=org.tanukisoftware.wrapper.
                                    WrapperStartStopApp

# Java Classpath (include wrapper.jar)
#Hier können beliebig viele Einträge hinzugefügt werden
#Sie müssen nur die Nummer erhöhen
wrapper.java.classpath.1=D:/jakarta-tomcat-5.0.27/bin/
                                    bootstrap.jar
wrapper.java.classpath.2=wrapper/wrapper.jar
wrapper.java.classpath.3=D:/j2sdk1.4.2_03/lib/tools.jar
wrapper.java.classpath.4=D:/jakarta-tomcat-5.0.27/
                                    bin/mx4j-tools.jar

# Java Library Path (Wrapper.DLL oder libwrapper.so)
#Auch hier können weitere native Bibliotheken
                                    hinzugefügt werden
wrapper.java.library.path.1=wrapper

# Java Additional Parameters
wrapper.java.additional.1=-Djava.endorsed.dirs=D:/
                    jakarta-tomcat-5.0.27/common/endorsed
wrapper.java.additional.2=-Dcatalina.home=D:/
                                    jakarta-tomcat-5.0.27
wrapper.java.additional.3=-Dcatalina.base=..
wrapper.java.additional.4=-Djava.io.tmpdir=../temp
wrapper.java.additional.5=-Dcatalina.config=file:../conf/
                                    catalina.properties
wrapper.java.additional.6=-Djava.security.manager
wrapper.java.additional.7=-Djava.security.policy==../
                                    conf/catalina.policy

# Initialer Java Heap Size (in MB)
wrapper.java.initmemory=32

# Maximum Java Heap Size (in MB)
wrapper.java.maxmemory=64

# Start- und Stop-Argumente
wrapper.app.parameter.1=org.apache.catalina.startup.
                                    Bootstrap
wrapper.app.parameter.2=1
wrapper.app.parameter.3=startd
wrapper.app.parameter.4=org.apache.catalina.startup.
                                    Bootstrap
wrapper.app.parameter.5=true
wrapper.app.parameter.6=1
wrapper.app.parameter.7=stopd

# Wrapper Logging Properties
wrapper.console.format=PM

# Console Log Level
#Hier können Sie den LogLevel auf DEBUG stellen, wenn Not
#am Mann ist und Sie den Wrapper über die Konsole starten
wrapper.console.loglevel=INFO

# Log-Datei
#Wenn Sie auch möglichst viele Informationen in der
#Log-Datei haben wollen, können Sie auch hier den
#LogLevel auf DEBUG stellen
wrapper.logfile=../logs/wrapper.log
wrapper.logfile.format=LPTM
wrapper.logfile.loglevel=INFO
wrapper.logfile.maxsize=10m
wrapper.logfile.maxfiles=3
# Syslog Log Level
#Loggt in das SystemLog
wrapper.syslog.loglevel=ERROR

# Wrapper logging Filter
#Falls eine java.lang.OutOfMemoryError-Exception auftritt,
#wird die JVMneu gestartet

wrapper.filter.trigger.1=java.lang.OutOfMemoryError
wrapper.filter.action.1=RESTART

# Wrapper NT Dienst
#Diese Einträge sind unter Unix wirkungslos, können aber
#in der Konfigurationen ohne weiteres enthalten sein

# Name des Dienst
wrapper.ntservice.name=webdevplus-7405

# Anzeigename
wrapper.ntservice.displayname=TomC@ Apache Tomcat
                                    Wrapper Webdevplus-7405

# Beschreibung
wrapper.ntservice.description=TomC@ Apache Tomcat
                    Server - http://tomcat.objektpark.org

# Fügt den Tomcat zur gleichen Gruppe hinzu wie der IIS
wrapper.ntservice.load_order_group=WebServices

# Modus AUTO_START oder DEMAND_START
wrapper.ntservice.starttype=AUTO_START

# Interaktion mit dem Desktop.
wrapper.ntservice.interactive=false
```

Der abschließende vierte Schritt für einen Tomcat Daemon erfolgt durch die Integration des Monitors (4). Hierzu muss das Programm tomcat5w.exe genutzt werden, das zusätzlich den Code für die grafische Konfiguration des Daemon enthält. Eine Integration in den Windows-Tray-Bereich ist damit vollzogen (Abb. 1). Für einen Test kann der Service auch mit dem Ant Target *console*. gestartet werden. Ein einfaches CTRL-C-Signal stoppt den Tomcat-Service in diesem Modus. Mit dem Befehl uninstall kann der Service wieder aus dem Dienstmanager ausgetragen werden. Die Unterschiede zum Originalskript bestehen im Wesentlichen in

Wrapper-Eigenschaften

- Filterung der Ausgabe: Kritische Fehler wie eine *java.lang.OutOfMemory-Exception* können abgefangen werden:

```
wrapper.filter.trigger.1=java.lang.OutOfMemoryError
wrapper.filter.action.1=RESTART
```

- *ThreadDump* während der Laufzeit erzeugen: Bei Problemen ist es interessant zu wissen, was im aktuellen Thread für Informationen verfügbar sind. Das können Sie im laufenden Betrieb realisieren. Dazu müssen der Wrapper im Konsolenmodus gestartet oder der Wrapper als MBean verfügbar sein (s.u.).

```
console.bat
<<Wrapper startet >>
CTRL-BREAK
```

- Rotation der Logdatei: Das Logging-System des Wrapper ist aus Perfomancegründen sehr einfach gehalten. Trotzdem gibt es eine Rotation der Logdatei. Sie können bestimmen, ab welcher Größe eine neue Logdatei erzeugt wird und die Anzahl der Backup bestimmen.

```
wrapper.logfile.maxsize=10m
wrapper.logfile.maxfiles=3
```

- Verwendung des System-Log: Sie können für das Logging auch das System-Log verwenden. Auch dort gibt es die Log-Level *NONE, FATAL, ERROR, DEBUG* und *STATUS*. Benutzen Sie keine zu hohen Log-Levels, da sonst das System-Log recht schnell voll ist.

```
wrapper.syslog.loglevel=INFO
```

- Kontrolle des Wrapper mittels JMX: In unserem Beispiel befindet sich ein *WrapperLifecycleListener*, den Sie in der Tomcat-Konfiguration *conf/server.xml* angeben können. Dieser Listener aktiviert das Wrapper MBean und stellt es jeder MBean-Konsole zur Verfügung. Ab diesem Zeitpunkt können Sie den Wrapper über JMX starten, erneut starten oder einen *ThreadDump* erzeugen (Abb. 5)

```
<Server port="7405" shutdown="SHUTDOWN" debug="0">
  <!— Enable JMX MBeans support —>
  <Listener className="org.apache.catalina.mbeans.ServerLifecycleListener"/>
        <Listener className="org.objektpark.catalina.wrapper.WrapperLifecycleListener"/>
...
</Server>
```

- Ping-Zeiten erweitern: In regelmäßigen Abständen wird die JVM vom Wrapper angepingt. Hiermit prüft der Wrapper-Prozess, ob die JVM (Tomcat) überhaupt noch reagiert. Bei ausgelasteten Systemen kann es schon mal passieren, dass die JVM zu spät antwortet und der Wrapper deswegen irrtümlich annimmt, die JVM sei abgestürzt. Dies kann passieren, wenn z.B. der Garbage Collector zu oft oder zu lange tätig ist. Auch echte CPU-Fresser wie Reportgeneratoren sind so schnell identifiziert und müssen evtl. in andere Subprozesse ausgelagert werden.

```
wrapper.ping.interval=5
wrapper.ping.timeout=30
```

- Bestehende Konfiguration auf anderen Systemen nutzen: Die Datei *wrapper.conf* kann grundsätzlich auf verschiedenen Systemen eingesetzt werden. Diese Systeme müssen noch nicht einmal auf dem gleichen Betriebssystem laufen. Kontrollieren Sie ggfs. Pfadangaben zwischen Windows- und Unix-Systemen. Natürlich müssen in Ihrer Umgebung die passenden Binaries des Wrappers vorliegen.

News über den Tomcat

- Das Release Tomcat 5.0.27 ist fertig
- Neue Bibliotheken: Commons DBCP 1.2.1; Commons Pool 1.2; Commons Logging 1.0.4; Ant 1.6.1
- Clustering: Farm Deployment (Experiment)
- Weitere Details befinden sich unter jakarta.apache.org/tomcat/tomcat-5.0-doc/changelog.html
- Das JK 1.2.6 Connector Release nimmt Gestalt an
- Neue Optionen: CPing/CPong-Erweiterung zur besseren Fehlererkennung; Verbesserung der Recovery-Option für den Betrieb im Cluster
- Aktivitäten zum Release *Tomcat 5.next* sind gestartet
- Logger wird durch den Commons-Logging-Einsatz restlos ersetzt
- Einige Packages sind gewandert: Connector; Commons Digester wird ersetzt durch Tomcat 3.3 Digester-Code (*org.apache.tomcat.util.digester*)
- Reduktion der Commons-Package-Abhängigkeiten soll erreicht werden
- Verbesserungen des Deployer sind geplant
- Noch mehr Management mit JMX soll möglich werden

News über die Centaurus-Platform

- Release 1.0beta-5
- Erstes Release eines neuer Managers für den Tomcat
- Integration der HSQL DB
- Verbesserung von *SecurityWatcher* und *HostCreator*
- Einsatz des Tomcat 5.0.27 mit eigenen Fixes
- Plugin-Konzept in den GUI- und Konsolen-Installer integriert
- GUI-Installation unter Unix und Windows erleichtert
- Integration des Wrapper 3.1.0 für verschiedene Betriebssysteme
- Integrierte HTTP-JMX-Konsole

der Konfiguration der JVM-Speicheroptionen und dass der Service immer mit einem Security-Manager gestartet wird [4]. Die Parameter für einen Service können in der Datei *build.properties* gepflegt werden:

```
server.url=http://localhost:7580/
engine.host=localhost
catalina.home=D:\\jakarta-tomcat-5.0.27
catalina.base=D:\\TomC@-2004-09\\daemon\\
                                    webdev-server
catalina.java.home=D:\\j2sdk1.4.2_03
catalina.executable=tomcat5
daemon.app.name=TomC
daemon.app.displayname=TomC@ Apache Tomcat
                                Daemon Webdevplus-7305
daemon.app.description=TomC@ Apache Tomcat Server -
                                http://tomcat.objektpark.org
jvm.minmemory=8
jvm.maxmemory=32
```

Java Service Wrapper

Im ersten Teil dieser Kolumne haben Sie erfahren, wie Sie Ihren Tomcat mittels Jakarta Commons Deamon in Ihr System integrieren können. Allerdings kommt der Appetit erst mit dem Essen. Der Jakarta Commons Daemon hat einige Schwachpunkte, die die Konfiguration und die Verteilung einer Konfiguration auf mehrere Server erschweren:

- Zwischen der Windows- und Unix-Version gibt es Unterschiede in der Installation.
- Die Unix-Version (*jsvc*) muss immer auf dem Zielsystem kompiliert werden.
- Die Syntax der Befehle ist nicht gerade selbsterklärend.
- Unter Windows werden die Konfigurationsparameter in der Registry gespeichert und nicht in einer Properties-Datei gehalten, die einfach von einem System zum anderen kopiert werden kann.
- Eine eigenständige Restart-Kontrolle wird von uns schmerzlich vermisst.

All diese Schwachpunkte verhindern eine problemlose Übertragung einer erfolgreichen Konfiguration auf einem anderen Server. Wenn Ihre Serverlandschaft auch noch aus Windows- und Linux-Servern besteht, dann muss auf jeden Fall ein Produkt mit einfacherer Struktur genutzt werden. Ein solches Produkt ist das Java-Service-Wrapper-Projekt [2]. Der Wrapper unterstützt jegliche Art von Java-Anwendungen und enthält einige Eigenschaften, die Sie beim Jakarta Commons Daemon vermissen werden:

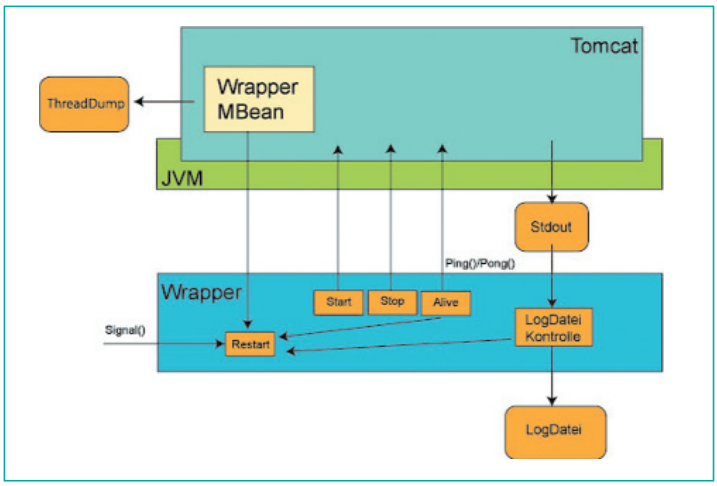

Abb. 3: Service-Wrapper-Architektur

- Der Wrapper selber kann als MBean zur Verfügung gestellt werden.
- Der Wrapper hat die Möglichkeit, die JVM neu zu starten.
- Wenn die JVM hängt oder gar abgestürzt ist, kann der Wrapper diese selbstständig neu starten.
- Die Ausgabe des Wrapper kann nach bestimmten Zeichenfolgen gefiltert werden und eine benutzerdefinierte Aktion, beispielsweise ein Neustart, erfolgen.
- Die Konfiguration erfolgt über eine Properties-Datei und kann ohne große Änderung zwischen verschiedenen Systemen bzw. Betriebssystemen installiert werden.
- Für nahezu alle Betriebssysteme gibt es eine Binärversion.
- Die Dokumentation ist gut strukturiert, ausführlich und immer auf dem aktuellen Stand des Projekts.

Aufbau des Wrapper

Auch der Wrapper besteht aus zwei Teilen. Der native C-Teil besteht aus einer Bibliothek (*wrapper.dll* unter Windows; *libwrapper.so* unter Unix) und einer ausführbaren Datei (*wrapper.exe* bzw. *wrapper*). Der Java-Teil wird in einem Archiv wrapper.jar geliefert. Diese beiden Teile werden durch eine Konfigurationsdatei *wrapper.conf* miteinander verbunden. Dieser Aufbau ermöglicht den Start beliebiger Java-Anwendungen unter der Kontrolle des Wrapper. Es gibt verschiedene Arten der Integration, die eine Java-Anwendung hierfür nutzen kann. Der Wrapper ist eine eigenständige Anwendung und die JVM wird separat gestartet. Dadurch ist diese Lösung sehr stabil und in der Lage die JVM wirklich zu kontrollieren und bei Bedarf neu zu starten.

Auf der Heft-CD finden Sie ein lauffähiges Beispiel. Installieren Sie das Beispiel aus dem Verzeichnis *windows/wrapper*. Wie gewohnt könnten Sie das Beispiel an Ihre Systeminstallation anpassen; es ist auch mit älteren Tomcat 5.0.2x-Versionen problemlos lauffähig. Mit Änderungen der Tomcat-Serverkonfiguration sollte es Ihnen auch schnell gelingen, eine eigene Anpassung für einen Tomcat 4 zu gestalten.

Kontrolle der Anwendung

Nach der erfolgreichen Installation unseres Beispiels müssen Sie noch den Service installieren. Die Skripte dafür finden Sie im Verzeichnis bin. Das Skript *install.bat*

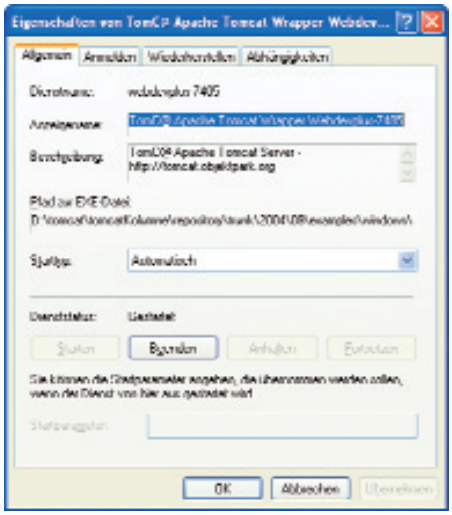

Abb. 4: Wrapper im Windows-Dienstmanager

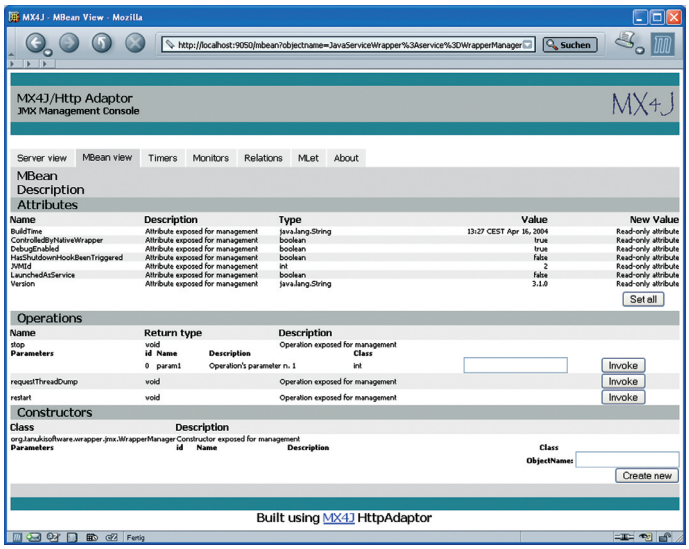

Abb. 5: Wrapper-JMX-Integration via MX4J-HttpAdaptor

installiert den Dienst; das Skript *uninstall.bat* entfernt es wieder. Kontrollieren können Sie das beschriebene Verhalten über den Windows-Dienstmanager. Über diesen können Sie Ihren Tomcat-Service in gewohnter Weise starten und stoppen. Natürlich können Sie den Start und Stopp des Tomcat auch über die entsprechenden Skripte im *bin*-Verzeichnis realisieren. Dort haben Sie sogar die Möglichkeit, den Wrapper im Konsolenmodus auszuführen, d.h., die Ausgabe des Wrapper erscheint zusätzlich auf der Konsole.

Konfiguration

Die Konfiguration des Wrapper erfolgt ausschließlich in der Datei *wrapper.conf*. Diese Datei befindet sich im Verzeichnis *conf*. Die Konfiguration hat ein einfaches Format, wie Sie es auch von den Java-Properties-Dateien gewohnt sind. In Listing 2 sehen Sie eine Beispielkonfiguration mit einigen Anmerkungen. Die vollständige Dokumentation aller möglichen Parameter finden Sie unter [5]. Wie Sie an diesem Beispiel erkennen können, ist es sehr einfach, auch komplexere Anwendungen mit dem Wrapper zu starten. Die Konfiguration unterstützt beliebig viele Einträge für die Klassenpfade und JVM-Parameter. Die einzige Voraussetzung, die Ihre Anwendung haben muss, ist, dass eine Main-Methode für Start und Stopp existiert.

Gegenüber dem Jakarta Commons Daemon ist es möglich, den Windows-Dienst umfangreicher zu konfigurieren. Neben dem Namen und einer Beschreibung des Dienstes können Sie Abhängigkeiten zu anderen Diensten, die Gruppe, zu der dieser Dienst gehört, und das Benutzerkonto für die Ausführung bestimmen.

Die bisher beschriebene Funktionalität unterscheidet sich noch nicht so sehr vom Jakarta Commons Daemon. Das wird sich jetzt ändern. Der Wrapper hat einige – für Systemdienste – eminent wichtige Eigenschaften, die wir im Kasten „Wrapper-Eigenschaften“ aufgelistet haben.

Weitere Servicealternativen

Die beschriebenen Lösungen eignen sich bereits hervorragend für die Windows-Service-Integration. Zwei weitere Projekte, die jeweils unter dem Namen Tomcat Service Manager veröffentlicht wurden, sind interessant. Das Projekt „tcsm“ kann gleich mehrere Tomcat-Services zusammenhängend verwalten [6]. Der Sourcecode steht in diesem Projekt zur Verfügung. Das Projekt tcservcfg zeichnet sich durch eine umfangreiche Dokumentation zum Thema Tomcat als Service aus [7]. Besonders ausgiebig wird auf wichtige Informationen für den Betrieb eines sichereren Service mit Tomcat eingegangen [8]. Weitere nützliche Informationen, um einen Tomcat mit mehr Sicherheit unter Windows zu nutzen, stehen im Buch „Professional Apache Tomcat 5“ [9]. Einen Tomcat-Service auf einem Produktionsserver zu betreiben, erfordert mehr von dem Administrator, als nur ein Tomcat-Release auf der Maschine zu installieren und ein paar Anwendungen in das *webapps*-Verzeichnis einzuspielen.

Fazit

Eine Serviceintegration ist mit den beschriebenen Projekten einfach und schnell möglich. Der Tomcat-Standard ist der Jakarta Commons Daemon. Die gute Integration in das Windows-Betriebssystem und die einfache Konfiguration rechtfertigen diese Wahl. Allerdings überzeugen die nicht verfügbare Dokumentation, das fehlende offizielle Release, die Unterschiedlichkeit der Konfiguration für verschiedenen Betriebssysteme und das eingeschränkte *service.bat*-Skript nicht. Hingegen kommt der Java Service Wrapper zwar mit keiner Oberfläche, dafür überzeugt der reichhaltige Funktionsumfang nachhaltig. Es ist kein Wunder, dass mittlerweile viele Open-Source-Projekte wie Avalon, JOnAS, JBoss, Jetty, James oder die Centaurus-Platform [10] diesen Service-Integrator nutzen.

Wir freuen uns auf Kommentare und Anregungen – besuchen Sie also meine TomC@ Site [11] und das TomC@-Forum [12] oder die Tomcat Mailing-Listen [13]. ■

Peter Roßbach (pr@objektpark.de) ist als freier J2EE-Systemarchitekt, Entwickler und Trainer tätig. Thorsten Kamann (thorsten.kamann@planetes.de) ist als Softwareentwickler und Administrator für Win 32- und Linux-Systeme tätig.

Links & Literatur

[1] jakarta.apache.org/commons/daemon/

[2] wrapper.tanukisoftware.org/doc/english/

[3] jakarta.apache.org/tomcat/

[4] Peter Roßbach, Lars Röwekamp: Die Saat liegt im Feld. Catalina.Base: Eine praktische Anleitung zur flexiblen Tomcat-Konfiguration, *in Java Magazin* 6.2004

[5] wrapper.tanukisoftware.org/doc/english/properties.html

[6] www.daveoxley.co.uk/tcsm/

[7] web.bvu.edu/staff/david/tcservcfg/

[8] Peter Roßbach, Lars Röwekamp, Thorsten Kamann: Tomcat als Trutzburg: Sicherheit für Leib und Leben – mein Tomcat ist eine Festung, in *Java Magazin* 4.2004

[9] Viveck Chopra et al.: Professional Apache Tomcat 5, Wrox, 2004

[10] centaurus.sourceforge.net/

[11] tomcat.objektpark.org/

[12] www.javamagazin.de/tomcat/

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